Parallel and distributed algorithms for computational fluid flow simulations

Vu, Thanh Tung

2018

Vu, T. T. (2018). Parallel and distributed algorithms for computational fluid flow simulations. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73789>

<https://doi.org/10.32657/10356/73789>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**PARALLEL AND DISTRIBUTED ALGORITHMS
FOR
COMPUTATIONAL FLUID FLOW SIMULATIONS**

VU THANH TUNG

**INTERDISCIPLINARY GRADUATE SCHOOL
NANYANG ENVIRONMENT & WATER RESEARCH
INSTITUTE (NEWRI)**

2017

**PARALLEL AND DISTRIBUTED ALGORITHMS
FOR
COMPUTATIONAL FLUID FLOW SIMULATIONS**

VU THANH TUNG

Interdisciplinary Graduate School
Nanyang Environment & Water Research Institute
(NEWRI)

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

2017

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

21 August 2017

Vu Thanh Tung

.....
Date

.....
Student Name

Acknowledgements

First of all, I would like to express my special appreciation and thanks to my supervisor Assoc. Prof. Law Wing-Keung Adrian, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advices on both research and my career have been priceless. You are also a great friend to me, who not only always accompany me in this long rough journey but also truly understand my research life. I could not have imagined having a better advisor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Assoc. Prof. Irvine Kim Neil and Prof. Ng Wun Jern for their insightful comments and encouragement, but also for the hard question which helped me widen my research from various perspectives.

Dozens of people have helped and taught me immensely during my study. I especially wish to express my gratitude to Mr. Tien Nguyen Hoanh for sharing with me his knowledge and vision of parallel computing and Mr. Alvin Chew Wei Ze for guiding me through the physical part in my work. I would also like to express my appreciation and thanks to Assoc Prof. Chua Hock Chye Lloyd and Dr. Nguyen Khac Tien Phuoc for their willingness to educate, guide and supervise me when I first started my study in graduate school. Without them, things could have been much more difficult for me when I came into a completely new scientific field like computational fluid modelling.

I also wish to thank NTU, IGS and NEWRI (DHI-NTU/EPMC) for granting me the financial support and opportunity to study and carry out research in this excellent academic environment.

Finally, I would like to express my heartfelt gratitude to my family and friends, especially to my parents and to my partner Huong for their continuous and unconditional support and love. That was a great support for me, which enabled me to complete my study.

Table of Contents

ACKNOWLEDGEMENTS	I
TABLE OF CONTENTS	III
SUMMARY	VI
LIST OF PUBLICATIONS	VIII
LIST OF TABLES	X
LIST OF FIGURES	XII
NOTATIONS	XVI
CHAPTER 1	1
INTRODUCTION	1
1.1 MOTIVATION	1
1.2 FOCUS OF THE WORK	2
1.3 CONTRIBUTIONS RESULTING FROM THE RESEARCH WORK	2
1.4 ORGANIZATION OF THE THESIS	3
CHAPTER 2	5
TWO-DIMENSIONAL HYDRODYNAMIC MODELING OF FLOOD INUNDATION IN MEKONG RIVER WITH TELEMAC-2D	5
2.1 INTRODUCTION	6
2.1.1 Model area	8
2.1.2 TELEMAC-2D model	11

2.2	METHODOLOGY	12
2.2.1	Model developments	12
2.2.2	Validation of the simulation results	14
2.3	RESULTS AND DISCUSSIONS	17
2.3.1	Simulation results	17
2.3.2	Model verification with MODIS satellite images	19
2.3.3	Model computing time with multi-core processor	24
CHAPTER 3		26
COMPUTATIONAL HYDRAULIC MODELLING WITH UPC ARCHITECTURE FOR LARGE SCALE SIMULATIONS		26
3.1	INTRODUCTION	27
3.2	NUMERICAL METHOD	30
3.2	UPC IMPLEMENTATION OF SWES	32
3.4	MODEL VERIFICATION AND PERFORMANCE EVALUATION	36
CHAPTER 4		49
COMPUTATIONAL HYDRODYNAMICS WITH UPC ARCHITECTURE		49
4.1	INTRODUCTION	50
4.2	NUMERICAL DISCRETISATION	53
4.3	DEVELOPMENT OF UPC-CHD	56
4.3.1	Computational structure	56
4.3.2	Domain decomposition algorithm	58
4.3.3	Data storage algorithm	59
4.3.4	Work-sharing function	59
4.4	MODEL VERIFICATION AND PERFORMANCE EVALUATION OF UPC-CHD	60
CHAPTER 5		71
UPC ARCHITECTURE FOR LATTICE BOLTZMANN MODEL		71
5.1	INTRODUCTION	72
5.2	LATTICE BOLTZMANN METHOD	75
5.3	UPC ARCHITECTURE	77
5.3.1	Computational structure	78
5.3.2	Work-sharing function	79
5.3.3	Domain decomposition algorithm	81

5.3.4	Data storage algorithm	82
5.4	RESULTS AND DISCUSSION	84
CHAPTER 6		88
CONCLUSIONS AND OUTLOOK		88
6.1	CONCLUSIONS	88
6.2	FUTURE WORK	91
REFERENCES		93
APPENDIX A		105
APPENDIX B		108

Summary

Hydraulic and hydrodynamic numerical models have become critical tools in doing research, forecasting and management of fluid flows. Studies so far show that multiple-dimensional physically based models that utilize the fully dynamic solution of the Navier-Stokes equation are the most appropriate choice for the computational simulations with high accuracy, stability and reliability. The utilization of this type of model requires the handling of an enormous amount of information, which normally involves the hydrological (spatial-temporal) characteristics, regional geographical information as well as the characteristics of the drainage system. Therefore, when the volume of input data increases significantly (e.g. the topographic data have higher resolutions), the excessively long simulation time becomes a major obstacle. Thus, distributed computing architectures have been explored to parallelize the computation scheme to speed up the simulation time for modelling. At present, three parallel computing architectures, namely the Message Passing Interface (MPI), OpenMP and GPU CUDA are used extensively.

This work proposes a new parallel approach for fluid flow simulations using the Partitioned Global Address Space (PGAS) architecture with Unified Parallel C (UPC) as the implementation solution. The new UPC model combines the benefits of the locality in shared memory architecture of OpenMP and data layout control of MPI. It also maintains the relative ease of programming and implementation works for the possibility of hybrid CPU-GPU architecture development in the future. UPC utilizes the advantage of optimized privatization in the data memory management to accelerate the communication processes. The algorithms of UPC model derived in

this study, including domain discretisation and data distribution, are all designed to capitalize on this key concept. The application-customized parallel computational structure for each numerical scheme is also highlighted. Furthermore, a load-balancing concept, which is essential for massive parallel approaches, is analyzed and described in detail.

The new approach is then customized for different numerical schemes in three fluid flow simulation cases: (1) flood modelling using a second-order Godunov-type monotone upstream scheme with second-order accuracy; (2) computational fluid dynamic (CFD) modelling for laminar flows using the 2-step explicit numerical scheme from the Lax-Wendroff family of predictors and correctors and (3) Lattice Boltzmann method for laminar flows with two dimensions and nine possible directions for streaming processes.

In this thesis, after describing the mathematical formulation of an incompressible Newtonian fluid flow based on the Navier- Stokes equations, the standard numerical methods as well as alternative Lattice Boltzmann approaches are discussed. The verification for the models' accuracy verification is then presented, which show the excellent agreement of the simulation results and reference values for the simulation cases. The computational efficiency of the UPC models is demonstrated using a shared- and distributed-memory system. The results show that the speedup is significantly enhanced compared to MPI and OpenMP models in all cases.

This work highlights potentials to efficiently utilize methods from computer science – especially from the field of high-performance computing to improve solutions from engineering-based domains to facilitate the treatment of complex problems. The synergistic effects between the two disciplines are obvious and promising.

List of Publications

Book Chapter

1. Tung T. Vu, Alvin W.Z. Chew and Adrian W.K. Law (2017).
"UPC Architecture for High Performance Computational Hydrodynamics".
Advances in Hydroinformatics - SimHydro 2017.

Journal Papers

1. Tung T. Vu and Adrian W.K. Law (2017). "UPC Architecture for Lattice Boltzmann Model". Under preparation.
2. Tung T. Vu, Alvin W.Z. Chew and Adrian W.K. Law (2017). "UPC Architecture for High Performance Computational Hydrodynamics". Under consideration.
3. Tung T. Vu, Adrian W.K. Law and Tien H. Nguyen (2017). "Computational Hydraulic Modelling with UPC Architecture for Large Scale Simulation". Submitted to Journal of Computing in Civil Engineering.
4. Tung T. Vu, Phuoc K.T. Nguyen, Lloyd H.C. Chua and Adrian W.K. Law (2015). "Two-Dimensional Hydrodynamic Modelling of Flood Inundation for a Part of the Mekong River with TELEMAC-2D". British Journal of Environment and Climate Change, 5(2), 162-175.

Conference Presentations and Proceedings

1. Tung T. Vu, Alvin W.Z. Chew, and Adrian W.K. Law (2017). "Computational Hydrodynamics Simulations with UPC parallel Architecture", 37th IAHR World Congress, Kuala Lumpur, Malaysia.
2. Tung T. Vu, Alvin W.Z. Chew, and Adrian W.K. Law (2017). "UPC Architecture for High Performance Computational Hydrodynamics", SimHydro: Choosing the right model in applied hydraulics, Sophia Antipolis, Nice, France.
3. Tung T. Vu, Adrian W.K. Law and Tien H. Nguyen (2016). "Computational Hydraulic Modelling with UPC Architecture", The 7th International Conference on Computational Methods – ICCM 2016, Berkeley, California, United States.
4. Tung T. Vu and Adrian W.K. Law (2016). "Development of Expert Modelling Tools for Applications in the Environmental and Water Industry", ASEAN-EU Science, Technology and Innovation days, Hanoi Vietnam.
5. Tung T. Vu, Lloyd H.C. Chua and Adrian W.K. Law (2015). "Flood Inundation Modelling in Mekong River and Quang Ngai Province with TELEMAC2D", International Expert Workshop on Toward Urban Water Security in Southeast Asia: Managing the Risk of Extreme Events, Phnom Penh, Cambodia.
6. Tung T. Vu, Phuoc K.T. Nguyen, Lloyd H.C. Chua and Adrian W.K. Law (2015). "Two-dimensional Hydrodynamic Modelling of Flood Inundation in Mekong", 19th Congress of the Asia and Pacific Division of the International Association for Hydro-Environment Engineering and Research – 19th IAHR, Hanoi, Vietnam.

List of Tables

Table	Title	Page
2.1	Percentage of agreement (defined by different ranges of water depths) between inundated areas in model and satellite image for various periods	20
3.1	Computer clusters used for model testing	38
3.2	Run time and relative speedup among UPC, MPI and OpenMP for 1D dam-break	39
3.3	Run time and relative speedup of UPC for inclined plane case using the SGI- UV server	43
3.4	Run time and relative speedup of UPC, MPI and OpenMP for the 2D case with the SGI server	46
3.5	Run time and relative speedup of UPC, MPI and Amdahl law for the 2D case with the IBM server	47
4.1	Physical dimensions and initial conditions of deployed numerical domains for validating implemented numerical cases in UPC-CHD (Cases A to C)	61

4.2	Physical dimensions and initial conditions of deployed numerical domains for evaluating computational parallelism efficiency of UPC, OpenMP and MPI in UPC-CHD (Cases A to C)	63
4.3	SGI UV-2000 cluster used for model testing	64
4.4	Comparison of run-time and speedup among UPC, MPI and OpenMP for Blasius boundary layer flow (Case A) at varying number of computer cores	65
4.5	Comparison of run-time and speedup among UPC, MPI and OpenMP for Poiseuille's flow (Case B) at varying number of computer cores	65
4.6	Comparison of run-time and speedup between UPC-A and UPC-NA for Couette's flow (Case C) at varying number of computer cores	68
5.1	SGI UV-2000 cluster used for model testing	86

List of Figures

Figure	Title	Page
2.1	Hydrographic Map of the Mekong Basin (Source: MRC, 2005)	9
2.2	The maximum flood extent in the historical 2000 flood event (MRC, 2011)	10
2.3	A 3D view of the mesh associated with the land surface elevation and bathymetry for the flood plain and river between Kratie and Kampong Cham	13
2.4	NDWI Determination in ArcGIS	15
2.5	Model results (Water Depth>0.5)	16
2.6	Observations using satellite image data (NDWI > -0.3)	16
2.7	Superimposed Layers of NDWI > -0.3, Model > 0.5	17
2.8	Flow velocity for (a) pre-flood (10 th June), (b) peak-flood (21 st August) and (c) post-flood (25 th November) periods during the wet season in 2001	18

2.9	Flood inundation identified by both satellite image and model for (a) 10 June 2001, (b) 21 August 2001, and (c) 25 November 2001	21
2.10	Number of flooded cells in model (water depth>0) and satellite images	23
2.11	Number of flooded cells in model (water depth>0.3) and satellite images	23
2.12	Number of flooded cells in model (water depth>0.5) and satellite images	24
2.13	Computing time of the simulation with 1, 2, 4 and 6 cores	25
3.1	Computational structure of ParaFlood2D	33
3.2	Sub-domains division	34
3.3	Comparison of numerical results and analytical solution at t = (a) 2.5s and (b) 6.3s	37
3.4	Speedup for the one-dimensional dam break simulations	40
3.5	Comparison of numerical results and analytical solution at t = (a) 0.1s and (b) 0.3s	41
3.6	Numerical results with the Manning's coefficient of n = (a) 0.035 and (b) 0.075	42
3.7	Initial water depth for the two-dimensional flood wave propagation case	44
3.8	Evolution of fluxes in the x-direction at t = 2s	44
3.9	Comparison of UPC, MPI and OpenMP with the shared-memory machine	46

3.10	Comparison of UPC, MPI and Amdahl Law with the distributed-memory machine	48
4.1	Representative control volume of a single node for discretizing Equation 1	54
4.2	(a) Numerical domain for Blasius boundary layer flow	54
	(b) Numerical domain for Poiseuille's flow	
	(c) Numerical domain for Couette's flow	
4.3	Parallel computational structure for sub-loops of UPC-CHD	57
4.4	Domain decomposition algorithm for a 8 x 8 numerical domain example with (left) 4 threads and (right) 3 threads	58
4.5	Sub-domains division and data storage algorithm for a n x n numerical domain example	59
4.6	(a) Comparison between numerical predictions and analytical solutions for Blasius boundary layer flow at location $x = 0.2m$ (Case A)	62
	(b) Comparison between numerical predictions and analytical solutions for Poiseuille's plate flow at location $x = 0.5m$ (Case B)	
	(c) Comparison between numerical predictions and analytical solutions for Couette's flow at location $x = 0.5m$ (Case C)	
4.7	Comparison of speedup among OpenMP, MPI and UPC for Blasius boundary layer flow (Case A) at varying number of computer cores	67

4.8	Comparison of speedup among OpenMP, MPI and UPC for Poiseuille's flow (Case B) at varying number of computer cores	67
4.9	Comparison of speedup between UPC-A (optimized UPC) and UPC-NA (non-optimized UPC) for Couette's flow (Case C) at varying number of computer cores	69
4.10	Schematic representation of UPC-NA (left) and UPC-A (right) concepts; T_i refers to thread i , D_i refers to domain i .	70
5.1	UPC-LBM computational structure	79
5.2	Illustration for $\text{thread}_{\text{start}}$ and $\text{thread}_{\text{end}}$ in UPC-LBM	80
5.3	Example of domain decomposition on a 5x9 2D domain with 3 threads	81
5.4	UPC memory structure	82
5.5	Sub-domains division and data storage algorithm for numerical domain (Vu et al., 2016)	83
5.6	Memory allocation for: (i) hybrid UPC-LBM and (ii) shared UPC-LBM	83
5.7	Comparison between numerical predictions and analytical solutions for Poiseuille's flow at location $x = 0.5m$	85
5.8	Comparison of speedup among hybrid UPC-LBM, shared UPC-LBM and OpenLB-Poiseuille MPI	87

List of Abbreviations

BGK	Bhatnagar-Gross-Krook
CFD	Computational Fluid Dynamics
CFF	Computational Fluid Flows
CHD	Computational Hydrodynamics
DEM	Discrete Element Method
FDM	Finite Difference Method
FVM	Finite Volume Method
GPU	Graphic Processing Unit
HPC	High Performance Computing
LBM	Lattice Boltzmann Method
LMB	Lower Mekong Basin
MODIS	Moderate-Resolution Imaging Spectroradiometer
MPI	Message Passing Interface
MRC	Mekong River Commission
NDWI	Normalized Difference Water Index
NS	Navier-Stokes
OpenFOAM	Open Field Operation and Manipulation

PGAS	Partitioned Global Address Space
SPMD	Single-Program Multiple-Data
SWEs	Shallow Water Equations
UPC	Unified Parallel C

Notations

Symbol	Definition
A	Area of the integration element
c	Lattice speed
C	Element boundary
c_D	Bed drag coefficient
$\vec{e}$	Particle velocity vector
f	Particle distribution function
f^0	Maxwell-Boltzmann equilibrium distribution function
f^{eq}	Equilibrium distribution function
F	Tensor of fluxes in the x direction
$F_{\perp}$	Vector of fluxes perpendicular to cell face
g	Gravitational acceleration
G	Tensor of fluxes in the y direction
h	Flow depth
N_c	Number of cores

$\hat{n}$	Outward unit vector normal to the element boundary
R	Matrix of right eigenvectors of Jacobian of $F_{\perp}$
S_0	Bed slope term
S_f	Friction slope term
q	Dimensionless relaxation time
t	Time
T	Number of threads
T'	Temperature
U	Vector of the conserved variables
u	Vertically averaged velocity in x direction
$u_{\perp}$	Vertically averaged velocity perpendicular to cell face
$u_{\parallel}$	Vertically averaged velocity parallel to the cell face
ν	Fluid kinematic viscosity
V	Vector of characteristic variables
v	Vertically averaged velocity in y direction
ω	Weighting factor
x	Spatial coordinate
y	Spatial coordinate
z	Bed elevation
Δ	Finite-difference operator
ζ	Free-surface elevation

ρ	Density
η	Computational coordinate
Λ	Diagonal matrix of eigenvalues of Jacobian of $F_{\perp}$
ξ	Computational coordinate
θ	Angle between the cell face normal vector and the x-axis
Ω	Collision term

Chapter 1

Introduction

1.1 Motivation

Computational fluid flows (CFF) models have garnered increasing significance for design, planning, management of water resources project, and assisting engineers to evaluate and optimize industrial-related applications. Examples included the hydraulic modelling for flood inundation (Gomes Pereira & Wicherson, 1999; Hervouet, 2000; Rabus et al., 2003; Hunter et al., 2007) and computational fluid dynamics simulations (Li et al., 2011; Jajcevic et al., 2013; Bucs et al., 2014; Sousa et al., 2014; Sobieski and Zhang, 2017). In these CFF models, various numerical schemes are implemented to discretize and resolve the Navier-Stokes (NS) or Shallow Water equations with the specific level of accuracy desired. Concurrently, the computation needs to be accomplished within a reasonable time frame. For large scale CFF simulations, optimization between the two factors is desired. Thus, the use of parallelization programming architectures is now common to accelerate the simulation time.

At present, the parallelization solutions available for CFF modelling are CPU-based message passing interface (MPI), CPU-based OpenMP and GPU-based CUDA architecture. Research efforts are still ongoing in attempts to develop an architecture which achieves an optimal balance between the accuracy of the numerical scheme

and the desired speedup with increasing number of cores, while not compromising on the relative ease in its programming and implementation works.

1.2 Focus of the work

In this study, we examine the overall design approach of a NS-based fluid flow model utilizing the Partitioned Global Address Space (PGAS) architecture with Unified Parallel C (UPC) as the implementation solution. The cornerstones are the application-customized set of algorithms, especially the dedicated UPC data structure which is capable of organizing the distributed data in an embedded locality manner for large scale simulation. Performance measurements as well as validations using standard benchmarks are shown.

The objective of the present work is not to introduce new numerical schemes for highly accurate flow computations, but rather to use standard computational kernels based on the incompressible Navier-Stokes equations with explicit time stepping scheme. In particular, three UPC fluid flow models are developed for the following specific numerical schemes:

- The second-order Godunov-type monotone upstream scheme for conservation law with second-order accuracy for flood modelling;
- The 2-step explicit numerical scheme from the Lax-Wendroff family of predictors and correctors for CFD simulations of laminar flows.
- The D2Q9 Lattice Boltzmann method for CFD simulations of laminar flows.

For each numerical scheme, the UPC architecture and its algorithms are introduced and explained in detail, which would also pave the way for future implementations of higher-order methods using the aforementioned highly efficient data management approach.

1.3 Contributions resulting from the research work

In this thesis, the following contributions are made:

- A study on the accuracy and performance of the existing flood model, namely

TELEMAC2D, has been made to access the necessity of an alternative parallel computing architecture for massive scale computational fluid flow simulations.

- A new parallel solution using PGAS architecture with UPC as the implementation solutions has been proposed to effectively accelerate the simulation processes of the massive scale computational fluid flows.
- A new flood inundation model, namely ParaFlood2D, has been developed using the second-order Godunov-type MUSCL with second-order accuracy and parallelized using the proposed PGAS-UPC architecture.
- A number of algorithms and techniques have been developed to customize the original proposed PGAS-UPC concept for a 2-step explicit scheme from Lax-Wendroff family of predictors-correctors to form a new computational hydrodynamics model, UPC-CHD.
- An efficient parallel architecture, namely UPC-LBM, has been designed to simulate the high resolution D2Q9 Lattice Boltzmann flow.
- Various combinations of algorithms and hardware settings for PGAS architecture have been examined to study the natural behavior of UPC language in general and the viability of applying it into the computational fluid flow simulation in particular.

1.4 Organization of the thesis

This report is organized into six chapters. Chapter 1 presents the background, objectives and the scope of the work done. The major contributions resulting from this research work are then presented in Chapters 2 to 5 as follow:

i) Chapter 2 introduces the development of the two-dimensional hydrodynamic model TELEMAC-2D for the flood simulation on the Mekong River. The focus of this work is to access the adequacy of the existing hydrodynamic model in flood simulations with a focus on the run time reduction through parallelization.

ii) Chapter 3 describes the development processes of ParaFlood2D, a new flood model using PGAS-UPC architecture. The UPC architecture of ParaFlood2D is designed for the second-order Godunov-type MUSCL scheme with second-order

accuracy. This chapter also presents the method for evaluating the computational efficiency of the new flood model by comparing the performance of UPC with OpenMP and MPI versions.

iii) Chapter 4 discusses the extensions of PGAS-UPC architecture into the new field of the large-scale computational hydrodynamics simulations. The UPC approach is coupled with the traditional 2-step explicit scheme from the Lax-Wendroff family of predictors-correctors to simulate various flow cases. The comparison between different combinations of UPC algorithms is also presented.

iv) Chapter 5 examines the viability of UPC in flow simulations using Lattice Boltzmann method. The UPC-LBM model is developed using D2Q9 setting and the accuracy of UPC-LBM is investigated using the incompressible, viscous Poiseuille's flow.

The overview structure of the main chapters in this thesis is outlined in Figure 1.1.

Finally, Chapter 6 summarizes the major findings and accomplishments during the course of this study together with a recommendation for future works. Some theoretical derivations and examination of classification scheme are included in the Appendices.

Figure 1.1. The thesis structures

Chapter 2

Two-dimensional hydrodynamic modeling of flood inundation in Mekong River with TELEMAC-2D

The content of this chapter is extracted from a research article*, which presents a study on the development of a 2-dimensional (2D) hydrodynamic model based on TELEMAC-2D for the flood simulation of the river from Kratie to Kampong Cham in Cambodia, a part of the Mekong River. The motivation behind the research was to study the feasibility of TELEMAC-2D in flood forecasting, and specifically to determine its adequacy in flood simulations with a focus on the reduction in model run-time through parallelization. An actual flood event that occurred between June to November in 2001 for the stretch of the Mekong River from Kratie to Kampong Cham, was simulated. The model simulations were compared with MODIS satellite Images for specific days in the pre-, peak- and post-flood period. It was found that the 2D simulation results were consistent with observations from satellite imaging, especially during the peak-flood period, there was high percentage (> 90%) match between the simulation results and observation obtained from satellite images while the match was below 50% for the pre- and post- flood periods. The discrepancy at pre- and post-flood may be due to the fact that (i) the model takes into account only hydrodynamic processes of flows in the river and flood plain, it does not consider

* The material presented in this chapter has been published in British Journal of Environment and Climate Change, doi: 10.9734/BJECC/2015/12885

other hydrological processes such as infiltration or evaporation which may be important during the pre- and post- flood periods, and (ii) the resolution of MODIS satellite image at 500m x 500m may be too coarse and therefore not sufficient to identify flooded areas when the area is small or water depth low. Finally, it was found that the computing time could be reduced significantly with parallelization using multi-core processors, albeit with lesser advantage in speedup when the number of cores increased beyond 4.

2.1 Introduction

The Mekong River is the 7th longest river in Asia, defined by its length and discharge (Liu et al., 2009). It has a huge impact to the livelihoods of the people of Cambodia, Laos and Southern Vietnam, which rely heavily on it for socio-economic benefits arising from fishery and agricultural farming activities. Flooding in the Mekong River generally occurs during the summer flood season (i.e. from May to October) due to overflows from the river branches and channels that spread into flood plains. Additionally, the low elevation of the floodplain allows the river to be more prone to overflow from the increased water levels. Tidal motion of the sea may also cause exacerbate the flooding problem. In 2000, it was discovered that tidal backwater obstructed the draining of high water levels in the Mekong Delta, inevitably resulting in floods (MRC, 2005). Annual floods with normal magnitude are known to bring benefits to the agriculture activities and provide water resources and nutrient rich sediments. Floods therefore provide abundant benefits to the riparian countries of Lao-PDR, Cambodia, Thailand and Vietnam, in terms of agriculture, fisheries, and the creation and maintenance of wetlands. However, when extreme floods occur, whereby water levels exceed the critical beneficial level, and flooding is prolonged, this results in damage to land and property, as well as loss of lives. One of the worst flood events in the Mekong Basin occurred in the year 2000, when flood levels exceeded more than 3 meters. Over 600 communities throughout the river basins were inundated impacted, with 11 million people affected by the flood (Nguyen et al., 2003). The flood also resulted in more than 800 human casualties (Plate, 2007). Around 160 thousand hectares (ha) of rice crops were inundated and had to be

harvested immediately and more than 55 thousand ha of rice crops were completely destroyed (Tinh & Hang., 2003). Therefore, studies on flood inundation with numerical models can provide useful information that allows efficient flood management for the mitigation of flood damage.

In principle, overland flows caused by over flows from river branches and canals and spread widely on flood plains are usually considered as shallow water waves which can be described mathematically by the 2-dimentional (2D) Saint-Venant equations. In this study the Saint-Venant equations (also Shallow Water Equations) were solved by the Finite Element Method in TELEMAC-2D (Hervouet, 2007), an open source hydrodynamic modeling software. The implementation of a 2D hydrodynamic numerical model for a large river system and flood plains based on the shallow water equations is data intensive and requires extensive computing facilities. Normally, in order to provide an accurate assessment of the development of a flood, the numerical models have to handle an enormous amount of information, typically involving the spatial-temporal characteristics of flood levels, regional geography, as well as infrastructure such as buildings and the drainage system. The volume of information and the computational complexity also increase when a large area is to be modelled or when a higher resolution is needed. Indeed, a flood modelling system which is deployed by sequential simulations is unlikely to be performed completely in a reasonable timeframe. In the present study, the TELEMAC-2D software was employed using a parallel processing approach based on the domain decomposition method in combination with MPI protocol. This decreased the simulation time on multi-core processor computers (Hervouet, 2000).

In this chapter, we first provide the background information about the model area as well as the TELEMAC-2D model, and then present the methodology for developing the 2D hydrodynamic model to investigate the flood event in 2001 for the model area. We also offer the comparison of the simulation results with satellite images of the inundation. Finally, we discuss the efficiency and stability for running TELEMAC-2D model.

2.1.1 Model area

The Mekong River can be divided into two parts – 1) The Upper Basin, also known as Lancang Jiang, located in China and 2) the Lower Mekong Basin (LMB) from Yunnan in China to the South China Sea. Fig. 2.1 shows the Mekong Basin and its main tributaries. The Upper Basin i.e. the yellow area in Fig. 2.1, only contributes 15 to 20 percent of the total flow in the Mekong River, with a majority of the flow contribution arising from the LMB, shown in blue in Fig. 2.1. The LMB contains two main areas i.e. the middle Mekong (between China and Northern Cambodia) and the Mekong delta (the region of the river before discharge into the South China Sea).

The study area is in Cambodia, from Kratie to Kampong Cham and consists of a single river route between the two locations. According to Mekong River Commission (MRC, 2005), the Cambodian floodplain takes up about 20% of the entire Mekong Basin, approximately 155,000 km² in basin area, and flow in this region contributes about 18% of the total flow in the Mekong River. As the Mekong River reaches Kratie, the topography of the LMB system changes abruptly. Upstream from Kratie, the river flows through mountainous or highland areas with a clearly identifiable mainstream channel. Downstream from Kratie, the terrain in the Cambodia region is flat. The mean annual mainstream flow of the Lower Mekong at Kratie from 1960 to 2004 is estimated to be 13,200 m³/s, 416 cubic kilometers (km³) in terms of volume, or 640 mm in terms of runoff. The average monthly flow in Kratie from 1960 to 2004 peak from June to November is corresponding to the flood season.

Figure 2.1. Hydrographic Map of the Mekong Basin (Source: MRC, 2005)

The hydrodynamics of the river system from Kratie to the river mouth is influenced by three main factors: the inflow from regions upstream from Kratie, the storage of the Great Lake (Tonle Sap in Cambodia) which connects via the Tonle Sap River to the Mekong at its confluence at Phnom Penh, and the tide level at the river mouth in Vietnam. The Great Lake functions as a natural storage reservoir for this region. It stores a large amount of floodwater during the wet season (June to November) which is slowly released during the dry season (December to May), and acts as a crucial source of water supply to the delta. In the Mekong Delta, the river branches are linked together by manmade channels forming an extremely complex network. The tide

effect is significant at stations near the river mouth in Vietnam to Phnom Penh in Cambodia.

This study was carried out as part of a larger study for real-time flood management for the region. The area from Kratie to Kampong Cham, with a river segment of 113 km in length and flood plain of 2,800 km² in size was selected. The model was used to simulate a flood event in 2001 which was comparable with another major flood event occurring in 2000. The maximum flood extent in 2000 was monitored by the Mekong River Commission (MRC, 2011) and is shown in Fig. 2.2, where the flooded area was approximately 45,000 km².

Figure 2.2. The maximum flood extent in the historical 2000 flood event (MRC, 2011)

2.1.2 TELEMAC-2D model

The TELEMAC system which was introduced by the National Hydraulics and Environmental Laboratory - a part of the R&D group of Electricite de France, is a numerical modelling system developed to study environmental processes in free surface transient flows. The primary purpose of the TELEMAC-2D model is to simulate the dynamics of flow in a water-body, including rivers, estuaries or coastal areas via the solution of Shallow Water Equations (SWEs). TELEMAC-2D model solves the depth-integrated SWEs where the horizontal length scale is much greater than the vertical, using the finite element method on an irregular mesh that covers the floodplain. At each computational node of the mesh, TELEMAC estimates the depth of flow and two horizontal velocity components (Bates et al., 1992; Stewart et al., 1998; Hervouet, 2000). Effects of bottom friction, diffusivity and Coriolis and hydraulic structures such as weirs are considered in the model. The governing equations considered are (George, 2004):

$$\frac{\partial}{\partial t} \int_A U dA + \int_C H \cdot \hat{n} dC = \int_A (S_0 + S_f) dA \quad (2.1)$$

where A is the area of the integration element, C is the element boundary, $\hat{n}$ is the outward unit vector normal to C , U is the vector of the conserved variables and $H = (F, G)$ is the tensor of fluxes in the x and y directions, i.e.

$$U = \begin{bmatrix} h \\ uh \\ vh \end{bmatrix}, F = \begin{bmatrix} uh \\ u^2h + \frac{1}{2}gh^2 \\ huv \end{bmatrix}, G = \begin{bmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 \end{bmatrix} \quad (2.2)$$

where h is the flow depth, u and v are the velocity components in the x and y directions respectively, and g is the gravitational acceleration. S_0 and S_f are the bed and friction slope terms, respectively, as:

$$S_0 = \left[0; -gh \frac{\partial z}{\partial x}; -gh \frac{\partial z}{\partial y} \right] \quad (2.3)$$

$$S_f = \left[0; -c_D u \sqrt{u^2 + v^2}; -c_D v \sqrt{u^2 + v^2} \right] \quad (2.4)$$

which z is the bed elevation, $c_D = gn^2/h^{1/3}$ is based on the Manning formula with n being the bed roughness factor.

The TELEMAC-2D model can be used for both steady and non-steady simulations taking into account non-linear effects including propagation of long waves. TELEMAC-2D supports parallel processing which can be performed on a multi-core workstation to improve computational speed, using the technique of domain-decomposition where the computational grid or mesh is split into a series of smaller sub-domains. Each sub-domain is run by individual processors and the architecture therefore requires information exchange activity between computational nodes i.e. processors operate independently and merely pass information between one another, such as fluxes at the boundary of each sub-domain. Parallel virtual machine is used as MPI interface to account for the communication between computation cores.

2.2 Methodology

2.2.1 Model developments

The topography data used in this study consists of the Digital Elevation Model (DEM) with resolution of 200 m for the flood plain and 42 cross-sections for the river segment from Kratie to Kampong Cham. In the pre-processing stage, the cross-section data were interpolated and combined with the DEM of the flood plain to obtain an overall topographical grid for the area, with resolution of 200 m and each cell consisting of information of the x - and y -coordinates and elevation z . The model domain boundary was estimated based on the maximum flood extent of historical flood events. The entire modelling domain was discretized using a system of irregular triangular elements combining the mesh for the channel and flood plains. The mesh for the channel was created at intervals of 250 m and the mesh for the flood plain was created with a default edge length of 450 m. The elevation z was assigned to nodes on the mesh using the nearest neighborhood method. A 3D view of the mesh adopted for this study is shown in Fig. 2.3.

Figure 2.3. A 3D view of the mesh associated with the land surface elevation and bathymetry for the flood plain and river between Kratie and Kampong Cham

The simulation implemented on TELEMAC-2D was used to model the flood event from 4th June to 30th November 2001. The initial conditions for our simulations were established from the last time step of the results file for the simulation of the period of 1st June to 3rd June 2001. The boundary conditions used include the specification of water depth and velocity components or components of flow-rate at model boundaries. The measured discharge at the river entrance and water level at the river outlet i.e. the inflow (m^3/s) at Kratie and the water level (m) at Kampong Cham therefore were adopted. The computational time step was 60 seconds. A number of physical parameters were specified. These include the bottom friction, turbulence model and diffusivity. Strickler's friction coefficient was used to model the friction on the bed, which was assumed to be the same throughout the entire computation domain. The friction coefficient was specified to be $30 \text{ m}^{1/3}/\text{s}$.

The Blue Kenue software (Canadian Hydraulics Centre) was used to create the mesh from the DEM and river bathymetry data, define the boundary conditions and view the post-processing data. The model was developed and performed on a DellTM Precision T5600 workstation with Intel Xeon six-core CPU with 64 GB RAM. The computing time of the model was evaluated using 1, 2, 4 and 6 cores to compare the run-time performance.

2.2.2 Validation of the simulation results

2.2.2.1 MODIS satellite image processing

Satellite images of the case study area were obtained for seven days selected through the simulation period to depict pre-, peak- and post-flooding. The Moderate-Resolution Imaging Spectroradiometer (MODIS) images from National Aeronautics and Space Administration (NASA), which are available online, were used. The MODIS images used for our project are MOD09A1 – MODIS Terra/Aqua Surface Reflectance 8-Day L3 Global 500 m images. Each image is a composite of eight consecutive days of surface reflectance images at 500 m resolution. Cloud coverage is reduced through using these 8 days of images, with the ‘best’ observation for every cell in the image retained to produce the MODIS image. The ‘best’ observation is selected based on criteria such as high observation coverage, low view angle, absence of clouds or cloud shadow, and aerosol loading (MODIS Surface Reflectance User’s Guide, Version 1.3, 2011). Each MODIS satellite image contains seven spectral bands of data, with wavelengths ranging from 620nm to 2155nm.

The Normalized Difference Water Index (NDWI) is applicable for remote sensing of waterbodies from space. It has been used for flood detection (Gao, 1996; McFeeters, 1996). The NDWI is given by:

$$NDWI = \frac{Green - NIR}{Green + NIR} \quad (2.5)$$

where Green is the Green Band, Band 2, while NIR is a Near-Infra Red Band, which can be taken as Band 4 from the MODIS satellite spectral bands. Both of these Bands are able to sense similar depths through vegetation canopies. While liquid absorption by the Green Band is negligible, there is weak liquid absorption by the NIR Band, which is also enhanced by canopy scattering. Hence, the NDWI is able to clearly show changes in liquid water content of vegetation canopies (Gao, 1996). The converted Band 2 and Band 4 from MODIS images were used in the computation of NDWI to compare with our model results. When the value of NDWI is more than 0, it indicates a flooded area. Fig. 2.4 shows an example of the processed satellite image and its segregation between areas with NDWI values more than or less than 0. The

lighter areas have values less than 0, indicating low or no flooding. On the other hand, the darker areas have NDWI values more than 0, indicating inundation. The case study area of Kratie to Kampong Cham is located within the boxed area.

Figure 2.4. NDWI Determination in ArcGIS

2.2.2.2 ArcGIS Comparison

The TELEMAC simulation results were post-processed in Blue Kenue (Canadian Hydraulics Centre). Water Depth of the simulation results for each of the same seven days were saved as shape files that can be read by ArcGIS. Since eight results were printed out for each day of the simulation, we used the results extracted for Mid-day (12:00:00) for comparison.

Several threshold values were used to determine the percentage of agreement between the NDWI and simulated water depth. For the NDWI values, we checked if NDWI exceeded 0, -0.1, -0.2, and -0.3; whereas for the model water depth results, we identified cells that exceeded 0, 0.3 and 0.5 in value. By matching the cells that give both NDWI and water depth exceeding the threshold values, we can then conclude whether or not the model simulation and satellite images give good agreement on flood occurrence. Figs. 2.5 and 2.6 show an example of the model result and satellite data for the flood peak on 21 Aug 2001, respectively. Fig. 2.7 shows the superimposed layers when $NDWI > -0.3$ and water depth > 0.5 .

Figure 2.5. Model results (Water Depth>0.5)

Figure 2.6. Observations using satellite image data (NDWI > -0.3)

Figure 2.7. Superimposed Layers of NDWI > -0.3, Model > 0.5

2.3 Results and Discussions

2.3.1 Simulation results

The simulation results consisting of velocity and water depth for the wet season from 4th June to 30th November 2001 was processed with Blue Kenue (Canadian Hydraulics Centre). The results for 10th June 2001, 21st August, and 25th November 2001 corresponding to pre-, post- and peak-flood periods, are compared. Figs. 2.8(a), (b) and (c) display the inundation in terms of the flow velocity in the river and flood plain. The water level measured at Kratie was 16.0 m on 21st August 2001. The estimated discharge from stage-discharge relationship was 67,358 (m³/s). This is comparable to the maximum water level at Kratie for the flood event in 2000, which was 15.90 m.

(a) pre-flood (10th June)

(b) peak-flood (21st August)

(c) post-flood (25th November)

Figure 2.8. Flow velocity for (a) pre-flood (10th June), (b) peak-flood (21st August) and (c) post-flood (25th November) periods during the wet season in 2001

2.3.2 Model verification with MODIS satellite images

The model results and satellite information were separately analyzed and the flooded cells, identified by both the model and satellite image were compared. A match between model results and satellite image is determined in ArcGIS when a cell is identified as “flooded” by both the model and satellite image. The matched cells are shown in dark blue in Fig. 2.9 for areas that are inundated.

In order to provide a quantitative evaluation of the agreement between results from these two approaches, an index of percentage of agreement was used:

$$\text{Percentage of agreement} = \frac{N}{M} \times 100\% \quad (2.6)$$

where N is the number of matched cells with NDWI > -0.3 and water depth > 0.5 m, M is the number of cells with water depth > 0.5 m. The percentage of agreement was 32.8%, 93.5%, and 38.4% for the pre-flood (10th June 2001), peak-flood (21st August 2001), and (c) post-flood (25th November in 2001) period, respectively.

The threshold values were varied to determine the best conditions for optimal match between NDWI and simulated water depth results. For NDWI values obtained from MODIS data, we used threshold values of 0 to -0.3, with values exceeding -0.3 giving more areas with inundation, compared to NDWI > 0. Similarly, for modelled water depth results, water depth more than 0 showed greater inundated areas with water depths > 0.5 m. With the different combinations of flood threshold values therefore, we are able to determine the optimum combination for agreement between the pre-, peak-, and post-flood periods.

Table 2.1 shows that the highest percentage of matched results for the three periods considered occurred when values of NDWI > -0.3 and model water depth > 0.5 m. With the lowest threshold of -0.3 for MODIS data, we extracted the maximum wetted area possible. The threshold for model water depth was set at 0.5, to extract the extreme flooded areas. Hence, this combination would give the highest agreement between both the model and satellite image.

Table 2.1 Percentage of agreement (defined by different ranges of water depths)
between inundated areas in model and satellite image for various periods

Water depth > 0				
DAY	NDWI > 0	NDWI> -0.1	NDWI > -0.2	NDWI > -0.3
10 Jun	7.4	11.0	17.8	26.3
12 Jul	12.0	18.6	25.2	35.6
5 Aug	3.3	38.3	56.9	67.0
21 Aug	64.3	80.7	87.8	91.7
6 Sep	63.8	75.0	81.9	86.2
24 Oct	13.7	19.4	25.8	34.8
25 Nov	10.5	19.6	26.0	31.6
Water Depth > 0.3				
DAY	NDWI > 0	NDWI> -0.1	NDWI > -0.2	NDWI > -0.3
10 Jun	9.1	13.4	21.4	31.1
12 Jul	12.6	19.2	26.1	36.6
5 Aug	3.4	39.9	58.1	68.2
21 Aug	66.5	83.0	89.7	93.3
6 Sep	65.6	76.9	83.9	88.0
24 Oct	14.7	20.8	27.4	36.8
25 Nov	12.7	23.3	30.3	36.4
Water Depth > 0.5				
DAY	NDWI > 0	NDWI> -0.1	NDWI > -0.2	NDWI > -0.3
10 Jun	9.7	14.3	22.6	32.8
12 Jul	12.7	19.4	26.3	36.8
5 Aug	3.5	39.9	58.0	68.3
21 Aug	66.6	83.2	89.8	93.5
6 Sep	66.0	77.2	84.2	88.3
24 Oct	15.0	21.2	27.9	37.3
25 Nov	13.4	24.7	32.1	38.4

(a) 10 June 2001

(b) 21 August 2001

(c) 25 November 2001

Figure 2.9. Flood inundation identified by both satellite image and model for (a) 10 June 2001, (b) 21 August 2001, and (c) 25 November 2001

The optimal matching combination can also be observed in Figs. 2.10 to 2.12. Each of these figures shows the number of flooded cells as predicted by the model and the cells that are flooded in the model matching those observed from the satellite images. It can be seen from these figures that during pre- and post-flood period, the difference in the number of flooded cells found in both model and satellite and that of the model is significant. On the other hand, at the peak flood period (Aug 2001), this difference is much lower. The lowest deviation in number of flooded cells occurs when $NDWI > -0.3$. This indicates that using this threshold for the NDWI values would give the highest agreement with model results. With this threshold, we explore the water depth threshold, which would yield the highest percentage match.

It can be seen that the percentage match for the pre- and post-flood are $< 50\%$. However, for the peak period, agreement between NDWI and modeled water depth results are high and are about 90% . The low match during the pre- and post- flood period may be attributed to the low water levels in the river that could have been undetected by MODIS satellite images. As the water levels were too low in the river when there was still minimal flooding, the areas with water did not show up clearly on the satellite imagery and hence could not be extracted on ArcGIS. Another possible reason for the observed discrepancy is that the model considers only hydrodynamic processes of flows in the river and flood plain and it does not consider other hydrological processes such as infiltration or evaporation. Thus, at times before and after the flood, the model tends to provide larger flood inundation area than that the actual case since the abstractions represented by these processes are not considered.

On the other hand, during peak flood, especially on 21st August, the percentage comparison between the MODIS satellite image and modeled water depth was high. Once again, the errors could be attributed to losses not considered in the model or the shallow depths at the fringes of the inundated zone. The high percentage match between model simulation results and MODIS data indicates that TELEMAC-2D is indeed suitable for flood studies; however, issues related to the poor match between model and observations during the pre- and post- flood periods will have to be further investigated. Nevertheless, it can be concluded that for the period around the peak

flood, the 2D hydrodynamic model from TELEMAC-2D is able to produce results that are in close agreement with observations.

Figure 2.10. Number of flooded cells in model (water depth>0) and satellite images

Figure 2.11. Number of flooded cells in model (water depth>0.3) and satellite images

Figure 2.12. Number of flooded cells in model (water depth>0.5) and satellite images

2.3.3 Model computing time with multi-core processor

The simulation for the flood event from 4th June to 30th November 2001 was implemented on a multi-core workstation. The computing time was 177, 87, 51, 44 minutes corresponding to 1, 2, 4, and 6 cores used as shown in Fig. 2.13.

As expected, the computing time was lower when the number of cores used increased; however, the improvement in computational speed was less significant when the number of cores was larger than 4. The decrease in computing time was about 50.8 % when 2 cores were used. The reduction in runtime was 41.3% when 4 cores were used. Finally, it was only 13.7% when 6 cores were used. This trend reveals that the computing time approaches a limitation when the number of cores is more than 6. This limitation is presumably due to the MPI operations as reported in other studies (Khaitan & McCalley, 2014). HPC applications often require numerous information exchange activities between computational nodes. In the case of the MPI model, the nodes do not share the physical memory; therefore, they communicate by sending information back and forth many times per second, which likely raises the possibility of latency when the number of node increases beyond a certain limit.

Figure 2.13. Computing time of the simulation with 1, 2, 4 and 6 cores.

Chapter 3

Computational hydraulic modelling with UPC architecture for large scale simulations

The content of this chapter is extracted from a research article*. Flood modelling and forecasting based on the two-dimensional Shallow Water Equations is increasingly more expensive due to the uncertainties with climate changes as well as higher demands on accuracy in dense urban environment. At present, two parallel computing architectures, namely the Message Passing Interface (MPI) and OpenMP, are used extensively to accelerate the hydraulic simulations. In this study, we present an alternative approach using the Unified Parallel C (UPC) architecture that combines the advantages of MPI scalability as well as the direct memory access of OpenMP. A second-order Godunov-type MUSCL scheme flood model with second-order accuracy named ParaFlood2D is first developed using UPC. The computational efficiency of the model is then demonstrated with two cases of flood wave propagation using a shared-memory and a distributed-memory system. The results show high accuracy compared to the analytical solution, and the computational speedup is significantly enhanced with UPC compared to MPI and OpenMP at their base designs in both cases.

* The material presented in this chapter has been prepared to be submitted for the consideration of publication

3.1 Introduction

Flooding is one of the most destructive natural disasters in the world. According to Krzhizhanovskaya et al. (2013), global floods caused the death of nearly 7000 people and claimed an economic loss of 13.7 billion USD annually from 1984 to 2013. With climate changes that are likely to further exacerbate the intensity and duration of rainfall (as well as draught), the risk of flooding will likely escalate further in the future for many countries. It is therefore essential that flood assessment be routinely carried out to protect the communities along the flood plains.

Over the last few decades, computational hydraulic modelling has evolved to become a critical tool for the design, planning and management of water infrastructure for flood protection and assessment (Hunter et al., 2007). With modelling, scenario forecasting for flood predictions can be made to minimize the damages and potential loss of life (Hall et al., 2005). The sophistication of the modelling in terms of theories, data availability, computational speed and verification technology (including satellite imagery) are also continuously improving with time (Gomes Pereira & Wicherson, 1999; Hervouet, 2000; Rabus et al., 2003).

Studies so far indicate that two-dimensional (2D) depth-integrated models based on the Saint-Venant equations (or the so-called Shallow Water Equations (SWEs) are the most appropriate choice for the simulation of flooding in river basins (Panday & Huyakorn, 2004; Hervouet, 2007). Many 2D models, differing in complexity with respect to their numerical algorithms for the SWEs, are now available both publicly and commercially (Neal et al., 2010). The required computational resources for these models depend on the objective, but are typically expensive for multi-objective assessment including uncertainties with trending climate changes and land uses. For example, Morgan et al. (2015) examined the flood prediction for the city of Nice in France using the model of FullSWOF_2D, including the uncertainties in the bathymetry resolution. Over 400,000 CPU-hours were needed for their exercise. Thus, multiprocessor parallelization (via CPU-based architectures) is now widely used for flood inundation modelling to accelerate the simulation time.

A popular parallel computing architecture for flood modelling is based on the distributed-memory approach using MPI as the implementation protocol. Examples of flood modelling software using the MPI protocol include TELEMAC-2D (Hervouet, 2000), RMA (Rao, 2005), TRENT (Villanueva & Wright, 2006) and CalTWiMS (Pau & Sanders, 2006). With MPI, each computing CPU core carries out its calculations independently and commands its own memory in a modular manner, while the communication protocols manage the memory access among the cores. With the flexibility and ease of use, MPI is the predominant architecture for flood modelling parallelization at present.

In recent years, many MPI applications reported a drawback of reaching the limit of performance too early when the number of computational nodes is increased. For example Vu et al. (2015) investigated the use of parallelization with the public-domain software, TELEMAC-2D, for the flood prediction of a sub-basin along the Lower Mekong River in Southeast Asia. The computational domain covered a total watershed area of approximately 2,800 km², with 200m resolutions both along the river and in the flood plain. They reported that the computation was accelerated significantly when the number of cores increased from 1 to 4. Beyond that, however, the speedup slowed down significantly, and only marginal gain was realized when the number of computing cores increased to 6 and beyond. In other words, no significant further acceleration can be achieved even though additional computing resources were available. The limiting experience was also reported by other studies (Khaitan & McCalley, 2014).

The above limitation can be attributed to the bottleneck in the information exchanges among the computing nodes. With MPI, since the nodes do not share the physical memory, messages have to be sent back and forth many times per second. Thus, the latency increases when the number of nodes increases. This limitation is a major drawback, particularly for large watersheds whereby the objective is to provide rapid predictions of the extent of flood inundation ahead of the flood wave arrival to forewarn the communities in the flood plain. The latency can be reduced by customizing the algorithms in the main part of model, including the load-balancing, domain decomposition, communication, etc., to minimize the information exchanges

among the working nodes. For example, Sanders et al. (2010) designed a MPI-based Godunov-type shallow-water model called ParBrezo. They introduced a static method of domain decomposition using the Metis library of graph partitioning tools, to overcome the challenge of allocating computational resources for unstructured meshes. As a result, ParBrezo achieved outstanding speedups up to 512 cores. Another example was the CFD model developed by Frisch (2014) using MPI, with new communication schemes to minimize the synchronized message exchange activities among the computing nodes.

Besides MPI, another popular computer architecture for flood modelling is OpenMP which has been in existence for the last few decades. OpenMP is a shared memory approach. Thus, the memory can be accessed by all computing nodes in a public manner, which allows for the simple programming of parallel applications. Examples of flood models that utilize OpenMP include LISFLOOD-FP (Bates & De Roo, 2000) and the upgraded version of LISFLOOD developed by Neal et al. (2009). However, the performance of OpenMP is bounded to the computational power of a single system with a fixed number of nodes (Aubry et al., 2011). Available computational resources cannot be added easily contrary to the MPI protocol. To overcome this scalability constraint, newer architectures have been developed using the hybrid MPI-OpenMP approach. The implementation of this hybrid approach for parallelization is however more difficult, and the performance gains might not compensate for the effort (Rabenseifner et al., 2006; 2009).

Recently, methods using Graphic Processing Units (GPUs) have also been explored to reduce the computational time for flood inundation modelling (Kurowski et al., 2011; Liu et al., 2011; Kalyanapu et al., 2012; Vacondio et al., 2014). GPU-based flood models have been cited for achieving significant speedups and energy efficiencies (Kalyanapu et al., 2011). However, these studies also reported that the GPU-based models faced a number of challenges, including the need for lengthy code development time and the challenge to implement the logical statements involved (Neal et al., 2010). Therefore, CPU-based approach is still the favorite at present for flood modelling based on traditional numerical methods.

In this study, our objective is to apply the Partitioned Global Address Space architecture (PGAS) as an alternative to flood modelling parallelization to overcome the above constraints. PGAS is a relatively new architecture that combines the benefits of the locality in shared memory architecture of OpenMP and data layout control of MPI. The key concept is to store the data having the affinity with the corresponding computing thread in the private/local memory. With the direct affinity, the memory access by the thread is enabled at the highest speed possible. At the same time, the global shared memory can be accessed by all threads, but at a higher cost than local. This arrangement enables PGAS to achieve an optimal combination of programmability, portability and performance scalability. Using UPC, which is the PGAS extension to the ISO C language and the Berkeley UPC compiler (Berkeley UPC), we developed a new computational hydraulic model (called ParaFlood2D) based on the second-order Godunov-type MUSCL scheme with second-order accuracy. Subsequently, we tested ParaFlood2D using two cases of 1D dam break and 2D flood wave propagation with two hardware configurations of a shared memory SGI system and a distributed memory HPC server. We also compared the performance of ParaFlood2D with the MPI and OpenMP implementation at their base designs with the identical numerical scheme.

This chapter is structured as follows. Section 3.2 describes in detail the development of the computational hydraulic model and numerical solutions in this study. Section 3.3 presents the UPC architecture of the model. The computational results for the two cases with the two multi-core systems are shown and analyzed in Section 3.4.

3.2 Numerical method

The integral form of the shallow-water equations can be expressed as (George, 2004):

$$\frac{\partial}{\partial t} \int_A U dA + \int_C H \cdot \hat{n} dC = \int_A (S_0 + S_f) dA \quad (3.1)$$

where A is the area of the integration element, C is the element boundary, $\hat{n}$ is the outward unit vector normal to C , U is the vector of the conserved variables and $H = (F, G)$ is the tensor of fluxes in the x and y directions, i.e.

$$U = \begin{bmatrix} h \\ uh \\ vh \end{bmatrix}, F = \begin{bmatrix} uh \\ u^2h + \frac{1}{2}gh^2 \\ huv \end{bmatrix}, G = \begin{bmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 \end{bmatrix} \quad (3.2)$$

where h is the flow depth, u and v are the velocity components in the x and y directions respectively, and g is the gravitational acceleration. S_0 and S_f are the bed and friction slope terms, respectively, as:

$$S_0 = \left[0; -gh \frac{\partial z}{\partial x}; -gh \frac{\partial z}{\partial y} \right] \quad (3.3)$$

$$S_f = \left[0; -c_D u \sqrt{u^2 + v^2}; -c_D v \sqrt{u^2 + v^2} \right] \quad (3.4)$$

which z is the bed elevation, $c_D = gn^2/h^{1/3}$ is based on the Manning formula with n being the bed roughness factor.

In this study, we concentrate our effort on the parallelization of the main computational algorithm with the Shallow Water Equations using the explicit predictor-corrector scheme (e.g. by Bradford and Sanders (2002)) and with the typical boundary conditions in flood modelling simulations. The objective is to test out the core concept in the adaptation of UPC, and to compare with the existing approaches of MPI and OpenMP at their base designs. To avoid potential issues with the explicit numerical scheme, we have chosen general test cases using structured domains without significant discrete changes in the bathymetry, and also with very small time steps to achieve a stable Courant number. In the future, we plan to continue the development of the software with the implicit methods (Casulli & Zanolli, 2002; Hodges, 2004) to allow for stability using larger time steps.

The details of the numerical scheme can be found in Appendix A. The flux at each cell is first computed at the $(n+\frac{1}{2})$ time level with an intermediate predictor step, and then adjusted at the $(n+1)$ time level with a subsequent corrector step. In order to achieve the second-order spatial accuracy, the monotone upstream scheme for conservation (MUSCL) is adopted to linearly reconstruct the values of h , u and v . To evaluate the fluxes, we apply the Godunov-type upwind scheme proposed by Roe

(1981). In addition, there are multiple methods for flood modelling to limit the spatial derivatives to realistic values, including minmod (Roe, 1986), VanLeer (van Leer, 1974) or VanAlbada (van Albada et al., 1997). Here, the superbee limiter method by Roe (1985) is used due to its effectiveness (Jeffrey et al., 1999). Finally, for the wet/dry tracking, we adopt the method suggested by Bradford and Sanders (2002).

3.2 UPC implementation of SWEs

In this section, the UPC implementation for the numerical solutions described above for the shallow water equations is presented. As discussed previously, the important advantage of UPC over the other approaches is the ability to control the physical association between the shared data items and threads. The overall design progress of ParaFlood2D is as follow: (1) developing the general structure of the model based on the numerical scheme; (2) identifying the time-consuming functions in the model and analyzing the data dependencies; (3) establishing the algorithms for data storage based on data dependencies and model workflows; and (4) parallelizing the model using UPC work-sharing functions.

The general computational structure of ParaFlood2D is summarized in Figure 3.1. The flux predictor at the $(n+\frac{1}{2})$ time level is computed by the function called "*FluxPredictorX/Y*" and the flux corrector at the $(n+1)$ time level is calculated by the function "*VelocityAdjustor*". The functions *InitValue*, *FluxPredictor* and *VelocityAdjustor* have a nested loop (i.e. an inner loop within the body of an outer one). Thus, the complexity of the algorithm is $O(N^2)$ where N is the number of cells in the x-direction. The function *InitValue* is used to set up the initial condition, while *FluxPredictor* and *VelocityAdjustor* compute the flux for every cell in the domain making them the most time-consuming functions in the model. The focus of the UPC implementation, therefore, is to reduce the run time of these two functions.

Figure 3.1. Computational structure of ParaFlood2D

The domain decomposition algorithm is performed on T_0 and summarized as below. With the decomposition, the computational matrix is split into a series of smaller sub-domains by rows (Figure 3.2), and each sub-domain has the affinity to one thread. With T threads, a $N*N$ domain would be decomposed into T sub-domains called D_i . Each D_i contains the data of $\frac{N}{T}$ rows as $\frac{N*N}{T}$ cells. The data of sub-domain D_i are assigned to the thread T_i using the blocked-cyclic techniques (Figure 3.2). Thus, while calculating the fluxes in D_i , T_i mainly works with the data which it has the direct affinity. The exceptions for T_i to work with data outside of D_i are thus limited to the computation related to the first and last rows only.

This domain decomposition algorithm has the advantage that it can handle different numbers of cells in the x- and y-directions without any constraint related to N and T . At the same time, the disadvantage is that the boundary data are fragmented, with the north/south boundaries stored in D_0 and D_{T-1} and the west/east boundaries in the other sub-domains. Thus, while calculating the fluxes, the working threads in the system will have to constantly check whether or not the cell is along the boundary layer.

Figure 3.2. Sub-domains division

Algorithm 3.1 Domain decomposition algorithm

- 1: **if** current thread is the main thread (MYTHREAD = 0) **then**
 - 2: set $rows_{perthread} \leftarrow \frac{\text{number of rows}}{THREADS}$
 - 3: **if** THREADS is NOT divisible by number of rows **then**
 - 4: set $rows_{perthread} \leftarrow rows_{perthread} + 1$
 - 5: **end if**
 - 6: **for** each thread of the activated threads
 - 7: set $thread_{start}(i) \leftarrow i * rows_{perthread}$
 - 8: set $thread_{end}(i) \leftarrow (i + 1) * rows_{perthread}$
 - 9: **if** $thread_{end}(i) > \text{number of rows}$ **then**
 - 10: set $thread_{end}(i) \leftarrow \text{number of rows}$
 - 11: **end if**
 - 12: **end for**
 - 13: **end if**
-

It should be noted that the algorithm should work well not only with structured Cartesian grids, but also with unstructured grids by using a pre-arranged static workload-balancing algorithm to divide the workload evenly and assign the data to the respective UPC threads. Thus, the algorithm is highly optimal for UPC implementation with the minimal shared data among threads. It minimizes the delay in global memory access and speeds up the model computation significantly when a large number of CPU cores are involved.

The computations of *FluxPredictor* and *VelocityAdjustor* are distributed using a work-sharing function named *upc_forall*. The pseudo code for *upc_forall* is shown in Figure 3.3. In UPC, the total number of threads can be determined at run time using the UPC identifier, *THREADS*, and each thread can be identified by using another identifier, *MYTHREAD*. With *upc_forall*, all threads with *MYTHREAD* from 0 to *THREADS-1* run through an identical code (i.e. the nested loop). However, they calculate the fluxes at different start- and end-points defined by the indexing technique (shown in Algorithm 3.2), which are the first- and last-cell of each sub-domain, respectively. Each thread is designed to calculate the fluxes in different sub-domains. This new approach developed in the present study is termed as the single-program multiple-data method (SPMD).

Algorithm 3.2 Pseudo code for *upc_forall*

```

1: upc_forall (t = 0; t < THREADS; ++t; t)
2:   for each row from  $thread_{start}(t)$  to  $thread_{end}(t)$ 
3:     for each column of the 2D domain
4:       UpdateFlux(row, column)
5:     end for
6:   end for
7: end upc_forall

```

After every work-sharing function, a synchronization point is inserted using another function called *upc_barrier* (as shown in Figure 3.1) to synchronize all threads before going to the next function. It is essential to include the *upc_barrier* between the *FluxPredictor* and *VelocityAdjustor*. This is because the input data for the corrector phase are the results derived from the predictor phase, and the *VelocityAdjustor* function is dependent on the *FluxPredictor* in terms of data resources. The aforementioned process is analogous to a producer-consumer scenario, which the consuming thread needs to ensure that the correct data have been made available to the producing thread.

In the following, we shall examine the accuracy and performance of ParaFlood2D for two cases of flood wave propagation to demonstrate the viability of the PGAS concept.

3.4 Model verification and performance evaluation

The accuracy of ParaFlood2D is first evaluated for a 1D dam-break configuration. The domain consists of a horizontal 100m long channel, with a dam located at a distance of 40m from the upstream boundary ponding an initial water depth of 1m. Downstream of the dam, the channel bottom is assumed to be dry. The initial conditions are thus:

$$h(x, t = 0) = \begin{cases} h_0 = 1m, & x < 40m \\ 0, & x \geq 40m \end{cases} \quad (3.5)$$

$$u(x, t = 0) = 0 \quad (3.6)$$

For this dam break configuration, the analytical solution had been obtained by Ritter (1892) as:

$$h(x, t) = \frac{2\sqrt{gh_0} - \frac{x}{t}}{9g} \quad (3.7)$$

$$u(x, t) = \frac{2}{3}(\sqrt{gh_0} + \frac{x}{t}) \quad (3.8)$$

within the interval of:

$$-1 < \frac{x}{t\sqrt{gh_0}} < 2 \quad (3.9)$$

Outside of this interval, the quantities h and u are not affected by the wave propagation and are thus identical to the initial conditions.

We perform numerical simulations of this dam break configuration using ParaFlood2D with the cell length set as 0.125m and the time interval at $\Delta t = 0.01s$. The numerical predictions and analytical solution for $u(x, t)$ are compared in Fig. 3.3 at $t = 2.5s$ and $t = 6.3s$. Overall, it can be seen that the two agree well with each other, demonstrating the high accuracy of the numerical prediction. In particular, the flood wave front is captured very well with the fine cell intervals used in the simulations.

(a)

(b)

Figure 3.3. Comparison of numerical results and analytical solution at $t =$ (a) 2.5s and (b) 6.3s

To evaluate the parallel performance, a common speedup parameter is used to measure the computational efficiency of the model which is defined as the ratio of execution time between the uni- and multi-processor:

$$S(N) = \frac{T(N)}{T_1(N)} \quad (3.10)$$

where $T(N)$ is the run time of the parallel algorithm, and $T_1(N)$ is the run time of the model using a single processor. The simulations are performed in two computer

clusters (shown in Table 3.1) with different hardware configurations of a shared memory SGI system and a distributed memory HPC server.

Table 3.1. Computer clusters used for model testing.

Cluster	Node CPUs	CPU speeds (GHz)	Cores per node	Node RAM (TB)	Available cores	Communication switch
SGI UV-2000	Intel Xeon					
	E5-4657LV & E5-2670	2.4	16	2	up to 100 cores	InfiniBand Shared-memory
IBM System xiDataplex	Intel Xeon 5500 Series	2.93	16	2	up to 128 cores	InfiniBand Distributed-memory

We now compare the computational performance of ParaFlood2D with UPC against the MPI and OpenMP implementation at their basic designs of the same numerical schemes in the SGI UV-2000 server. The dam break configuration is modified to a 500-km long channel with a regular Cartesian grid of 4 million cells for a total duration of 100s. Table 3.2 and Figure 3.4 show the results of the run time and speedup comparison.

By deploying the number of cores of 2, 4 and 8, UPC and MPI achieve the speedup values of 1.6/1.8, 2.7/3.4 and 5.4/6.4; whereas OpenMP achieves 2.4, 4.6 and 8.3, respectively. Although OpenMP has higher speed-up values compared to UPC and MPI, the run time of all three are in fact similar. This is expected as both UPC and OpenMP can exploit the advantages of the data locality. Thus, they are able to read and write directly to the internal memory section without experiencing any delay. Furthermore, with the small number of cores, the amount of time for querying and retrieving with messages with MPI is trivial. Therefore, the three has almost identical performance within one node, which has 16 cores. However, UPC and MPI outperform OpenMP significantly beyond that up to the maximum of 100 cores. In fact, UPC remains effective in term of computational speedup with no sign of

performance decline after reaching the maximum. In other words, further acceleration can still be achieved if additional cores are available.

Table 3.2. Run time and relative speedup among UPC, MPI and OpenMP for 1D dam-break

Number of Cores	UPC		MPI		OpenMP	
	Run time (hours)	Speed-up	Run time (hours)	Speed-up	Run time (hours)	Speed-up
1	14.5	1.0	17.0	1.0	24.0	1.0
2	9.0	1.6	9.5	1.8	10.0	2.4
4	5.4	2.7	5.0	3.4	5.2	4.6
8	2.6	5.4	2.7	6.4	2.9	8.3
16	1.7	8.3	1.6	10.5	2.6	9.1
32	0.9	15.7	1.0	16.7	10.6	2.2
64	0.5	29.0	0.6	27.3	10.4	2.3
100	0.3	43.7	0.6	29.0	8.2	2.9

It is particularly striking to note that UPC is almost 25 times faster than OpenMP using 100 cores. The very poor performance of OpenMP after 16 cores is expected, and can be attributed to the performance penalty with over-access to the shared-memory section. Similar to other shared-memory systems, the SGI UV-2000 server provides a software solution that allows applications to access all available memory as a virtual shared-memory block. However, the memory in UV server is still physically located in different nodes which are connected to each other using the network cable. Therefore, when extending the parallel computation to multiple nodes, the access to the shared-memory section by OpenMP is subjected to the communication delay. Comparatively, UPC is designed so that the threads mostly access only the memory section which they have the affinity. Thus, the communication delay is minimized leading to the good performance.

With MPI, the speedup in computational speed is significant up to 64 cores but becomes ineffective beyond that. The MPI speedup is 16.6 and 27.2 with 32 and 64 cores, respectively; but it is only 28.0 with 100 cores. This trend again reveals that the limitation for MPI at its basic design. In the MPI architecture, when the number of processing cores increases, the amount of message passing in the system also expands exponentially. Eventually, the message processing time surpasses the actual computational time for each CPU, and limits any further gain in performance.

Figure 3.4. Speedup for the one-dimensional dam break simulations

We also compare the model prediction and analytical solution for a dam break case with a sloping channel as shown in Fig. 3.5. The analytical solution for the unsteady non-linear wave on initially dry bed was reported earlier by Dressler (1958). The simulation is executed with an initial water depth of 1m, a grid spacing of $x = 0.014\text{m}$ and a Courant number of 0.2. It can be seen from the figure that the comparison is satisfactory. Subsequently, we also introduce bottom friction to this case with two Manning's roughness coefficients: 0.035 for the floodplains - pasture, farmland and 0.075 for floodplains - heavy brush (Manning's Roughness Coefficients, 2016).

(a)

(b)

Figure 3.5. Comparison of numerical results and analytical solution at $t =$ (a) 0.1s and (b) 0.3s

Figure 3.6 shows the simulation results with the bottom friction. Clearly, the friction does not significantly affect the fluxes, which is expected physically since the initial water depth is high.

(a)

(b)

Figure 3.6. Numerical results with the Manning's coefficient of $n =$ (a) 0.035 and (b) 0.075

Table 3.3 shows the run time and speedup with the dam-break case on a slopping channel. With the wet/dry bed condition, a computational step is added into the computational structure to compute the new water depth after the *VelocityAdjustor* function. However, the calculations are not extensive, thus the additional time required is not significant. In addition, the required data for the wet/dry calculations

are evenly spread over the entire domain, which is perfectly suitable for UPC to parallelize. Thus, the model speedup remains the same.

Table 3.3 Run time and relative speedup of UPC for the inclined plane case using the SGI UV server

Number of Cores	1	2	4	8	16	32	64	100
Run time (hours)	16.3	10.4	6.7	3.1	2.0	1.4	0.6	0.4
Speedup	1.0	1.6	2.4	5.3	8.1	11.0	27.3	41.3

Next, we simulate another 2D flood wave propagation case in the two different machines of the above shared-memory SGI system and also another distributed-memory system for performance evaluation. The distributed-memory system consists of the IBM HPC server with 128 CPU cores shown in Table 3.1. The 2D domain consists of a horizontal area which is 1000 m long and 1000 m wide. We use a Cartesian grid with 100 million cells for the simulation to demonstrate the capability of ParaFlood2D in handling an ultra-large computational domain (the cell length is set to 0.1m). The model is run for a duration of 100s with the time interval $\Delta t = 0.01s$. Figure 3.8 shows the evolution of the fluxes in the x-direction after 200 time-steps (i.e. at $t = 2s$). With the shared-memory SGI machine, the run time and relative speedup of UPC, OpenMP and MPI are shown in Table 3.4 and Fig. 3.9.

The initial conditions are described below, with Figure 3.7 illustrating the water depth distribution inside the domain:

$$h(x, y)_{t=0} = \begin{cases} h_0 = 1m, & 400m < x < 600m \text{ and } y < 180m \\ 0, & x < 400m \text{ or } x > 600m \text{ or } y > 180m \end{cases} \quad (3.11)$$

$$u(x, y)_{t=0} = 0 \quad (3.12)$$

Figure 3.7. Initial water depth for the two-dimensional flood wave propagation case

Figure 3.8. Evolution of fluxes in the x-direction at $t = 2s$

The performance comparison again illustrates the two different trends before and after 16 cores, which is the number of CPUs in a single node of the SGI server. In particular, all three architectures achieve similar speedup level up to 16 cores. Beyond that, UPC outperforms significantly, while OpenMP demonstrates very poor scalability and MPI only maintains the speedup up to 64 cores at their basic designs. Again, UPC remains effective in term of computational acceleration up to the

maximum 100 CPU cores with no sign of performance decline. In terms of the run time, UPC significantly shortens the modelling processes from 14 days to 6 hours by increasing the number of cores from 1 to 100 for the simulation.

Overall, the relative performance of UPC and OpenMP for the 2D flood wave propagation case is similar to 1D dam break case in the share-memory SGI machine. This is because the 2D data are stored continuously in the memory as arrays of 1D data, which are kept in track by the compiler. Thus, the only difference between the two is that the compiler has to convert the cell's index to the corresponding 1D indicator in the memory section when querying the information from the cell. On the contrary, the performance of MPI is inferior for the 2D flood wave propagation case. The maximum performance for MPI is reached at 64 cores, and the computing efficiency actually reduces beyond that. The difference can be attributed to the size of the message in the system. For the 1D case, the message only contains the data of two cells i.e. the first and the last cell of the sub-domain. For the 2D case, however, each thread has to duplicate and send the data of the first and last rows, which consist of thousands of cells, to others. The processing time for each message is, therefore, much longer. This also explains the different number of limiting cores for MPI in different studies. For example, in the case of TELEMAC-2D reported by Vu et al. (2015), the limit was 6 cores; while for the FloodMap model developed by Yu (2010), the limit was 16 cores. It is important to note that with proper optimization, MPI can also reach a high level of speedup with a large number of cores as shown by Sanders et al. (2010) and Lai & Khan (2016). Nevertheless, the objective of the present study is to investigate the adaptation of UPC for large scale flood modelling as an alternative, thus we have kept OpenMP and MPI at their simplest designs.

Figure 3.9. Comparison of UPC, MPI and OpenMP on SGI server

Table 3.4. Run time and relative speedup of UPC, MPI and OpenMP for the 2D case with the SGI server

Number of Cores	UPC		MPI		OpenMP	
	Run time (hours)	Speed-up	Run time (hours)	Speed-up	Run time (hours)	Speed-up
1	334	1.0	397	1.0	551	1.0
2	179	1.9	225	1.8	280	2.0
4	88	3.8	130	3.1	158	3.5
8	44	7.8	87	4.6	88	6.2
16	23	14.8	43	9.2	61	9.1
32	14	26.1	32	12.5	245	2.2
64	8	40.6	25	15.4	239	2.3
100	6	55.9	29	13.6	189	2.9

Table 3.5 shows the performance comparison between UPC and MPI for the 2D flood wave propagation case with the distributed memory system of the HPC server with 128 cores, which is in essence a hybrid configuration (using both InfiniBand and shared memory communication - a.k.a. distributed memory machine). The comparison is based on the relative speedup ratio with 16 cores onward for large scale simulations. In addition, we also compare the performance to the maximum speedup rate Sp based on the Amdahl law (Amdahl, 1967) as:

$$Sp = \frac{1}{(1 - P) + P/N_c} \quad (3.13)$$

where P is the proportion of the program that can be executed in parallel and N_c is the number of CPU cores used. This ideal performance of Amdahl law is plotted in Figure 3.10 for comparison.

Table 3.5. Run time and relative speedup of UPC, MPI and Amdahl law for the 2D case with the IBM server

Number of Cores	UPC		MPI	
	Run time (hours)	Speed-up	Run time (hours)	Speed-up
16	24	1.0	29	1.0
32	14	1.7	22	1.3
64	8.7	2.7	17	1.7
128	4.2	5.4	21	1.4

The results show that UPC consistently outperforms MPI at its basic designs, particularly from 64 cores toward. This demonstrates that ParaFlood2D, as a communication-intensive code, benefits significantly from the UPC intranode shared memory communication. Comparing the UPC performance with the Amdahl law, the trend of UPC speedup is on-par though below the theoretical value. UPC is therefore

a viable computer achitecture to accelerate the simulation time for large scale simulations.

Figure 3.10. Comparison of UPC, MPI and Amdahl Law with the distributed-memory machine

Chapter 4

Computational hydrodynamics with UPC architecture

The content of this chapter is extracted from a research article*. Alternative computer architecture, known as PGAS-UPC, was coupled with high performance computing (HPC) for computational hydrodynamics (CHD) simulations in this study. PGAS-UPC harnesses the respective strength of two traditional parallelism architectures, namely MPI's scalability and OpenMP's direct memory access, to accelerate the computational run time of CHD simulations. A model, termed as UPC-CHD, was developed on the proposed approach together with the 2-step explicit scheme from Lax-Wendroff family of predictors-correctors. UPC-CHD was evaluated on three incompressible, viscous flow cases having moderate flow velocities under laminar conditions, namely (a) Blasius boundary layer, (b) Poiseuille's flow, and (c) Couette's flow. The accuracy of the implemented numerical scheme in UPC-CHD was first validated by comparing the derived numerical results with the respective analytical solutions for the different cases, which showed good overall agreement. The computational performance of UPC-CHD was then compared with that of MPI and OpenMP at their basic designs in a SGI UV-2000 server with 100 cores maximum. UPC-CHD performed most efficiently and achieved a near 1:1 speedup, which suggests the viability of the proposed approach for large-scale CHD simulations in the future.

* The material presented in this chapter has been prepared to be submitted for the consideration of publication

4.1 Introduction

Computational hydrodynamics (CHD) simulations have garnered increasing significance for assisting civil engineers to evaluate and optimize industrial-related applications. Examples included flow simulations within the tight spacers of membrane modules to mitigate both fouling tendency and flow short-circuiting during filtering processes (Bucs et al., 2014; Jajcevic et al., 2013; Sousa et al., 2014). Combinations of CHD with other computational tools such as discrete element method (DEM) in porous-media related applications had also been introduced (Sobieski and Zhang, 2017; Li. et al., 2011). In these CHD simulations, accuracy is required in the numerical scheme to discretize and resolve the Navier Stokes (NS) hydrodynamic equations. Concurrently, the computation should be accomplished within a reasonable time frame. For large scale CHD simulations, optimization between both factors is desired.

Typically, a large-scale CHD simulation run with high performance computing (HPC) requires significant management of the parallelization architecture in attempts to achieve the desired optimization. For example, a 100 million two-dimensional (2D) mesh involving 3 equations (continuity and momentum equations only) would result in an approximate 600 million-cell information ($100 \text{ million} * 2 * 3$) to be managed during each iterative step. Data sharing among computer cores (CPUs) is unavoidable in CHD mesh-bounded numerical domains. At present, the common parallelization architecture available for data sharing includes the distributed-memory approach with message passing interface (MPI) and the shared-memory approach using OpenMP.

Examples of CHD software using MPI include the very popular Open-source Open Field Operation and Manipulation (OpenFOAM) and ANSYS FLUENT, which involves communication of the shared data between computer cores by transmitting the messages back and forth with either point-to-point or collective communication protocol (Jamshed, 2015). Both types of communication can involve blocking or non-blocking methodologies. The blocking method puts the program execution on hold until the message buffer slots within the computer memory is ready, which might incur significant amount of idle time for significant number of cores. The non-

blocking method proceeds on with the program execution and does not wait for the completion of the communication buffer, which minimizes the total idle time. However, data loss might be incurred during the process. To harness the scalability advantage of MPI's, computational parallelism is always required in application-customized algorithms to reduce the communication latency (Frisch, 2014). Thus, designing MPI applications for considerable number of computer cores with escalated levels of memory hierarchy has been difficult so far (Gourdain et al., 2009; Jamshed, 2015).

Comparatively, OpenMP facilitates the programming ease by utilizing a shared memory architecture which can be accessed by all computing nodes publicly. OpenMP architecture is, however, confined to the computational power of a singular system of a fixed number of nodes (Aubry et al., 2011), and additional cores cannot be incorporated into its architecture readily which circumscribes its scalability (Jamshed, 2015). Thus, it is more popular for CHD software to employ MPI as compared to OpenMP. Hybrid parallelism which harnesses both MPI and OpenMP architectures has since been attempted for CHD simulations. Readers are referred to the recent references (Berger et al., 2005; Djomehri et al., 2002; Giovannini et al., 2015; Yilmaz et al., 2009) for further details. However, hybrid parallelism is more difficult to implement, and the performance gains might not justify the required programming efforts (Rabenseifner et al., 2006; 2009).

Acceleration of CHD simulations can also be accomplished by exploiting graphic processor units (GPUs) from the hardware perspective. For instance, simulations of laminar, turbulent and reactive flows had been attempted (Niemeyer and Sung, 2014), and the achieved speedups were faster than CPU-based approaches. An existing FORTRAN Euler code was altered by Brandvik and Pullan (2008) to work on GPU-based architecture, which achieved an improved speedup for two-dimensional (2D) and three-dimensional (3D) flow cases. However, challenges such as the difficulty of tailoring original codes to the GPU-based architecture and the need for lengthy code development time were mentioned. Readers are referred to recent GPU-based architecture works (Appleyard et al, 2011; Kuo et al., 2011; Salvatore et al., 2012; Schalkwijk et al., 2012) for further details.

Research efforts are still ongoing in attempts to develop an architecture which achieves an optimal balance between the accuracy of the numerical scheme and the desired speedup with increasing number of cores, while not compromising on the relative ease in its programming and implementation works. In this study, we developed an alternative parallelism approach for CHD simulations by adopting the Partitioned Global Address Space (PGAS) concept with Unified Parallel C (UPC) as the programming language. PGAS is a relatively new architecture that has been developed on the working principles of message passing and pure shared memory programming paradigms (Tarek et al., 2005). Two key advantages can be achieved with PGAS: (a) locality in the shared memory model which facilitates the ease of use, and (b) data layout control of MPI's which enables performance scalability. The PGAS architecture has been attempted in past literature for other applications, and readers are referred to the following references (Johnson, 2006 and Simmendinger et al., 2011) for further details. To the best of our knowledge, CHD simulations with the PGAS architecture, as performed in this study, have not yet been reported in the literature so far.

By coupling the PGAS-UPC architecture with the 2-step explicit numerical scheme from the Lax-Wendroff family of predictors and correctors, a UPC-CHD model was developed and evaluated on three incompressible, viscous flow cases having moderate flow velocities under laminar conditions, namely (a) Blasius boundary layer, (b) Poiseuille's flow, and (c) Couette's flow. Validation of the implemented numerical scheme was achieved by comparing the three cases with their respective analytical solutions for the given hydrodynamic conditions, which showed good overall agreement. Lastly, we shall show that UPC-CHD performed more efficiently than MPI and OpenMP at their base designs in a SGI UV-2000 server with a maximum of 100 cores in this study.

This chapter is structured as follows. In Section 4.2, we describe the numerical scheme implemented in the UPC-CHD model, which is followed by the description of UPC-CHD development with the adopted UPC architecture in Section 4.3. The parallelism performance of UPC-CHD is then examined in Section 4.4.

4.2 Numerical discretization

The governing equations, namely the continuity, momentum and energy equations, to describe the fluid dynamics within an engineering system (Anderson, 2009) can be expressed in the compact and conservative form of Equation 4.1.

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \frac{\partial G_{Vx}}{\partial x} + \frac{\partial G_{Vy}}{\partial y} \quad (4.1)$$

where Q is the conservative temporal term, F and G are the convective flux vectors in the x and y directions respectively, and G_{Vx} and G_{Vy} are the viscous flux vectors in the x and y directions respectively. By assuming fluid incompressibility, adiabatic flow conditions and no external forcing on fluid, the exact representation of the Q , F , G , G_{Vx} and G_{Vy} are shown in Equation 4.2.

$$\begin{aligned} Q &= \begin{bmatrix} 0 \\ u \\ v \\ E_t \end{bmatrix}, & F &= \begin{bmatrix} u \\ \frac{p}{\rho} + u^2 \\ uv \\ (E_t u) + \frac{p}{\rho} u \end{bmatrix}, & G &= \begin{bmatrix} v \\ uv \\ \frac{p}{\rho} + v^2 \\ (E_t v) + \frac{p}{\rho} v \end{bmatrix} \\ G_{Vx} &= \nu \begin{bmatrix} 0 \\ u_x \\ v_x \\ 2uu_x + vu_y + vv_x \end{bmatrix}, & G_{Vy} &= \mu \begin{bmatrix} 0 \\ u_y \\ v_y \\ 2vv_y + uu_y + uv_x \end{bmatrix} \end{aligned} \quad (4.2)$$

where u is the horizontal velocity [LT^{-1}], v the vertical velocity [LT^{-1}], $\frac{p}{\rho}$ the pressure divided by the fluid density [L^2T^{-2}], E_t the total energy per unit mass [L^2T^{-2}], ν the kinematic viscosity [L^2T^{-1}], u_x the x -derivative of u [T^{-1}], u_y the y -derivative of u [T^{-1}], v_x the x -derivative of v [T^{-1}], and v_y the y -derivative of v [T^{-1}].

Figure 4.1: Representative control volume of a single node for discretizing equation

Figure 4.2a: Numerical domain for Blasius boundary layer flow (Case A)

Figure 4.2b: Numerical domain for Poiseuille's flow (Case B)

Figure 4.2c: Numerical domain for Couette's flow (Case C)

Equation 4.1 is first discretized over the representative control volume of a single node in Figure 4.1. All other nodes within the numerical domain undergo the same discretization format. The temporal term of the discretized form is evaluated using the two-step explicit scheme from the Lax-Wendroff family of predictors-correctors. The predictor step computes the flow condition at the half-time step, whereas the corrector step follows up with the predictor step by employing a central differencing

with time at the full-time step. A comprehensive description of the numerical scheme implemented in UPC-CHD is described below.

4.3 Development of UPC-CHD

The following parallelization procedures are adopted within UPC-CHD: (i) the time-consuming functions and different forms of data dependences are first identified, (ii) the appropriate algorithms are then adopted for data divisions and storage as based on the data dependences and model workflow, and (iii) lastly the unique work-sharing function of PGAS-UPC is introduced to parallelize the workflow internally. The details of the implemented computational structure and the respective algorithm for domain decomposition, data storage and work-sharing function involved in developing UPC-CHD will be described in the following.

4.3.1 Computational structure

The adopted computational structure in UPC-CHD is summarized in the pseudo-form of Algorithm 4.3.1. The proposed structure minimizes the runtime required to compute the predictor and corrector fluxes within each node in Figure 4.1, which constitutes the largest portion of the total computational runtime. Referencing to Algorithm 3.1 in Appendix B, the flux predictor at the $n + 1/2$ time level is first computed which is then followed by the flux correction at the $n + 1$ time level. Both the flux predictor and flux corrector are within the nested loop (step 4 and 5 of Algorithm 3.1), and the complexity of the nested loop algorithm is defined as $O(N^2)$, where N is the number of computational cells in a singular direction. The original nested loop is divided into multi sub-loops to prevent data conflicts. After every new nested loop, a synchronization point is inserted using an UPC function, termed as *upc_barrier*, to synchronize all threads before proceeding to the next function. The sub-loops architecture is summarized in Figure 4.3.

Algorithm 4.3.1 Computational structure of UPC-CFD model

```

1:  set  $t \leftarrow 0$ 
2:  assign initial values to  $\vec{u}, p, T, \rho, \dots$ 
3:  while  $t < t_{end}$  do
4:    for each row of the 2D domain
5:      for each column of the 2D domain
6:        compute the convective fluxes at the half-time step with linear approximation
          of ROE scheme
7:        compute the ROE's averaged condition, eigenvalues of the Jacobian matrix,
          wave amplitudes and the eigenvector of the Jacobian matrix for each node
8:        compute the inner and outer values of the convective fluxes for each node
9:        compute the viscous term using 2nd order central differencing scheme
10:       repeat the process and update the velocity for full-time step
11:     end for
12:   end for
13:   set  $t \leftarrow t + \Delta t$ 
14: end while
  
```

Figure 4.3: Parallel computational structure for sub-loops of UPC-CHD

4.3.2 Domain decomposition algorithm

The required fluxes within each of the nodes in Figure 4.1 are computed via a row-by-row approach utilizing the respective data values of the two upper and two lower rows. To do so, the two-dimensional (2D) Cartesian computational matrix of UPC-CHD is divided into a series of smaller sub-domains having a defined number of rows, for which each sub-domain has affinity to a single thread. With T number of threads, a $N \times N$ domain is decomposed into T sub-domains ($D_i, i = 0$ to $T - 1$). The adopted protocol for domain decomposition is as follows: (i) if N is divisible by T , then each D_i contains the data of $\frac{N}{T}$ rows which results in $\frac{N \times N}{T}$ cells, or (ii) if N is not divisible by T , then the first $(T-1)$ sub-domains D_i ($i = 0$ to $T - 2$) contain $(\text{int}(\frac{N}{T}) + 1)$ rows and the last sub-domain D_{T-1} contains the remaining rows. Considering an 8×8 domain example in Figure 4.4 where N is 8, the mathematical representation described in pointers (i) and (ii) can be illustrated by the use of 4 threads ($T = 4$) and 3 threads ($T = 3$) respectively. The former comprises of 4 sub-domains for which each contains 2 rows whereas for the latter, the first 2 sub-domains, as measured from the top boundary, comprise of 3 rows each and the last sub-domain contains 2 rows. For a rectangular domain of $L \times B$, the number of sub-domains will be solely dependent on the division of L by the number of threads (T). The respective threads assigned to the first and last row are termed as $thread_{start}$ and $thread_{end}$, and the domain decomposition is first performed on thread 0.

Figure 4.4: Domain decomposition algorithm for a 8×8 numerical domain example with (left) 4 threads and (right) 3 threads

4.3.3 Data storage algorithm

The stored data of sub-domain D_i are assigned to the thread T_i using the blocked-cyclic technique which enables T_i to access the data having the direct affinity. The only exceptions for T_i to access outside of D_i are restricted to the first and last rows, which result in shorter computational time as compared to that of $(\frac{N}{T} - 2)$ rows for the Case of N being divisible by T , and $(\text{int}(\frac{N}{T}) - 1)$ rows for the Case of N not being divisible by T . The proposed blocked-cyclic technique is shown in Figure 4.5 and further discussed in Section 4.4.

Figure 4.5: Sub-domains division and data storage algorithm for a $n \times n$ numerical domain example

4.3.4 Work-sharing function

The computations within the nested loops are distributed using a work-sharing function, termed as *upc_forall*, in the pseudo-form of Algorithm 4.3.2. In UPC, the total number of threads is determined with a UPC identifier, *THREADS*. Each thread is identified by using another identifier, *MYTHREAD*. With *upc_forall*, all threads with *MYTHREAD* from 0 to *THREADS-1* undergo the identical code run (i.e. the nested loop). Each thread is designed to compute the fluxes on the different sub-domains which are identified by using different $\text{thread}_{\text{start}}$ and $\text{thread}_{\text{end}}$. The adopted approach is termed as the single-program multiple-data method (SPMD).

Algorithm 4.3.2

```
1:  upc_forall (t = 0; t < THREADS; ++t; t)
2:      for each row from  $thread_{start}(t)$  to  $thread_{end}(t)$ 
3:          for each column of the 2D domain
4:              compute the ROE's averaged condition
5:              compute eigenvalues of the Jacobian matrix
6:              compute wave amplitudes
7:              compute the eigenvector of the Jacobian matrix
8:          end for
9:      end for
10: end upc_forall
```

4.4 Model verification and performance evaluation of UPC-CHD

To validate the implemented numerical scheme in UPC-CHD, the numerical predictions derived for Cases A to C were compared with the respective analytical solution: (Case A) with the analytical solution of White's (White, 1991), (Case B) with Equation 4.3 in the following (Munson et al., 2006), and (Case C) with Equation 4.4 in the following (Munson et al., 2006).

$$\frac{u}{U} = \frac{(y^2 - h^2)}{2Uv} \left(\frac{\partial \left(\frac{p}{\rho} \right)}{\partial x} \right) \quad (4.3)$$

$$\frac{u}{U} = \frac{y}{h} - \frac{h^2}{2Uv} \left(\frac{\partial \left(\frac{p}{\rho} \right)}{\partial x} \right) \left(1 - \frac{y}{h} \right) \left(\frac{y}{h} \right) \quad (4.4)$$

where U is the freestream velocity [LT^{-1}], h is the total vertical height of the domain [L]. The physical dimensions of the deployed numerical domains and the initial flow conditions for Cases A to C are summarized in Table 4.1, which includes the respective Reynolds number (Re) based on the characteristic length of the domain. It is worth noting that the resulting Re for Case A is close to that of the Stanford's SU2 Open-source CFD code (Kline, 2017), despite the differences in the adopted

hydrodynamic flow conditions, i.e. flow velocity and temperature of domain. The implemented Roe scheme for discretizing the convective flux terms in Equation 4.1 has also been tailored to handle high speed flows in the recent references (Kermani and Plett, 2001).

It can be observed from Figures 4.6a to 4.6c that there is a very good agreement between the numerical predictions and respective analytical solutions for the test cases. However, minor differences can still be observed between the two at locations with the largest changes in the velocity gradient, which suggest that the grid sizes still need to be refined at these locations to achieve a higher resolution. An extension of the total simulation run time would also be useful to improve on the accuracy of the numerical values attained.

Table 4.1: Physical dimensions and initial conditions of deployed numerical domains for validating implemented numerical cases in UPC-CHD
(Cases A to C)

Parameter	Case A	Case B	Case C
x (L)	0.3	0.5	0.5
h (L)	0.02	0.00016	0.00016
N x N	65 × 65	300 × 45	300 × 45
U (m/s)	0.05	10	10
Re	15000	1600	1600
$\frac{\partial \left(\frac{p}{\rho}\right)}{\partial x}$ (ms ⁻²)	0	4700	4700
Δt (s)	10 ⁻⁶	10 ⁻⁶	10 ⁻⁶
Total runtime (s)	0.01	0.01	0.01
Temperature (K)	293.15	293.15	293.15
ν (m ² s ⁻¹)	10 ⁻⁶	10 ⁻⁶	10 ⁻⁶

Figure 4.6a: (Case A) Comparison between numerical predictions and analytical solutions for Blasius boundary layer flow at location $x = 0.2\text{m}$ (n' is the eta value)

Figure 4.6b: Comparison between numerical predictions and analytical solutions for Poiseuille's flow at location $x = 0.5\text{m}$ (Case B)

Figure 4.6c: Comparison between numerical predictions and analytical solutions for Couette's flow at location $x = 0.5m$ (Case C)

By using the speedup parameter in Equation 3.10, the parallelism performance of UPC was compared with that of OpenMP and MPI at their basic designs in a SGI UV-2000 server for Cases A and B of ultra-large numerical domains. The configuration of the SG UV-2000 server is summarized in Table 4.2. The physical dimensions of the deployed numerical domains with its initial conditions for performance evaluation are summarized in Table 4.3.

Table 4.2: SGI UV-2000 cluster used for model testing

Cluster	Node CPUs	CPU speeds (GHz)	Cores per node	Node RAM (TB)	Available cores	Communication switch
SGI UV-2000	Intel Xeon E5-4657LV & E5-2670	2.4	8/16	2	up to 100 cores	InfiniBand Shared-memory

Table 4.3: Physical dimensions and initial conditions of deployed numerical

domains for evaluating computational parallelism efficiency of UPC, OpenMP and MPI in UPC-CHD (Cases A to C)

Parameter	Case A	Case B	Case C
x (L)	0.3	0.5	0.5
h (L)	0.02	0.00016	0.00016
N x N	5000 × 5000	10000 × 10000	10000 × 10000

Comparison of the speedup performance for Cases A and B is summarized in Tables 4.4 and 4.5 respectively. Generally, UPC, OpenMP and MPI achieved efficient speedup up to 16 computer cores, which is the number of cores allocated to singular node in SGI server. For a relatively small number of cores deployed, the comparable speedup performance can be ascribed to the following reasons: (a) both UPC and OpenMP exploit the advantages of data locality for reading and writing directly to the local memory Section without incurring any delays, and (b) the amount of time for MPI to query and retrieve the data with the required messages is trivial for the relatively small number of cores. However, beyond 16 and up to 100 cores maximum in this study, UPC and MPI outperformed OpenMP significantly as shown Figures 4.7 and 4.8. UPC's speedup was most significant by having a close ratio of 1:1, and reached nearly 90 times speedup for Cases A and B up to the 100 cores maximum. The observed trend in UPC's speedup demonstrates the potential for further speedup if additional cores are available for allocation.

Table 4.4: Comparison of run-time and speedup among UPC, MPI and OpenMP for Blasius boundary layer flow (Case A) at varying number of computer cores

Number of Cores	UPC		MPI		OpenMP	
	Run time (hours)	Speed-up	Run time (hours)	Speed-up	Run time (hours)	Speed-up
1	132	1.0	133	1.0	135	1.0
2	66	2.0	66	2.0	71	1.9
4	33	4.0	33	4.0	42	3.2
8	17	7.9	17	7.9	21	6.3
16	8	15.7	8	15.8	19	6.9
32	4	28.6	5	26.7	56	2.4
64	2	56.9	3	48.1	48	2.8
100	1.4	88.9	2	60.9	58	2.3

Table 4.5: Comparison of run-time and speedup among UPC, MPI and OpenMP for Poiseuille’s flow (Case B) at varying number of computer cores

Number of Cores	UPC		MPI		OpenMP	
	Run time (hours)	Speed-up	Run time (hours)	Speed-up	Run time (hours)	Speed-up
1	53	1.0	53	1.0	54	1.0
2	26	2.0	27	2.0	28	1.9
4	13	4.0	14	4.0	15	3.5
8	6.8	7.8	7	7.9	8.5	6.3
16	3.5	15.8	4	13.3	7.4	7.2
32	1.8	30.1	2.4	22.5	19	2.8
64	1.1	56.6	1.2	46.7	14	3.8
100	0.5	89.2	1	50.2	20	2.6

On the contrary, OpenMP achieved a less-ideal speedup from 16 cores beyond as demonstrated in Figures 4.7 and 4.8, which could be attested to its over-accessing to the shared-memory component. The SGI server enables applications to access all

available memory in a unified manner via a virtual shared-memory block. However, the memory sections are still physically located in the different nodes which are connected to one another via network cables. Unavoidably, the access to the shared-memory section in OpenMP architecture is subjected to communication delay during the parallelism with multiple nodes. The less than ideal performance of OpenMP indicates its general unsuitability for CHD simulations of ultra-large numerical domains.

Despite having a slower speedup than that of UPC, MPI achieved relatively significant speedup for Cases A and B for the number of cores beyond 16. The distinction of the speedup between Cases A and B is likely ascribed to the obvious difference in the number of computational grids deployed, i.e. 25 million grids for Case A and 100 million grids for Case B, which affected the size of each message to be transmitted within the system. For instance, when running with 32 cores, MPI achieved a speedup of 26.7 and 22.5 for Case A and B respectively. With the MPI architecture, thread T_i transmits multiple messages to the neighboring threads at every time-step which include: (a) velocity data values in x- and y- direction corresponding to thread T_{i-1} and T_{i+1} , (b) convective fluxes data values in x- and y- direction, and (c) updated data to the main thread. At the maximum 100 cores in this study, over 900 messages were processed in the system at each iterative step, despite having only 100 rows of data to be computed for each core. The respective difference of the message size to be transmitted in Case A and B resulted in different processing time for each message. Thus, it was possible that the total message processing time outweighed the actual computational time on each core in this study, which restricted the continual speedup with an increasing number of cores.

Figure 4.7: Comparison of speedup among OpenMP, MPI and UPC for Blasius boundary layer flow (Case A) at varying number of computer cores

Figure 4.8: Comparison of speedup among OpenMP, MPI and UPC for Poiseuille's flow (Case B) at varying number of computer cores

UPC-CHD has been designed to avoid the performance penalty due to over-accessing the global memory, which enabled the model to maintain a stable performance on varying sizes of the numerical domain as compared to that of MPI. The advantage of embedded locality consciousness of UPC was further investigated in Case C by examining the impact of affinity on its performance. The computational data of the domain was first stored in-block to gain the memory locality properties, while the global memory accessing activities were overlapped with remote control technique using the split-phase barrier to conceal the synchronization cost. We then evaluated the performance of UPC for Case C under two scenarios: (a) UPC-A, i.e. UPC with optimizations, and (b) UPC-NA, i.e. UPC without optimizations and employs the defaults setting of the GPAS compilers.

UPC-NA's performance was vividly inferior to that of UPC-A's as shown in Table 4.6 and Figure 4.9.

Table 4.6: Comparison of run-time and speedup between UPC-A and UPC-NA for Couette's flow (Case C) at varying number of computer cores

Number of Cores	UPC-A		UPC-NA	
	Run time (hours)	Speed-up	Run time (hours)	Speed-up
1	53	1.0	53	1.0
2	26	2.0	64	0.8
4	13	4.0	56	0.9
8	6.8	7.8	26	2.0
16	3.5	15.8	12	4.3
32	1.8	30.1	7	7.3
64	1.1	56.6	4.6	11.4
100	0.5	89.2	3.9	13.0

Figure 4.9: Comparison of speedup between optimized UPC and non-optimized UPC for Couette's flow (Case C) at varying number of computer cores

For instance, at 16 cores, the speedup attained was 15.8 and 4.3 for UPC-A and UPC-NA, respectively. A detailed comparison of the speedup between UPC-A and UPC-NA at the varying number of cores for Case C is summarized in Table 4.6. The distinction of the attained speedup between UPC-A and UPC-NA can be further inferred from Figure 4.10, which involves a 3 x 3 numerical domain example having an affinity block of 3. In UPC-NA of Figure 10, thread 0 contains the fluxes data of element a, d and g in its local memory section whereas in UPC-A, thread 0 contains the fluxes data of element a, b and c in its local memory section. The observed performance inferiority of UPC-NA can be attested to the need for thread 0 to function with non-affinity data, i.e. element b in thread 1 and element c in thread 2, which resulted in longer computational run time for UPC-NA. With 1 core, there was only a singular thread which computed the fluxes in the entire numerical domain and only worked with data having affinity with. With increasing number of cores, the latency issue arose which resulted in less than ideal computational performance. For instance, at 2 and 4 cores respectively, 50% and 25% of the total runtime was attributed to accessing of the non-affinity data by the respective threads. While the

addition of cores would reduce the amount of global accessing activity in UPC-NA, optimization is still unlikely to be achieved as observed in Figure 4.9. Thus, it is recommended to distribute the array of data in contiguous blocks as in UPC-A, which each of the threads attends to a system of elements dependent on the number of available threads in the system. In other words, the implementation of UPC architecture without any affinity optimization is likely to be not recommended notwithstanding its potential advantage in parallelism performance as compared to OpenMP and MPI.

Figure 4.10: Schematic representation of UPC-NA (left) and UPC-A (right) concepts; T_i refers to thread i , D_i refers to domain i .

Chapter 5

UPC architecture for Lattice Boltzmann model

The content of this chapter is extracted from a research article*. In the present work, a parallel Lattice Boltzmann model, namely UPC-LBM, was developed to investigate the viability of the Partitioned Global Address Space architecture (PGAS) in simulating high-resolution fluid flows. The PGAS architecture harnesses the respective strength of two traditional parallelism architectures, namely the scalability of distributed-memory approach and the direct memory access of shared-memory method, to accelerate the computational run time of the LBM simulations. UPC-LBM model was developed using UPC language, a PGAS extension of standard C, using the 2-dimensional 9-velocity flows LBM setting. The validity of UPC-LBM was confirmed by comparing the numerical solutions to the exact solution of the plane Poiseuille's flow. The computational performance of UPC-LBM was then evaluated using a SGI UV-2000 server. A maximum speed-up of 97 times was obtained for 100 cores maximum on the multi-core CPU SGI platform.

* The material presented in this chapter has been prepared to be submitted for the consideration of publication.

5.1 Introduction

The continuous growth of computer power has motivated the scientific community to use CFD for the numerical solution of the governing equation of fluid dynamics (Chung, 2002). Over the years, there are two numerical methods that are frequently being used in CFD, finite difference method (FDM) and finite volume method (FVM) (Perumal, 2010). However, in the last two decades, a different kind of numerical method for applications in CFD, namely the Lattice Boltzmann method (LBM), has gained popularity (Yeomans, 2006). LBM has emerged as a promising tool for modelling the Navier-Stokes equations and simulating complex fluid flows (Chen & Doolen, 1998). It has been successfully applied to many challenging problems in hydrodynamics as well as reaction-diffusion processes and wave propagation phenomenon (Grunau et al., 1993; Han-Taw & Jae-Yuh, 1993; Martinez et al., 1994; Ho et al., 2002; Gupta et al., 2006).

In the LBM approach, one solves the kinetic equation for the particle distribution function. The macroscopic variables such as velocity and pressure are obtained by evaluating the hydrodynamic moments of the particle distribution function (He & Luo, 1997a). One of the most popular LBM approach is the Lattice Boltzmann equation with linearized collision operator based on the Bhatnagar–Gross–Krook (BGK) collision model, in which, one can recover the governing continuity and momentum equations in the low Mach number limit (He & Luo, 1997b). Another beauty of the LBM is that the collision and streaming processes are local, hence it can be programmed naturally for parallel processing (Girimaji, 2012). In recent years, a number of studies has been conducted to develop LBM models for simulating large and complicated CFD systems, with several hundred thousand to more than billion particles in the domain.

Today, a wide range of parallel LBM implementations is available ranging from specialized academic packages such as Ludwig for the simulation of complex fluids (Desplat et al., 2001) and HemeLB for the simulation of flow in blood vessels (Mazzeo & Coveney, 2008). Very versatile open-source projects such as OpenLB (OpenLB) and Palabos (Palabos) and commercial applications included PowerFlow

(Next Limit Technologies) and XFlow (Exa Corporation) were also established. The widely used parallel algorithms for LBM in those models are OpenMP (i.e. shared-memory architecture), MPI paradigms (i.e. distributed-memory approach) and GPUs architecture.

The shared-memory architecture using OpenMP library is generally preferred for LBM problems with small to medium domains having a limited number of lattice particles, which are normally executed on a personal computer. The OpenMP architecture allows the public memory access to all computing nodes, leading to programming ease with an accepted level of speedup. Examples of LBM software using OpenMP include the OpenMP-LBM model applied to multiphase mass transfer in porous media developed by Cekmer et al. (2016), which achieved the speedup of 3.4 with 4 processors; and the OpenMP version of the Hydrodynamic LBM code by Bella et al. (2002) which obtained the maximum a speedup of 5.9 with 6 processors. OpenMP architecture is, however, confined to the computational power of a singular system with a fixed number of nodes (Aubry et al., 2011). Thus, it is more popular for LBM-CFD software to employ MPI as compared to OpenMP for the high-resolution simulations, which requires running in a massive computer system.

MPI approach provides a parallel solution that allows the model to scale up to a larger number of processors. With MPI, each processor carries on their own data and communicates with each other by sending messages back and forth (Jamshed, 2015). With proper design and customization, the MPI-LBM model can achieve an outstanding performance while running with a large number of processors. For example, Puurtinen et al. (2017) developed an LBM solver with suspended particles using the MPI inter-communication framework and the model maintained the high efficiency up to 1000 cores. The efficiency of the model, however, decreased after 1000 cores and when reaching 2000 cores, no further acceleration can be gained even though additional computing recourses were available. This behavior of MPI model can be attributed to the bottleneck in the information exchanges among the computing processors, as the latency occurs and increases when the number of messages used by the processors increases. Therefore, algorithms including data distribution, messaging structure and mechanism, as well as the workload balancing are always

needed to take into account while developing the MPI-LBM model. Thus, designing MPI applications for a considerable number of computer cores with escalated levels of memory hierarchy has been difficult so far.

Acceleration of LBM-CFD simulations can also be accomplished by exploiting the GPUs architecture. In 2003, Li et al. (2003) attained first promising results of an LBM based flow solver on GPUs using the OpenGL graphics API. More recently, Tolke and Krafczyk (2008) successfully implemented an LBM in three spatial dimensions on NVIDIA GPUs using the CUDA toolkit. The studies reported that the simple kernel structure of LBM qualifies the GPUs architecture and the acceleration gained is very promising. However, challenges such as the shortage of GPU's local memory in realistic CFD application as well as the latency in data exchange between host and GPU using MPI communication were also mentioned (Habich et al., 2011).

In recent years Partitioned Global Address Space (PGAS) architecture has emerged as a viable alternative for cluster programming. The PGAS programming languages such as Unified Parallel C (UPC - a PGAS extension of standard C) or Coarray Fortran try to exploit the principle of locality on a computer node. The syntax to access data is similar to the OpenMP approach, which is usually easier for the programmer compared to MPI (Yelick et al., 2007). However, the advantages of UPC over MPI is that it offers additional locality control, which is important for distributed memory systems. Thus, a PGAS model is able to naturally operate on both share- and distributed-memory cluster. In addition, in the PGAS architecture, the processor can access data using one-sided communication primitives, even if the actual data resides on a different node. Therefore, if designed well, PGAS approach can be as effective as MPI and as easy to develop as OpenMP. However, the literature review has shown that the naive implementation does not result in optimal performance, and application-customized algorithms are always needed for PGAS model (Prugger et al., 2016).

In this study, we developed a new LBM model for the large scale CFD simulation using the PGAS architecture with UPC as the programming language. The new model, namely UPC-LBM, is built using D2Q9 LBM BKG method and validated on the Poiseuille's flow which is incompressible, viscous flow under laminar conditions.

Lastly, the performance of UPC-LBM is evaluated using a SGI UV-2000 server with a maximum of 100 cores in this study.

This chapter is structured as follow. In section 5.2, the LBM numerical scheme implemented in UPC-LBM is described, which is followed by the description of UPC-LBM development with the adopted UPC architecture in Section 5.3. The parallelism performance of UPC-LBM is then examined in Section 5.4.

5.2 Lattice Boltzmann Method

The Boltzmann equation is a partial differential equation (PDE) describing the evolution of the single particle distribution function $\mathbf{f}$ in phase space. Function $\mathbf{f}(\mathbf{x}, \mathbf{r}, \mathbf{t})$ is the probability distribution for the position and momentum of particles at time $\mathbf{t}$ where $\mathbf{x}$ and $\mathbf{r}$ are the spatial and velocity vector, respectively. The physical quantities, including the density ρ and the momentum $\rho\mathbf{u}$, can then be obtained by evaluating the first moments of function $\mathbf{f}$.

In this section, the major steps of the procedures of Lattice Boltzmann simulation are outlined. The particle distribution function $f(\mathbf{x}, \mathbf{r}, \mathbf{t})$ can be expressed by the Boltzmann equation (Körner et al., 2006):

$$\frac{\partial f}{\partial t} + \mathbf{r} \cdot \frac{\partial f}{\partial \mathbf{x}} = \Omega(f, f) \quad (5.1)$$

The collision term $\Omega(\mathbf{f}, \mathbf{f})$ representing the rate of change of the particle distribution due to collisions. In this study, the Bhatnagar-Gross-Krook (BGK) model (Bhatnagar et al., 1954) is used to simplify the collision integral for the near equilibrium state as follow

$$\Omega(f, f) = -\frac{1}{\lambda}(f - f^0) \quad (5.2)$$

where $\mathbf{f}^0$ is the Maxwell-Boltzmann equilibrium distribution function and λ is the relaxation time which controls the rate of approaching equilibrium, or in other words the viscosity of the fluid.

To solve f numerically, Equation (5.1) is first discretized in the velocity space using a finite set of velocity vector $\vec{e}_i$ ($i = 0, \dots, N$) leading to the velocity discrete Boltzmann equation (Qian et al., 1992):

$$\frac{\partial f_i}{\partial t} + \vec{e}_i \cdot \frac{\partial f_i}{\partial \mathbf{x}} = -\frac{1}{\lambda}(f_i - f_i^{\text{eq}}), \quad i = 0, \dots, N \quad (5.3)$$

where $f_i(\mathbf{x}, t)$ is equivalent to $f(\mathbf{x}, \vec{e}_i, t)$ and f_i^{eq} is the equilibrium distribution function and can be written in the following form

$$f_i^{\text{eq}} = \rho \omega_i \left[1 + \frac{3}{c^2} \vec{e}_i \cdot \mathbf{u} + \frac{9}{2c^2} (\vec{e}_i \cdot \mathbf{u})^2 - \frac{3}{2c^2} \mathbf{u} \cdot \mathbf{u} \right] \quad (5.4)$$

with $\mathbf{c} = \frac{\Delta \mathbf{x}}{\Delta t}$ is the lattice speed and ω_i is the weighting factor.

In this study, the UPC-LBM model is developed to simulate two dimensional flow, therefore the 9-velocity D2Q9 is utilized (Qian et al., 1992). The velocity vector $\vec{e}_i$ and the weighting factor ω_i for D2Q9 model are given below

$$\vec{e}_i = \begin{cases} (0, 0), & i = 0 \\ (1, 0), (0, 1), (-1, 0), (0, -1), & i = 1, 2, 3, 4 \\ (1, 1), (-1, 1), (-1, -1), (1, -1), & i = 5, 6, 7, 8 \end{cases} \quad (5.5)$$

$$\omega_i = \begin{cases} \frac{4}{9}, & i = 0 \\ \frac{1}{9}, & i = 1, 2, 3, 4 \\ \frac{1}{36}, & i = 5, 6, 7, 8 \end{cases} \quad (5.6)$$

The macroscopic fluid density can be defined as a summation of microscopic particle distribution function as

$$\rho = \sum_{i=0}^N f_i^{\text{eq}} \quad (5.7)$$

The macroscopic velocity $\mathbf{u}$ is averaged by

$$\mathbf{u} = \frac{1}{\rho} \sum_{i=0}^8 \vec{e}_i f_i^{\text{eq}} \quad (5.8)$$

The *streaming* and *collision processes* are now obtained in the following explicit form:

$$f_i(\mathbf{x} + \vec{e}_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = - \frac{[f_i(\mathbf{x}, t) - f_i^{\text{eq}}(\mathbf{x}, t)]}{\varrho} \quad (5.9)$$

where $\varrho = \frac{\lambda}{\Delta t}$ is the dimensionless relaxation time and is related to the fluid kinematic viscosity ν by

$$\nu = \frac{2\varrho - 1}{6} \frac{(\Delta x)^2}{\Delta t} \quad (5.10)$$

The right-hand side of equation (5.9) is called the *collision step* and the left-hand side *streaming step*. For the collision step, the equilibrium distribution function has to be calculated at each cell and at each time step from the local density ρ using (5.7) and the local macroscopic flow velocity $\mathbf{u}$ using (5.8).

5.3 UPC Architecture

The Lattice Boltzmann algorithms as outlined in Section 5.2 can be implemented easily as a parallelization application. However, for realistic applications, the LBM model is always computationally demanding since it needs a high spatial resolution and small time steps. Therefore, a number of studies has been conducted to investigate the work-load reduction algorithms for this LBM parallelization implementation. Numerous improvements are possible starting from this implementation such as running a numerical analysis to avoid calculating the components with $\vec{e}_i = 0$; or performing a vector and cache optimization algorithms to minimize the inner communication cost. The details can be found in (Iglberger, 2003; Donath, 2004; Pohl et al., 2004; Wellein et al., 2006).

In this study, we focus on developing the data storage algorithm to maximize the data locality and minimize the access time made by threads while reading-writing to the

memory section. In the following section, the computational structure developed based on the outline LBM algorithm in Sections 5.2 is presented, and the data divisions and storage are then described.

5.3.1 Computational structure

The implementation of the outline LBM algorithm is summarized in the Figure 5.1. Each of the step (from streaming to collision and propagation) is performed on all particle cells during each time step and executed using a 2-layer nested loop. The complexity of the nested loop algorithm is $O(N^2)$, where N is the number of computational cells in a singular direction. Each of the nested loop is then paralleled using a work-sharing function, termed as `upc_forall`. The details of the UPC work-sharing function `upc_forall` is further discussed later. After every step (i.e. before a next nested loop), a synchronization point is inserted using an UPC built-in function, namely `upc_barrier`, to synchronize all threads.

Figure 5.1: UPC-LBM computational structure

5.3.2 Work-sharing function

The UPC compiler provides a number of built-in identifiers and functions, allowing the control and tracking of the threads' activities. The main identifiers that have been used in the UPC-LBM model include (i) **THREADS**: a global property used to determine the total number of activated threads in the program and (ii) **MYTHREADS**: a unique number used to identify each thread. **MYTHREADS** is numbered from 0 to **THREADS**-1. **THREADS** and **MYTHREADS** are used by `upc_forall` to distribute the computational workload within the nested loops. An example of `upc_forall` distributing the workload of the streaming step into threads is shown in Algorithm 5.3.1.

Algorithm 5.3.1

```
1: upc_forall ( $t = 0$ ;  $t < \text{THREADS}$ ;  $++t$ ;  $t$ )
2:   for each row from  $\text{thread}_{\text{start}}(t)$  to  $\text{thread}_{\text{end}}(t)$ 
3:     for each column of the 2D domain
4:       stream: move  $f_i \rightarrow f_i^*$  in the  $e_i$  direction
5:       update fluid density
6:       update local hydrodynamic velocity
7:     end for
8:   end for
9: end upc_forall
```

The `upc_forall` function uses the single-program multiple-data method (SPMD) to divide the original massive work-load into number of sub-workloads and assign them to threads for the calculation processes. Each sub-workload is marked using $\text{thread}_{\text{start}}$ and $\text{thread}_{\text{end}}$ properties, which are pre-computed by the domain decomposition algorithms. The illustration for $\text{thread}_{\text{start}}$ and $\text{thread}_{\text{end}}$ is shown in the Figure 5.2.

Figure 5.2: Illustration for $\text{thread}_{\text{start}}$ and $\text{thread}_{\text{end}}$ in UPC-LBM

5.3.3 Domain decomposition algorithm

For UPC-LBM, the fluxes are computed via a row-by-row manner requiring the respective data value of the upper and previous cells. In addition, the row implementation works better with the UPC compiler compared to the patched implementation (Prugger et al., 2016). Therefore, in UPC-LBM, the original 2D Cartesian computational matrix is divided into sub-domains by rows, for which each sub-domain has the affinity to a single thread. In particular, with T number of threads, a $N * N$ domain is decomposed into T sub-domains D_i ($i = 0$ to $T - 1$) and D_i has affinity with T_i . An example of domain decomposition on a 2D domain with the thread affinity is demonstrated in the Figure 5.3.

In this study, we adopted the algorithm proposed by Vu et al. (2017) to calculate the locations $\text{thread}_{\text{start}}$ and $\text{thread}_{\text{end}}$. In short, (i) if N is divisible by T , then each D_i contains the data of $\frac{N}{T}$ rows which results in $\frac{N*N}{T}$ cells, or (ii) if N is not divisible by T , then the first $(T-1)$ sub-domains D_i ($i = 0$ to $T - 2$) contain $(\text{int}(\frac{N}{T}) + 1)$ rows and the last sub-domain D_{T-1} contains the remaining rows.

Figure 5.3: Example of domain decomposition on a 5x9 2D domain with 3 threads.

5.3.4 Data storage algorithm

UPC represents a variant instant of the PGAS paradigm with additional private address spaces for local computations (El-Ghazawi et al., 2005). Under UPC, memory is composed of a locally partitioned shared memory space and additional private memory sections, as shown in the Figure 5.4. The shared memory spaces can be accessed by all thread while the private data sections can only be read and written by their affinity thread. The shared space, however, is logically divided into portions, each with a special association (i.e. affinity) to a given thread. In this way, UPC enables the programmer to keep the shared data that will be dominantly processed by a given thread associated with that thread.

Figure 5.4 UPC memory structure.

While developing the data storage algorithm for UPC application, one can choose between (i) shared UPC: storing all of the data in the shared space although only a small part of the data is needed to be accessed by multiple threads (i.e. the global data) and (ii) hybrid UPC: storing the global data in the shared spaces while the remaining data in the private memory section (El-Ghazawi et al., 2005). El-Ghazawi et al. (2005) has measured the memory access time for a UPC thread to read and write to a private and shared memory sections. The results have shown that the private memory access time is shorter than that shared space.

Examples of shared UPC data storage model include ParaFlood2D and UPC-CFD developed by Vu et al. (2016, 2017). The blocked-cyclic techniques that have been utilized in ParaFlood2D and UPC-CFD have been summarized in Figure 5.5. Although all of the data are stored in the shared memory section, Vu et al. (2016, 2017) prioritized the data locality and still achieved the good performance of the UPC model.

Figure 5.5. Sub-domains division and data storage algorithm for numerical domain (Vu et al., 2016)

In this study, two versions of UPC-LBM i.e. shared UPC-LBM and hybrid UPC-LBM were developed. Figure 5.6 demonstrates the memory allocation for the two versions. Their performances were evaluated and discussed in the next section.

Figure 5.6. Memory allocation for: (i) hybrid UPC-LBM and (ii) shared UPC-LBM.

5.4 Results and Discussion

To validate the implemented numerical scheme in UPC-LBM, the numerical predictions derived for the Poiseuille's flow were compared with the analytical solution. The physical dimensions of the deployed numerical domain for the Poiseuille's flow are summarized in Table 4.1 and Fig. 4.2b.

The steady Poiseuille's flow has the exact solution as:

$$u(x, y, t) = \frac{\Delta p}{2\mu L} y(y - H) \quad (5.11)$$

$$v(x, y, t) = 0 \quad (5.12)$$

The criterion of steady state is:

$$\frac{\sum_{ij} |u_{ij}^{n+1} - u_{ij}^n|}{\sum_{ij} |u_{ij}^{n+1}|} \leq 5.0 * 10^{-9} \quad (5.13)$$

where u_{ij}^n is the horizontal velocity component on (x_i, y_j) at the n^{th} time step.

The numerical predictions and analytical solution for $u(y)$ are compared in Fig. 5.8 at the steady state. Overall, it can be seen that the two agree well with each other, demonstrating the high accuracy of the numerical prediction.

By using the speedup parameter in Equation 3.10, the parallelism performance of UPC-LBM was evaluated using a SGI UV-2000 server for the Poiseuille's flow case with an ultra-large numerical domains of 100 million cells.

Figure 5.7. Comparison between numerical predictions and analytical solutions for Poiseuille's flow at location $x = 0.5m$

The parallelization efficiency of UPC-LBM model was evaluated via a speedup comparison between the (i) Hybrid private/shared UPC-LBM version; (ii) Native Shared UPC-LBM version and (iii) Poiseuille's flow demo code provided by OpenLB, which is parallelized using MPI protocol (OpenLB). The comparison conducted in this study is relative due to several reasons such as (i) the numerical scheme utilized by OpenLB for Poiseuille's flow differs from the outline LBM algorithms in Section 5.2 and (ii) the runtime measurement methods are different between the two. The speedup comparison is summarized in Fig. 5.9.

Table 5.1: SGI UV-2000 cluster used for model testing

Cluster	Node CPUs	CPU speeds (GHz)	Cores per node	Node RAM (TB)	Available cores	Communication switch
SGI UV-2000	Intel Xeon E5-4657LV & E5-2670	2.4	8/16	2	up to 100 cores	InfiniBand Shared-memory

Figure 5.8. Comparison of speedup among hybrid UPC-LBM, shared UPC-LBM and OpenLB-Poiseuille MPI

Generally, the three achieved near ideal speedup up to 32 cores. This was expected as both versions of UPC can exploit the advantages of the data locality. Thus, they were able to read and write directly to the internal memory section without experiencing any delay. Furthermore, with the small number of cores, the amount of time for messages querying and retrieving with MPI was trivial. However, from 32 cores onward to the maximum of 100 cores, Hybrid UPC-LBM and Shared UPC-LBM outperformed MPI significantly. In fact, UPC remains effective in term of computational speedup with no sign of performance decline after reaching the maximum. In other words, further acceleration can still be achieved if additional cores are available.

Despite having a slower speedup than that both versions of UPC, the MPI implementation of OpenLB-Poiseuille still achieved significant speedup up to 64 cores. The performance of the OpenLB-Poiseuille model however became ineffective beyond that. In particular, the MPI speedup was 23.5 and 46.6 with 32 and 64 cores, respectively; but it was only 50.2 with 100 cores. Although it was unclear how the runtime was measured in OpenLB-Poiseuille (whether it included the initialization

and output steps or it just took into account the main solver run time), this behavior of the model still revealed the limitation of MPI. In the MPI architecture, when the number of processing cores increases, the amount of message passing in the system also expands exponentially. Eventually, the message processing time surpasses the actual computational time for each CPU, and limits any further gain in performance.

Both shared and hybrid UPC-LBM showed outstanding performances in the evaluation test with the speedup rates of 89.9 and 95.1 at 100 processing cores, respectively. The hybrid UPC-LBM version demonstrated a slightly better speedup compared to the shared UPC approach, which was expected as the time needed to access the private section in the hybrid design is shorter than that for accessing the shared memory. However, the difference in the performances between the two was trivial. This can be explained as (i) the UPC-LBM model was designed to minimize the number of memory access made by thread with the row-by-row domain decomposition, leading to the fact that the total memory access time was small compared to the total execution time for the main solver; and (ii) the multiple synchronization points used in the UPC-LBM model after each nested loop may affect the overall performance. Thus, for developing the UPC parallelization of the LBM model, the shared approach is preferred over the hybrid design because it is much simpler to implement especially when upgrading the model to 3-dimensions.

Chapter 6

Conclusions and outlook

The major conclusions of the work are outlined in this chapter followed by a list of recommended areas for future work.

6.1 Conclusions

In this thesis, an alternative computer architecture, namely UPC-PGAS, was combined with specific numerical schemes for the computational flow simulations including: (1) the 2nd-order Godunov-type MUSCL scheme; (2) the explicit predictors/correctors scheme from the Lax-Wendroff family and (3) D2Q9 Lattice Boltzmann method. The combination explored the benefits of (a) ease of implementation, and (b) maintaining a near linear or exponential relationship between the computational speedup and the number of CPUs introduced.

The first parallelization work was with TELEMAC 2D, a computational flood model based on the Shallow Water equations (i.e. St Venant equations) with the finite element numerical scheme. It utilized the message passing interface protocol (MPI) architecture and the computational speedup minimized when the number of computational cores (CPUs) increased. A further increase to the number of CPUs, in fact, increased the total computational runtime, which thus developed into a bottleneck issue. The exercise demonstrated the common issue of the MPI-based CFD software in modelling the massive scale flow simulation with large number of

cores. The CFD software often requires numerous information exchange activities between computational cores for transferring and updating the data of the flow conditions. When using the MPI approach, the internal communication of CFD software is executed by sending information package back and forth between those cores, which raises the latency when the number of cores increase beyond a certain limit. The new combination of UPC-PGAS and CFD numerical schemes was expected to address the preceding the above statements.

In the second phase of this study, the viability of the PGAS-UPC architecture was examined in the computational hydraulic simulations. The new flood inundation model, ParaFlood2D, was developed using the UPC approach as the implementation solution for parallel computation. ParaFlood2D used the second-order Godunov-type monotone upstream scheme for conservation law (MUSCL) with second-order accuracy to solve the St Venant Equations. The architecture algorithms of ParaFlood2D, including domain discretization and data distribution, were tailored to capitalize on the numerical scheme adopted. In particular, the UPC work-sharing functions were used to distribute the workload of two main phases in numerical scheme i.e. flux predictors and correctors; the internal communication was processed via the Global Address to avoid the bottle neck; and the model was speeded up by using the affinity data-storage techniques. The accuracy of ParaFlood2D was first verified using a simple 1D dam break case. The computational efficiency of the model was then evaluated with both the 1D dam break and 2D flood wave propagation cases, using the two different hardware systems of the 100-core shared-memory machine and the 128-core distributed-memory system. The results demonstrated the good performance of UPC over the OpenMP and MPI implementation of the identical numerical scheme at their basic designs. The trend of speedup ratio of UPC also remained close to the ideal Amdahl law up to the maximum cores available in this study. The results, therefore demonstrated the viability of flood modelling using the UPC architecture for parallelization.

Both the viability of UPC architecture in large-scale computational flow simulation as well as its flexibility in a new area with the different type of numerical scheme were examined in the next phase of the study. A new model, termed as UPC-CHD,

was developed on the working architecture of PGAS-UPC coupled with the traditional 2-step explicit scheme from the Lax-Wendroff family of predictors-correctors. Number of techniques were introduced to customize the PGAS concept for the Lax-Wendroff explicit scheme including synchronization algorithm for data-conflict avoidance, blocked-cyclic method for flow conditions data storage and computational loop refactoring approach for minimizing the computational complexities. The accuracy of the implemented numerical scheme in UPC-CHD was first examined on three incompressible, viscous flow cases having moderate flow velocities under laminar conditions. The parallelism performance of UPC-CHD was then evaluated by comparing the total computational runtime with that of MPI and OpenMP at their basic designs on an SGI UV-2000 server. UPC performed better than MPI and OpenMP at their basic designs, with a near 1:1 speedup till 100 cores available in this study. The efficient parallelism performance achieved by UPC-CHD demonstrates its capability to exploit data locality and adopt affinity optimization to ensure maximization of the parallelism performance.

Finally, the viability of UPC in the flow simulations using Lattice Boltzmann method was examined. The UPC-LBM model is developed using D2Q9 setting i.e. two-dimensional and nine possible directions for streaming processes. The accuracy of UPC-LBM was investigated using the incompressible, viscous flow Poiseuille's flow. The parallel performance of UPC-LBM was then evaluated by relatively comparing with the OpenLM open source for the same case. The UPC-LBM maintained the good performance with a near 1:1 speedup till 100 cores available as expected.

In conclusion, the study demonstrated that the UPC architecture can be successfully implemented and customized for specific numerical schemes to accelerate the computational runtime for the massive scale simulation using large number of cores.

6.2 Future work

Future developments are needed to further develop UPC for fluid flow simulation with improved features on numerical schemes, mesh generation, boundary condition and turbulence. Implementation on supercomputer with GPU hybrid architecture also needs to be further examined. The details of future developments for UPC are discussed in the following.

1) Turbulence: In this study, we have confined the scope to classical numerical schemes and laminar flow cases. To make the model to be useful for practical engineering applications, improvements turbulent flow modelling and unstructured numerical meshes, etc., need to be sought.

2) Implicit schemes: Too large a time step can be unstable and diverge in some cases. Hence, small time step should be available for usage.

3) Immersed Boundary Conditions and Unstructured Mesh: The immersed boundary conditions should be implemented for the computational fluid flow model to reduce the error in the modelling of boundaries, especially if an object is not aligned along the main axes of the simulation domain. The immersed boundary conditions offer a higher order treatment of boundaries instead of a step-wise approximation of boundaries. Furthermore, to emulate actuality in a realistic fluid flow events for both predictive and analytical purposes, the coupling of the dynamic meshing algorithm with the UPC architecture will be necessary.

4) Sub-domain time-step treatment: The dynamic meshing is an inevitable development step for the UPC model; the sub-domain time-step treatment, therefore, should also be implemented. The model is expected to be capable of choosing a smaller time-step on sub-domain with finer grids and a larger on a coarser grid, eliminating the need for one global small time step.

5) Supercomputer Test: The UPC model is currently tested up to 100 cores on the SGI server and 128 cores on the IBM machine due to the limited hardware available in this study. The performance evaluation with a larger number of cores should be done for examining the capability and at the same time, the limitation of UPC

architecture.

6) Hybrid Architecture: As mentioned previously, more and more supercomputers are equipped with accelerators of GPUs. A highly hybrid programming concept can be applied, by performing the intra– and inter- node communication with UPC Global Address techniques and the programming of the graphic card with CUDA for example.

Nonetheless, we are hopeful that with the further improvement, the model can serve as a viable alternative for large-scale computational fluid simulations in the future.

References

- [1]. Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. Proceedings, Spring Joint Computer Conference, Atlantic City, New Jersey, USA, April 18-20, 1967.
- [2]. Anderson, J. D. (2009), "Governing Equations of Fluid Dynamics," In J. F. Wendt (Ed.), Computational Fluid Dynamics. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15 - 51.
- [3]. Appleyard, J. and Drikakis, D. (2011), "Higher-order CFD and interface tracking methods on highly-Parallel MPI and GPU systems," Computers & Fluids, 46 (1), 101-105.
- [4]. Aubry, R., Houzeaux, G., Vázquez, M., and Cela, J. M. (2011). Some useful strategies for unstructured edge-based solvers on shared memory machines. International Journal for Numerical Methods in Engineering, 85(5), 537-561.
- [5]. B. van Leer (1974). Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme. Journal of Computational Physics, 14(4), 361-370.
- [6]. Bates, P. D., Anderson, M. G., Baird, L., Walling, D. E., and Simm., D. (1992). Modelling floodplain flows using a two-dimensional finite element model. Earth Surface Processes and Landforms, 17(6), 575-588.
- [7]. Bates, P. D., Stewart, M. D., Siggers, G. B., Smith, C. N., Hervouet, J. M., and Sellin. R. H. J. (1998). Internal and external validation of a two-dimensional finite element code for river flood simulations. Proceedings of the Institution of Civil Engineers: Water and Maritime Engineering, 130(3), 127-141

- [8]. Bates, P. D., and De Roo, A. P. J. (2000). A simple raster-based model for flood inundation simulation. *Journal of Hydrology*, 236(1–2), 54-77.
- [9]. Bella, G., Filippone, S., Rossi, N., and Ubertini, S. (2002). Using openMP on a hydrodynamic lattice-Boltzmann code. *Proceedings of the Fourth European Workshop on OpenMP* (54), Roma.
- [10]. Berkeley UPC Compiler 2.22.2, 2016. <http://upc.lbl.gov/download/> (accessed 10.07.16)
- [11]. Berger, M. J., Aftosmis, M. J., Marshall, D. D. and Murman, S. M. (2005), "Performance of a new CFD flow solver using a hybrid programming paradigm," *Journal of Parallel and Distributed Computing*, 65 (4), 414-423.
- [12]. Bhatnagar, P. L., Gross, E. P., and Krook, M. (1954). A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*, 94(3), 511-525.
- [13]. Bradford, S. F., and Sanders, B. F. (2002). Finite-volume model for shallow-water flooding of arbitrary topography. *Journal of Hydraulic Engineering*, 128(3), 289-298.
- [14]. Brandvik, T, and Pullan, G. (2008), "Acceleration of a 3D Euler Solver Using Commodity Graphics Hardware," in 46th AIAA Aerospace Sciences Meeting and Exhibit, American Institute of Aeronautics and Astronautics.
- [15]. Bucs, S. S., Radu, A. I., Lavric, V., Vrouwenvelder, J. S. , and Picioreanu, C. (2014), "Effect of different commercial feed spacers on biofouling of reverse osmosis membrane systems: A numerical study," *Desalination*, 343, 26-37.
- [16]. Casulli, V., and Zanolli, P. (2002). Semi-implicit numerical modeling of nonhydrostatic free-surface flows for environmental problems. *Mathematical and Computer Modelling*, 36(9), 1131-1149.
- [17]. Cekmer, O., Um, S., and Mench, M. M. (2016). A combined path-percolation – Lattice-Boltzmann model applied to multiphase mass transfer in porous media. *International Journal of Heat and Mass Transfer*, 93, 257-272.
- [18]. Chen, S., and Doolen, G. D. (1998). Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, 30 (1998), pp. 282-300.

- [19]. Djomehri, M. J. and Jin, H.H. (2002), "Hybrid MPI+OpenMP Programming of an Overset CFD Solver and Performance Investigations," , NASA Ames Research Center, Moffett Field, CA United States, Tech. Rep.
- [20]. Chung, T. J. (2002). Computational Fluid Dynamics: Cambridge University Press.
- [21]. Desplat, J.-C., Pagonabarraga, I., and Bladon, P. (2001). LUDWIG: A parallel Lattice-Boltzmann code for complex fluids. Computer Physics Communications, 134(3), 273-290.
- [22]. Donath, S. (2004). On Optimized Implementations of the Lattice Boltzmann Method on Contemporary High Performance Architectures. Bachelor thesis, Lehrstuhl für Informatik 10 (Systemsimulation), Universität Erlangen-Nürnberg.
- [23]. Dressler, R. F. (1958). Unsteady Non-Linear Waves in Sloping Channels. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 247(1249), 186-198.
- [24]. El-Ghazawi, T., Cantonnet, F., Yao, Y., and Rajamony, R. (2005). Developing an Optimized Upc Compiler for Future Architectures. Technical report, IDA Center for Computing Sciences.
- [25]. El-Ghazawi, T. A., Carlson, W., Sterling, and Yelick. T. K. (2005), "UPC: Distributed Shared Memory Programming," John Wiley and Sons, ISBN: 0-471022048-5.
- [26]. El-Ghazawi, T., Carlson, W., Sterling, T., & Yelick, K. (2005). Programming View and UPC Data Types, UPC (17-31): John Wiley & Sons, Inc.
- [27]. ExaCorporation. Exa Simulation Software Solutions — Exa Corporation (<http://exa.com>).
- [28]. Frisch, J. (2014), "Towards Massive Parallel Fluid Flow Simulations in Computational Engineering", (Doctoral dissertation). Retrieved from the Technical University Munich database. (ISBN: 978-3-8440-3164-5)
- [29]. Gao. (1996). NDWI—A normalized difference water index for remote sensing of vegetation liquid water from space. Remote Sensing of Environment, 58(3), 257-266.

- [30]. George DL. Numerical approximation of the nonlinear shallow-water equations with topography and evolving dry beds (MSc dissertation). University of Washington, WA, U.S.A., 2004
- [31]. Giovannini, M., Marconcini, M., Arnone, A. (2015), " A Hybrid Parallelization Strategy of a CFD Code for Turbomachinery Applications," in 11th European Conference on Turbomachinery Fluid dynamics & Thermodynamics, Madrid Spain.
- [32]. Girimaji, S. (2012). Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes. AIAA Journal, 51(1), 278-279.
- [33]. Gourdain, N., Gicquel, L., Montagnac, M., Vermorel, O., Gazaix, M., Staffelbach, G. (2009), "High performance parallel computing of flows in complex geometries: I. Methods," Computational Science & Discovery, 2 (1), 015003.
- [34]. Gomes Pereira, L. M., and Wicherson, R. J. (1999). Suitability of laser data for deriving geographical information. A case study in the context of management of fluvial zones. ISPRS Journal of Photogrammetry and Remote Sensing, 54(2-3), 105-114.
- [35]. Grunau, D., Chen, S., and Eggert, K. (1993). A lattice Boltzmann model for multiphase fluid flows. Physics of Fluids A: Fluid Dynamics, 5(10), 2557-2562.
- [36]. Gupta, N., Gorthi, R. C., and Mishra, S. C. (2006). Lattice Boltzmann Method Applied to Variable Thermal Conductivity Conduction and Radiation Problems. Journal of Thermophysics and Heat Transfer, 20(4), 895-902.
- [37]. Habich, J., Zeiser, T., Hager, G., and Wellein, G. (2011). Performance analysis and optimization strategies for a D3Q19 lattice Boltzmann kernel on nVIDIA GPUs using CUDA. Advances in Engineering Software, 42(5), 266-272.
- [38]. Hall, J. W., Sayers, P. B., and Dawson, R. J. (2005). National-scale assessment of current and future flood risk in England and Wales. Natural Hazards, 36(1-2), 147-164.

- [39]. Han-Taw, C., and Jae-Yuh, L. (1993). Numerical analysis for hyperbolic heat conduction. *International Journal of Heat and Mass Transfer*, 36(11), 2891-2898.
- [40]. He, X., and Luo, L. S. (1997a). A priori derivation of the lattice Boltzmann equation. *Phys. Rev. E*, 55 (1997), R6333-R6336.
- [41]. He, X., and Luo, L. S. (1997b). Theory of the Boltzmann method: from the Boltzmann equation to the lattice Boltzmann equation. *Phys. Rev. E*, 56 (1997), 6811-6817.
- [42]. Heather, Kline. "Su2code/SU2." GitHub. SU2/Open-source CFD Code, 13 Jan. 2017. Web. 31 Mar. 2017.
- [43]. Hervouet. (2000). A high-resolution 2-D dam-break model using parallelization. *Hydrological Processes*, 14(13), 2211-2230.
- [44]. Hervouet. (2007). *Hydrodynamics of Free Surface Flows Modelling with the Finite Element Method*. Wiley. 978-0-470-03558-0.
- [45]. Ho, J.-R., Kuo, C.-P., Jiaung, W.-S., and Twu, C.-J. (2002). Lattice Boltzmann scheme for hyperbolic heat conduction equation. *Numerical Heat Transfer, Part B: Fundamentals*, 41(6), 591-607.
- [46]. Hodges, B. R. (2004). Accuracy Order of Crank–Nicolson Discretization for Hydrostatic Free-Surface Flow. *Journal of Engineering Mechanics*, ASCE, 130(8).
- [47]. Hunter, N. M., Bates, P. D., Horritt, M. S., and Wilson, M. D. (2007). Simple spatially-distributed models for predicting flood inundation: a review. *Geomorphology*, 90(3–4), 208-225.
- [48]. Jajcevic, D., Siegmund, E., Radeke, C., and Khinast, J. G. (2013), "Large-scale CFD–DEM simulations of fluidized granular systems," *Chemical Engineering Science*, 98, 298-310.
- [49]. Jamshed, S. (2015), "Chapter 3 - The Way the HPC Works in CFD," in *Using HPC for Computational Fluid Dynamics*, ed Oxford: Academic Press, 41-79.
- [50]. Jeffrey, R. K., Stephen, G. M., and Edward, S. G. (1999). Evaluation of Advective Schemes for Estuarine Salinity Simulations. *Journal of Hydraulic Engineering*, ASCE, 125(1), 32-46.

- [51]. Johnson, A. A. (2006), "Using Unified Parallel C to Enable New Types of CFD Applications on the Cray X1/E", Cray User Group Conference.
- [52]. Kalyanapu, A. J., Shankar, S., Pardyjak, E. R., Judi, D. R., and Burian, S. J. (2011). Assessment of GPU computational enhancement to a 2D flood model. *Environmental Modelling and Software*, 26(8), 1009-1016.
- [53]. Kalyanapu, A. J., Judi, D. R., McPherson, T. N., and Burian, S. J. (2012). Monte Carlo-based flood modelling framework for estimating probability weighted flood risk *Journal of Flood Risk Management*, 5(1), 37-48.
- [54]. Kermani, M. and Plett, E. (2001), "Roe scheme in generalized coordinates. I - Formulations," in 39th Aerospace Sciences Meeting and Exhibit, ed: American Institute of Aeronautics and Astronautics.
- [55]. Kermani, M. and Plett, E. (2001), "Roe scheme in generalized coordinates. II - Application to inviscid and viscous flows," in 39th Aerospace Sciences Meeting and Exhibit, ed: American Institute of Aeronautics and Astronautics.
- [56]. Körner, C., Pohl, T., Råde, U., Thürey, N., and Zeiser, T. (2006). Parallel Lattice Boltzmann Methods for CFD Applications. In A. M. Bruaset and A. Tveito (Eds.), *Numerical Solution of Partial Differential Equations on Parallel Computers* (439-466). Berlin, Heidelberg
- [57]. Khaitan, and McCalley. (2014). SCALE: A hybrid MPI and multithreading based work stealing approach for massive contingency analysis in power systems. *Electric Power Systems Research*, 114(0), 118-125.
- [58]. Krzhizhanovskaya, V. V., Melnikova, N. B., Chirkin, A. M., Ivanov, S. V., Boukhanovsky, A. V., and Slood, P. M. A. (2013). Distributed Simulation of City Inundation by Coupled Surface and Subsurface Porous Flow for Urban Flood Decision Support System. *Procedia Computer Science*, 18, 1046-1056.
- [59]. Kumm, Tes, Yin, Adamson, Józsa, Koponen, Sarkkula. (2014). Water balance analysis for the Tonle Sap Lake–floodplain system. *Hydrological Processes*, 28(4), 1722-1733.
- [60]. Kuo, F. A., Smith, M. R., Hsieh, C. W., Chou, C. Y., and Wu, J. S. (2011), "GPU acceleration for general conservation equations and its application to several engineering problems," *Computers & Fluids*, 45 (1), 147-154.

- [61]. Kurowski, K., Kulczewski, M., and Dobski, M. (2011) Parallel and GPU based strategies for selected CFD and climate modeling models. *Environmental Science and Engineering* (Subseries: Environmental Science) (735-747)
- [62]. Lai, W., and Khan, A. A. (2016). A Parallel Two-Dimensional Discontinuous Galerkin Method for Shallow-Water Flows Using High-Resolution Unstructured Meshes. *Journal of Computing in Civil Engineering*. DOI: [http://dx.doi.org/10.1061/\(ASCE\)CP.1943-5487.0000647](http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000647)
- [63]. Lee, Sang-Heon (2010). Numerical modelling of rapidly varied river flow (Doctoral dissertation). Retrieved from the University of Nottingham repository.
- [64]. Iglberger, K. (2003). Cache Optimizations for the Lattice Boltzmann Method in 3D. *Studienarbeit, Lehrstuhl für Informatik 10 (Systemsimulation), Universität Erlangen-Nürnberg*.
- [65]. Li, W., Wei, X., and Kaufman, A. (2003). Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19(7), 444-456.
- [66]. Li, Y .L., Lin, P. J., Tung, K. L. (2011), "CFD analysis of fluid flow through a spacer-filled disk-type membrane module," *Desalination*, 283, 140-147.
- [67]. Liu, Lu, Liu, Jin, and Wang. (2009). Pinpointing the sources and measuring the lengths of the principal rivers of the world. *International Journal of Digital Earth*, 2(1), 80-87.
- [68]. Liu, Y., Schmidt, B., and Maskell, D. L. (2011). DecGPU: Distributed error correction on massively parallel graphics processing units using CUDA and MPI. *BMC Bioinformatics*, 12:85. doi:10.1186/1471-2105-12-85.
- [69]. Manning's Roughness Coefficients for common materials (2016). www.engineeringtoolbox.com/mannings-roughness-d_799.html (accessed 29.12.16)
- [70]. Martinez, D. O., Chen, S., and Matthaeus, W. (1994). Lattice Boltzmann magnetohydrodynamics. *Phys. Rev. E*, 47 (1994), R2249-R2252.
- [71]. Mazzeo, M. D., and Coveney, P. V. (2008). HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. *Computer Physics Communications*, 178(12), 894-914.

- [72]. McFeeters. (1996). The use of the Normalized Difference Water Index (NDWI) in the delineation of open water features. *International Journal of Remote Sensing*, 17(7), 1425-1432.
- [73]. Morgan, A., Nathalie, B., Olivier, D., Claire-Marie, D., and Philippe, G. (2015). Global Sensitivity Analysis in a 2D high resolution hydraulic modeling application – Sobol index maps to rank uncertain parameters. *Proceedings, 36th IAHR World Congress, 28 June – 3 July, 2015, The Hague, the Netherlands.*
- [74]. MRC (2005). Overview of the Hydrology of the Mekong Basin. *Mekong River Commission Report*, ISSN: 1728 3248.
- [75]. MRC (2011) Flood Situation Report, November 2011, MRC Technical Paper No.36, Mekong River Commission, 57pp.
- [76]. Munson, B. R., Young, D. F., and Okiishi, T. H. (2006). *Fundamentals of fluid mechanics*. 6th Ed., Hoboken, NJ: J. Wiley & Sons, 6, 263-331
- [77]. Neal, J., Fewtrell, T., and Trigg, M. (2009). Parallelisation of storage cell flood models using OpenMP. *Environmental Modelling & Software*, 24(7), 872-877.
- [78]. Neal, J., Fewtrell, T. J., Bates, P. D., and Wright, N. G. (2010). A comparison of three parallelisation methods for 2D flood inundation models. *Environmental Modelling & Software*, 25(4), 398-411.
- [79]. Next Limit Technologies. XFlow is a next generation CFD software system from Next Limit Technologies that uses a proprietary, particle-based, meshless approach which can easily handle traditionally complex problems (<http://www.xflowcf.com>).
- [80]. Niemeyer, K. E., and Sung, C. J. (2014), "Recent progress and challenges in exploiting graphics processors in computational fluid dynamics," *The Journal of Supercomputing*, 67 (2), 528-564.
- [81]. Nguyen, Robson., and George. (2003). *Report on Residential Clusters in An Giang, Dong Thap and Long An Provinces in the Mekong Delta*. Adam Fforde and Associates, Melbourne, Australia.
- [82]. OpenLB. Open Source Lattice Boltzmann Code (<http://www.optilb.org>).
- [83]. Palabos. Palabos - CFD Complex Physics (<http://www.palabos.org>).

- [84]. Panday, S., and Huyakorn, P. S. (2004). A fully coupled physically-based spatially-distributed model for evaluating surface/subsurface flow. *Advances in Water Resources*, 27(4), 361-382.
- [85]. Pau, J. C., and Sanders, B. F. (2006). Performance of parallel implementations of an explicit finite-volume shallow-water model. *Journal of Computing in Civil Engineering*, 20(2), 99-110.
- [86]. Plate. (2007). Early warning and flood forecasting for large rivers with the lower Mekong as example. *Journal of Hydro-environment Research*, 1(2), 80-94.
- [87]. Pohl, T., Deserno, F., Thurey, N., Rude, U., Lammers, P., Wellein, G., and Zeiser, T. (2004, 06-12 Nov. 2004). Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures. Paper presented at the Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference.
- [88]. Prugger, M., Einkemmer, L., and Ostermann, A. (2016). Evaluation of the partitioned global address space (PGAS) model for an inviscid Euler solver. *Parallel Computing*, 60, 22-40.
- [89]. Puurtinen, T. A., Toivanen, J. I., Mattila, K., Hyväluoma, J., Nash, R. W., Coveney, P. V., and Timonen, J. (2017). Coupling of lattice-Boltzmann solvers with suspended particles using the MPI intercommunication framework. *Advances in Engineering Software*, 111, 52-57.
- [90]. Qian, Y. H., Humières, D. D., and Lallemand, P. (1992). Lattice BGK Models for Navier-Stokes Equation. *EPL (Europhysics Letters)*, 17(6), 479.
- [91]. Rabenseifner, R., Hager, G., Jost, G., and Keller, R. (2006). Hybrid MPI and OpenMP Parallel Programming. In B. Mohr, J. L. Träff, J. Worringer, and J. Dongarra (Eds.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 13th European PVM/MPI User's Group Meeting*, Bonn, Germany, September 17-20, 2006.
- [92]. Rabenseifner, R., Hager, G., and Jost, G. (2009). Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes. *Proceedings, 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Weimar, Germany.

- [93]. Rabus, B., Eineder, M., Roth, A., and Bamler, R. (2003). The shuttle radar topography mission - A new class of digital elevation models acquired by spaceborne radar. *ISPRS Journal of Photogrammetry and Remote Sensing*, 57(4), 241-262.
- [94]. Rao, P. (2005). A parallel RMA2 model for simulating large-scale free surface flows. *Environmental Modelling & Software*, 20(1), 47-53.
- [95]. Ritter, A. (1892). Die fortpflanzung der wasserwellen. *Zeitschrift des Vereins Deutscher Ingenieure*, 36, 947–954.
- [96]. Roe, P. L. (1981). Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2), 357-372.
- [97]. Roe, P. L. (1985). Some contributions to the modelling of discontinuous flows. *Lectures in Applied Mathematics*, 22, 163-193.
- [98]. Salvatore, F., Bernardini, M. and Botti, M. (2013), "GPU accelerated flow solver for direct numerical simulation of turbulent flows," *Journal of Computational Physics*, 235, 129-142.
- [99]. Samuels, P. G. (1990). Cross-section location in 1-D models. *Proceedings, International conference on river flood hydraulics*, 17-20 September, Wallingford, UK.
- [100]. Schalkwijk, J., Griffith, E. J., Post, F. H., and Jonker, H. J. J. (2012), "High-Performance Simulations of Turbulent Clouds on a Desktop PC: Exploiting the GPU," *Bulletin of the American Meteorological Society*, 93, 307-314.
- [101]. Simmendinger, C., Jägersküpper, J., Machado, R., Lojewski, C. (2011), "A PGAS-based Implementation for the Unstructured CFD Solver TAU", *Partitioned Global Address Space Programming Models*, Galveston Island.
- [102]. Sobieski, W. and Zhang, Q. (2017), "Multi-scale modeling of flow resistance in granular porous media," *Mathematics and Computers in Simulation*, 132, 159-171.
- [103]. Sousa, P., Soares, A., Monteiro, E., and Rouboa, A. (2014), "A CFD study of the hydrodynamics in a desalination membrane filled with spacers," *Desalination*, 349, 22-30, 2014.
- [104]. Tinh and Hang. (2003). *Living with Floods in the Mekong River Delta of Vietnam*, Department of Dike Management, Flood and Storm Control,

Ministry of Agriculture and Rural Development, Hanoi, Socialist Republic of Vietnam.

- [105]. Tolke, J., and Krafczyk, M. (2008). TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *Int. J. Comput. Fluid Dyn.*, 22(7), 443-456.
- [106]. Toro, E. F. (2009), "The Riemann Solver of Roe," in *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*, ed Berlin, Heidelberg: Springer Berlin Heidelberg, 11, 345-376.
- [107]. Tung T. Vu, Phuoc K.T. Nguyen, Lloyd H.C. Chua and Adrian W.K. Law (2015). Two-Dimensional Hydrodynamic Modelling of Flood Inundation for a Part of the Mekong River with TELEMAC-2D. *British Journal of Environment and Climate Change*, 5(2), 162-175.
- [108]. Tung T. Vu, Alvin W.Z. Chew and Adrian W.K. Law (2017). "UPC Architecture for High Performance Computational Hydrodynamics". Submitted to *Computational Physics*.
- [109]. Tung T. Vu, Adrian W.K. Law and Tien H. Nguyen (2017). "Computational Hydraulic Modelling with UPC Architecture for Large Scale Simulation". Submitted to *Environmental Modelling & Software*.
- [110]. Vacondio, R., Dal Palù, A., and Mignosa, P. (2014). GPU-enhanced finite volume shallow water solver for fast flood simulations. *Environmental Modelling & Software*, 57, 60-75.
- [111]. Van Albada, G. D., van Leer, B., and Roberts, W. W. (1997). A Comparative Study of Computational Methods in Cosmic Gas Dynamics. In M. Y. Hussaini, B. van Leer, and J. Van Rosendale (Eds.), *Upwind and High-Resolution Schemes* (95-103).
- [112]. Villanueva, I., and Wright, N. G. (2006). Linking Riemann and storage cell models for flood prediction. *Proceedings of the Institution of Civil Engineers: Water Management*, 159(1), 27-33.
- [113]. Wellein, G., Zeiser, T., Hager, G., and Donath, S. (2006). On the single processor performance of simple lattice Boltzmann kernels. *Computers & Fluids*, 35(8), 910-919.
- [114]. White, F. M. (1991), *Viscous Fluid Flow*, 2nd Ed., McGraw-Hill, 7, 457-528.

- [115]. Yelick, K., Bonachea, D., Chen, W.-Y., Colella, P., Datta, K., Duell, J., and Wen, T. (2007). Productivity and performance using partitioned global address space languages. Paper presented at the Proceedings of the 2007 international workshop on Parallel symbolic computation, London, Ontario, Canada.
- [116]. Yeomans, J. M. (2006). Mesoscale simulations: Lattice Boltzmann and particle algorithms. *Physica A: Statistical Mechanics and its Applications*, 369(1), 159-184.
- [117]. Yilmaz, E., Payli, R. U., Akay, H. U., and Ecer, A. (2009), "Hybrid Parallelism for CFD Simulations: Combining MPI with OpenMP," in *Parallel Computational Fluid Dynamics 2007: Implementations and Experiences on Large Scale and Grid Computing*, ed Berlin, Heidelberg: Springer Berlin Heidelberg, 401-408.
- [118]. Yu, D. (2010). Parallelization of a two-dimensional flood inundation model based on domain decomposition. *Environmental Modelling & Software*, 25(8), 935-945.

Appendix A

The predicted flux at time step $n+1/2$ is calculated as follow:

$$h_{j,k}^{n+\frac{1}{2}} = h_{j,k}^n - \frac{\Delta t}{2} [u_{\xi} \overline{\Delta_{\xi} h} + u_{\eta} \overline{\Delta_{\eta} h} + h(\xi_x \overline{\Delta_{\xi} u} + \xi_y \overline{\Delta_{\eta} u} + \eta_x \overline{\Delta_{\xi} v} + \eta_y \overline{\Delta_{\eta} v})]_{j,k}^n \quad (\text{A.1})$$

$$u_{j,k}^{n+\frac{1}{2}} = u_{j,k}^n - \frac{\Delta t}{2} \left[u_{\xi} \overline{\Delta_{\xi} u} + u_{\eta} \overline{\Delta_{\eta} u} + g(\xi_x \overline{\Delta_{\xi} \zeta} + \eta_x \overline{\Delta_{\eta} \zeta}) + c_D u \sqrt{\frac{u^2 + v^2}{h}} \right]_{j,k}^n \quad (\text{A.2})$$

$$v_{j,k}^{n+1/2} = v_{j,k}^n - \frac{\Delta t}{2} [u_{\xi} \overline{\Delta_{\xi} v} + u_{\eta} \overline{\Delta_{\eta} v} + g(\xi_x \overline{\Delta_{\xi} \zeta} + \eta_y \overline{\Delta_{\eta} \zeta}) + c_D v \sqrt{\frac{u^2 + v^2}{h}}]_{j,k}^n \quad (\text{A.3})$$

where ξ and η are in the directions of contiguous j and k indices, respectively; the terms ξ_x , ξ_y , η_x and η_y are the grid transformation metrics; Δt is the time-step; ζ is the free-surface elevation; $u_{\xi} = u_{\xi_x} + u_{\xi_y}$ and $u_{\eta} = u_{\eta_x} + u_{\eta_y}$; finally $\bar{\Delta}$ denotes the cell-average gradient of the variables.

To achieve second-order spatial accuracy, the monotone upstream scheme for conservation (MUSCL) is adopted to linearly reconstruct the value of h , u and v as:

$$\begin{aligned} \frac{U_{j,k}^{n+1} - U_{j,k}^n}{\Delta t} + \frac{1}{S_{j,k}} [(F_{\perp}^{n+\frac{1}{2}} s)_{j+\frac{1}{2},k} - (F_{\perp}^{n+\frac{1}{2}} s)_{j-\frac{1}{2},k} + (F_{\perp}^{n+\frac{1}{2}} s)_{j,k+\frac{1}{2}} \\ - (F_{\perp}^{n+\frac{1}{2}} s)_{j,k-\frac{1}{2}}] = S_{j,k}^{n+\frac{1}{2}} \end{aligned} \quad (\text{A.4})$$

where U and S are cell-average values; S is the area of the cell; s is cell face length; finally $F_{\perp}$ is an average value on the cell face defined as:

$$F_{\perp} = \begin{pmatrix} hu_{\perp} \\ hu u_{\perp} + \frac{1}{2} gh^2 \cos \theta \\ hv u_{\perp} + \frac{1}{2} gh^2 \sin \theta \end{pmatrix} \quad (\text{A.5})$$

with $u_{\perp}$ is velocity perpendicular to the cell face and θ is the angle between the face normal vector and the x-axis.

To evaluate the fluxes, we apply the Godunov-type upwind scheme proposed by Roe (1981) as:

$$F_{\perp}^{n+\frac{1}{2}} = \frac{1}{2} (F_{\perp}^L + F_{\perp}^R - \hat{R} |\hat{\Lambda}| \Delta \hat{V}) \quad (\text{A.6})$$

This evaluation uses the calculated results from the MUSCL reconstruction step for the left and right flux ($F_{\perp}^L$ and $F_{\perp}^R$) at the cell face. For the diagonal matrix $|\hat{\Lambda}|$, the columns of $\hat{R}$ contain the corresponding right eigenvectors, and the characteristic variables $\Delta \hat{V}$ are computed as:

$$|\hat{\Lambda}| = \begin{pmatrix} |\hat{u}_{\perp} - \hat{c}| & 0 & 0 \\ 0 & |\hat{u}_{\perp}| & 0 \\ 0 & 0 & |\hat{u}_{\perp} + \hat{c}| \end{pmatrix} \quad (\text{A.7})$$

$$\hat{R} = \begin{pmatrix} 1 & 0 & 1 \\ \hat{u}_{\perp} - \hat{c} \cos \theta & -\sin \theta & \hat{u}_{\perp} + \hat{c} \cos \theta \\ \hat{u}_{\perp} - \hat{c} \sin \theta & \cos \theta & \hat{u}_{\perp} + \hat{c} \sin \theta \end{pmatrix} \quad (\text{A.8})$$

$$\Delta\hat{V} = \begin{pmatrix} \frac{1}{2}(\Delta h - \frac{\hat{h}\Delta u_{\perp}}{\hat{c}}) \\ \hat{h}\Delta u_{\parallel} \\ \frac{1}{2}(\Delta h + \frac{\hat{h}\Delta u_{\perp}}{\hat{c}}) \end{pmatrix} \quad (\text{A.9})$$

where $\hat{c}$ is the wave celerity and $u_{\parallel}$ is the velocity parallel to the cell face and Δ is the finite difference across the cell face.

Appendix B

Considering a representative control volume of a single node in Figure 1, Equation 2 is discretized over the control volume (Kermani and Plett, 2001) as shown in Equation A.1. All other nodes within the numerical domain undergo the same discretization procedure.

$$\frac{\partial \bar{Q}}{\partial t} + \frac{F_E - F_W}{\Delta x} + \frac{G_N - G_S}{\Delta y} = \frac{G_{Vx,E} - G_{Vx,W}}{\Delta x} + \frac{G_{Vy,N} - G_{Vy,S}}{\Delta y} \quad (B.1)$$

where $\partial \bar{Q}$ is the discretized form of Q which is defined as either $(Q^{n+\frac{1}{2}} - Q^n)$ for the predictor step, or $(Q^{n+1} - Q^n)$ for the corrector step.

The temporal term in Equation B.1 undergoes the 2-step explicit scheme from the Lax-Wendroff family of predictors-correctors. The predictor step computes the flow condition at the half-time step as shown in Equation B.2, whereas the corrector step follows by central differencing with time at the full-time step in Equation B.3. For the respective computations at the half- and full time step, the convective and viscous fluxes are first computed before computing for $Q^{n+1/2}$ and Q^{n+1} .

$$\begin{aligned} \frac{Q^{n+1/2} - Q^n}{\Delta t/2} + \left(\frac{F_E - F_W}{\Delta x}\right)^n + \left(\frac{G_N - G_S}{\Delta y}\right)^n \\ = \left(\frac{G_{Vx,E} - G_{Vx,W}}{\Delta x}\right)^n + \left(\frac{G_{Vy,N} - G_{Vy,S}}{\Delta y}\right)^n \end{aligned} \quad (B.2)$$

$$\begin{aligned}
& \frac{Q^{n+1} - Q^n}{\Delta t} + \left(\frac{F_E - F_W}{\Delta x} \right)^{n+\frac{1}{2}} + \left(\frac{G_N - G_S}{\Delta y} \right)^{n+\frac{1}{2}} \\
& = \left(\frac{G_{Vx,E} - G_{Vx,W}}{\Delta x} \right)^{n+\frac{1}{2}} + \left(\frac{G_{Vy,N} - G_{Vy,S}}{\Delta y} \right)^{n+\frac{1}{2}}
\end{aligned} \tag{B.3}$$

The convective fluxes (F and G) in Equation 4 are computed by the Roe scheme which is represented by Equation B.4 and B.5 respectively (Kermani and Plett, 2001 and Toro, 2006). The computed F_E at node (j, k) equates to F_W of node (j+1, k), and the computed G_N at node (j, k) equates to G_S of node (j, k-1).

$$F_E = 0.5[F_E^L + F_E^R] - 0.5 \sum_{k=1}^4 |\hat{\lambda}_E^k| \delta w_E^{(k)} \hat{T}_E^k \tag{B.4}$$

$$G_N = 0.5[G_N^L + G_N^R] - 0.5 \sum_{k=1}^4 |\hat{\lambda}_N^k| \delta w_N^{(k)} \hat{T}_N^k \tag{B.5}$$

where F_E^L and F_E^R are the inner and outer values of F computed at the east face, G_N^L and G_N^R are the inner and outer values of G computed at the north face, $\hat{\lambda}_E^k$ and $\hat{\lambda}_N^k$ are the respective eigenvalues of the Jacobian matrix determined at the Roe's averaged condition, $\hat{T}_E^k$ and $\hat{T}_N^k$ are the respective eigenvectors corresponding to the determined eigenvalues, and both $\delta w_E^{(k)}$ and $\delta w_N^{(k)}$ are the respective wave amplitudes.

To compute the eigenvalues, corresponding eigenvectors and wave amplitudes of both F and G, similar computational procedures are undertaken. Here, only the computations for the required parameters of F are shown for brevity. For further details, the reader is referred to references (Kermani and Plett, 2001 and Toro, 2006).

For each of the nodes within the numerical domain, the left (L) and right (R) flow conditions are first computed by Equation B.6 and B.7 respectively based on the 3rd order upwind-biased algorithm (Kermani and Plett, 2001).

$$q_E^L = q_{j,k} + 0.25 \left[\left(\frac{2}{3} \right) (q_{j,k} - q_{j-1,k}) + \left(\frac{4}{3} \right) (q_{j+1,k} - q_{j,k}) \right] \quad (B.6)$$

$$q_E^R = q_{j+1,k} - 0.25 \left[\left(\frac{2}{3} \right) (q_{j+2,k} - q_{j+1,k}) + \left(\frac{4}{3} \right) (q_{j+1,k} - q_{j,k}) \right] \quad (B.7)$$

where q represents the primitive variables (u, v).

Following which, the Roe's averaged conditions are determined in Equations B.8 to 11 (Kermani and Plett, 2001 and Toro, 2006). Since the fluid is incompressible, no averaged condition is computed for the density (ρ) parameter.

$$\widehat{u}_E = \frac{\sqrt{\rho_W} u_E^R + \sqrt{\rho_W} u_E^L}{2\sqrt{\rho_W}} \quad (B.8)$$

$$\widehat{v}_E = \frac{\sqrt{\rho_W} v_E^R + \sqrt{\rho_W} v_E^L}{2\sqrt{\rho_W}} \quad (B.9)$$

$$\widehat{H}_E = \frac{\sqrt{\rho_W} H_E^R + \sqrt{\rho_W} H_E^L}{2\sqrt{\rho_W}} \quad (B.10)$$

$$\widehat{c}_E = \sqrt{(\gamma - 1)(\widehat{H}_E - 0.5(\widehat{u}_E^2 + \widehat{v}_E^2))} \quad (B.11)$$

where $\widehat{u}_E$, $\widehat{v}_E$, $\widehat{H}_E$ and $\widehat{c}_E$ are the Roe averaged conditions, $\gamma = 1.33$ for the ratio of specific heats applied to water and $\rho_W = 1000 \text{kgm}^{-3}$ for the density of water.

With the computed Roe's averaged conditions from Equations B.8 to B.11, the values of $\widehat{\lambda}_E^k$, $\widehat{T}_E^k$ and $\delta w_E^{(k)}$ are then computed in the following (Kermani and Plett, 2001).

$$\begin{bmatrix} \widehat{\lambda}_E^1 \\ \widehat{\lambda}_E^2 \\ \widehat{\lambda}_E^3 \\ \widehat{\lambda}_E^4 \end{bmatrix} = \begin{bmatrix} \widehat{u}_{\perp E} - \widehat{c}_E \\ \widehat{u}_{\perp E} \\ \widehat{u}_{\perp E} \\ \widehat{u}_{\perp E} + \widehat{c}_E \end{bmatrix} \quad (B.12)$$

$$\begin{bmatrix} \delta w_E^{(1)} \\ \delta w_E^{(2)} \\ \delta w_E^{(3)} \\ \delta w_E^{(4)} \end{bmatrix} = \begin{bmatrix} \frac{\delta_{PE} - \rho_W \delta_{u \perp E} \hat{c}_E}{2 \hat{c}_E^2} \\ \rho_W \delta u_{||E} \\ -\frac{\delta_{PE} - \hat{c}_E^2 \delta_{\rho E}}{\hat{c}_E^2} \\ \frac{\delta_{PE} + \rho_W \delta_{u \perp E} \hat{c}_E}{2 \hat{c}_E^2} \end{bmatrix} \quad (B.13)$$

$$\begin{aligned} T_E^{(1)} &= \begin{bmatrix} 1 \\ \hat{u}_E - \hat{c}_E \cos \theta_E \\ \hat{v}_E - \hat{c}_E \sin \theta_E \\ \hat{H}_E - \hat{u}_{\perp E} \hat{c}_E \end{bmatrix}, T_E^{(2)} = \begin{bmatrix} 0 \\ -\sin \theta_E \\ \cos \theta_E \\ u_{||E} \end{bmatrix} \\ T_E^{(3)} &= \begin{bmatrix} 1 \\ \hat{u}_E \\ \hat{v}_E \\ \frac{\hat{u}_E^2 + \hat{v}_E^2}{2} \end{bmatrix}, T_E^{(4)} = \begin{bmatrix} 1 \\ \hat{u}_E + \hat{c}_E \cos \theta_E \\ \hat{v}_E + \hat{c}_E \sin \theta_E \\ \hat{H}_E + \hat{u}_{\perp E} \hat{c}_E \end{bmatrix} \end{aligned} \quad (B.14)$$

where $\delta_{PE} = p_{j+1,k} - p_{j,k}$, $\delta u_{||E} = u_{||j+1,k} - u_{||j,k}$, $\delta_{\rho E} = \rho_{j+1,k} - \rho_{j,k}$, $\delta_{u \perp E} = u_{\perp j,k+1} - u_{\perp j,k}$ and $u_{||E}^2 = \vec{V}_E \cdot \vec{V}_E - u_{\perp E}^2$.

Lastly, the viscous terms (G_{Vx} and G_{Vy}) in Equation 3 can be discretized using the 2nd order central differencing scheme (Kermani and Plett, 2001) as shown.

$$\frac{G_{Vx,E} - G_{Vx,W}}{\Delta x} = \frac{G_{(i+1,j)} - 2G_{(i,j)} + G_{(i-1,j)}}{(\Delta x)^2} \quad (B.15)$$

$$\frac{G_{Vy,N} - G_{Vy,S}}{\Delta y} = \frac{G_{(i,j+1)} - 2G_{(i,j)} + G_{(i,j-1)}}{(\Delta y)^2} \quad (B.16)$$

As discussed previously, the implemented numerical scheme in UPC-CHD was validated by examining three CHD flow cases. The boundary and initial conditions adopted for Cases A, B and C are represented in Figure 4.2a, 4.2b and 4.2c respectively. The temperature of all numerical domains was kept at 293.15K.