UNIVERSITY OF SOUTHERN QUEENSLAND

PARALLEL COMPUTATIONS BASED ON DOMAIN DECOMPOSITIONS AND INTEGRATED RADIAL BASIS FUNCTIONS FOR FLUID FLOW PROBLEMS

A thesis submitted by

NAM PHAM-SY

B.Eng. (Hon.), M.Sc. (Hon.)

For the award of the degree of

Doctor of Philosophy

2015

Dedication

To my family

Abstract

The thesis reports a contribution to the development of parallel algorithms based on Domain Decomposition (DD) method and Compact Local Integrated Radial Basis Function (CLIRBF) method. This development aims to solve large scale fluid flow problems more efficiently by using parallel high performance computing (HPC). With the help of the DD method, one big problem can be separated into sub-problems and solved on parallel machines. In terms of numerical analysis, for each sub-problem, the overall condition number of the system matrix is significantly reduced. This is one of the main reasons for the stability, high accuracy and efficiency of parallel algorithms. The developed methods have been successfully applied to solve several benchmark problems with both rectangular and non-rectangular boundaries.

In parallel computation, there is a challenge called Distributed Termination Detection (DTD) problem. DTD concerns the discovery whether all processes in a distributed system have finished their job. In a distributed system, this problem is not a trivial problem because there is neither a global synchronised clock nor a shared memory. Taking into account the specific requirement of parallel algorithms, a new algorithm is proposed and called the Bitmap DTD. This algorithm is designed to work with DD method for solving Partial Differential Equations (PDEs). The Bitmap DTD algorithm is inspired by the Credit/Recovery DTD class (or weight-throw). The distinguishing feature of this algorithm is the use of a bitmap to carry the snapshot of the system from process to process. The proposed algorithm possesses characteristics as follows. (i) It allows any process to detect termination (symmetry); (ii) it does not require any central control agent (decentralisation); (iii) termination detection delay is of the order of the diameter of the network; and (iv) the message complexity of the proposed algorithm is optimal.

In the first attempt, the combination of the DD method and CLIRBF based collocation approach yields an effective parallel algorithm to solve PDEs. This approach has enabled not only the problem to be solved separately in each subdomain by a Central Processing Unit (CPU) but also compact local stencils to be independently treated. The present algorithm has achieved high throughput in solving large scale problems. The procedure is illustrated by several numerical examples including the benchmark lid-driven cavity flow problem.

A new parallel algorithm is developed using the Control Volume Method (CVM) for the solution of PDEs. The goal is to develop an efficient parallel algorithm

especially for problems with non-rectangular domains. When combined with CLIRBF approach, the resultant method can produce high-order accuracy and economical solution for problems with complex boundary. The algorithm is verified by solving two benchmark problems including the square lid-driven cavity flow and the triangular lid-driven cavity flow. In both cases, the accuracy is in great agreement with benchmark values. In terms of efficiency, the results show that the method has a very high efficiency profile and for some specific cases a super-linear speed-up is achieved.

Although overlapping method yields a straightforward implementation and stable convergence, overlapping of sub-domains makes it less applicable for complex domains. The method even generates more computing overhead for each sub-domain as the overlapping area grows. Hence, a parallel algorithm based on non-overlapping DD and CLIRBF has been developed for solving Navier-Stokes equations where a CLIRBF scheme is used to solve the problem in each sub-domain. A relaxation factor is employed for the transmission conditions at the interface of sub-domains to ensure the convergence of the iterative method while the Bitmap DTD algorithm is used to achieve the global termination. The parallel algorithm is demonstrated through two fluid flow problems, namely the natural convection in concentric annuli (Boussinesq fluids) and the lid-driven cavity flow (viscous fluids). The results confirm the high efficiency of the present method in comparison with a sequential algorithm. A super-linear efficiency is also observed for a range of numbers of CPUs.

Finally, when comparing the overlapping and non-overlapping parallel algorithms, it is found that the non-overlapping one is less stable. The numerical results show that the non-overlapping method is not able to converge for high Reynolds number while overlapping method reaches the same convergence profile as the sequential CLIRBF method. Thus, in this research when dealing with non-Newtonian fluids and large scale problems, the overlapping method is preferred to the non-overlapping one. The flow of Oldroyd-B fluid through a planar contraction was considered as a benchmark problem. In this problem, the singularity of stress at the re-entrant corners always poses difficulty to numerical methods in obtaining stable solutions at high Weissenberg numbers. In this work, a high resolution simulation of the flow is obtained and the contour of streamline is shown to be in great agreement with other results.

Certification of Thesis

I certify that the idea, experimental work, results and analyses, software and conclusions reported in this thesis are entirely my own effort, except where otherwise acknowledged. I also certify that the work is original and has not been previously submitted for any other award.

Nam PHAM-SY	Date
ENDORSEMENT	
Dr. Canh–Dung TRAN, Principal supervisor	Date
Prof. Nam MAI–DUY, Co-supervisor	Date
Prof. Thanh TRAN–CONG, Co-supervisor	Date

Acknowledgements

I would like to express my deepest gratitude to my supervisors Dr. Canh-Dung Tran, Professor Nam Mai-Duy and Professor Thanh Tran-Cong. Without their help this thesis would not have been possible.

My candidature was financially supported by a postgraduate scholarship from the University of Southern Queensland (USQ) and partly supported by the Computational Engineering and Science Research Centre. This support is gratefully acknowledged.

My family including my parents in law has always been a great source of inspiration and encouragement during the candidature. I have been unconditionally loved and supported by my wife Thuy-Tram. I am grateful that she has always been by my side through this challenging episode of my life.

I would also like to thank my friends and colleagues with whom I have had great discussions and shared great empathy and encouragement.

I would like to express my gratitude to the staff of the Faculty of Health, Engineering and Sciences for their administrative support; Mr. Richard Young for his technical support regarding the use of USQ High Performance Computing (HPC) system; Dr. Francis Gacenga for his support in accessing external HPC resources.

The computation in this thesis was performed mainly on the USQ HPC cluster and partly on the University of Queensland and Queensland University of Technology HPC cluster under an agreement of Queensland Cyber Infrastructure Foundation (QCIF).

Publications resulting from the Thesis

Journal papers

1. **Pham-Sy, N.**, Tran, C.-D., Hoang-Trieu, T.-T., Mai-Duy, N., and Tran-Cong, T. (2013). Compact local IRBF and domain decomposition method for solving PDEs using a distributed termination detection based parallel algorithm. *CMES Computer Modeling in Engineering and Sciences*, 92(1):1-31.

2. **Pham-Sy, N.**, Tran, C.-D., Mai-Duy, N., and Tran-Cong, T. (2014). Parallel Control-volume Method Based on Compact Local Integrated RBFs for the Solution of Fluid Flow Problems. *CMES Computer Modeling in Engineering and Sciences*, 100(5):363-397.

3. Nguyen, H., Tran, C.-D., **Pham-Sy, N.**, and Tran-Cong, T. (2014). A Numerical Solution Based on the Fokker-Planck Equation for Dilute Polymer Solutions Using High Order RBF Methods. *Applied Mechanics and Materials*, 553: 187-192.

4. Tien, C.M.T, **Pham-Sy, N.**, Tran, C.-D., Mai-Duy, N, Tran-Cong, T. (2015) A high-order coupled compact integrated RBF approximation based domain decomposition algorithm for 2nd-order differential problems, *CMES Computer Modeling in Engineering and Sciences*, 104(4):251-304.

5. **Pham-Sy, N.**, Tran, C.-D., Mai-Duy, N., and Tran-Cong, T. (2015). A symmetric Bitmap Distributed Termination Detection approach for domain decomposition based parallel algorithms. *Journal of Parallel and Distributed Computing* (Submitted).

6. **Pham-Sy, N.**, Tran, C.-D., Mai-Duy, N., and Tran-Cong, T. (2015). A parallel non-overlapping domain decomposition algorithm using integrated radial basis function for solving Navier-Stoke equations. *CMES Computer Modeling in Engineering and Sciences* (Submitted).

7. **Pham-Sy, N.**, Tran, C.-D., Mai-Duy, N., and Tran-Cong, T. (2015). Parallel IRBF algorithm for the solution of solving large scale problems in fluid mechanics. *CMES Computer Modeling in Engineering and Sciences* (To be submitted).

Conference papers

1. **Pham-Sy, N.**, Tran, C.-D., Hoang-Trieu, T.-T., Mai-Duy, N., and Tran-Cong, T. (2012). Development of Parallel algorithm for Boundary Value Problems using Compact Local Integrated RBFN and Domain Decomposition. In Y.T. Gu and S. C. Saha (Eds). *4th International Conference on Computational Methods (ICCM2012)*, Gold Coast, Australia, 25-28 November 2012, CD Proceedings, Queensland University of Technology (ISBN: 978-1-921897-54-2), page 94.

Table of Contents

Acronyms and Abbreviations

AB	Artificial Boundary
ALU	Algebraic Unit
BEM	Boundary Element Method
BVP	Boundary Value Problem
CFD	Computational Fluid Dynamics
CLIRBF	Compact Local Integrated Radial Basis Function
CM	Convergence Measure
CPU	Central Processing Unit
CV	Control Volume
DD	Domain Decomposition
DoF	Degrees of Freedom
DRBF	Differential Radial Basis Function
DTD	Distributed Termination Detection
FDM	Finite Difference Method
FEM	Finite Element Method
FPR	Floating Point Register
FPU	Floating Point Unit
FVM	Finite Volume Method
GPGPU	General-Purpose Computing on Graphics Processing Unit
GPR	General Purpose Register
GPU	Graphics Processing Unit
HPC	High Performance Computing
IRBF	Integrated Radial Basis Function

LDC	Lid Driven Cavity
LSD	List of Sub-domains
MIMD	Multiple Instruction and Multiple Data
MISD	Multiple Instruction and Single Data
MLPG	Meshless Local Petrov-Galerkin
MPP	Massively Parallel Processor
MQ	Multiquadric
NIC	Network Interface Card
NS	Neighbour Sub-domain
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
SIMD	Single Instruction and Multiple Data
SPMD	Single Program and Multiple Data
STE	Step To End
VLIN	Very Long Instruction Word

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, an introduction to the thesis is presented. Firstly, the motivation, significance and objectives of the thesis are discussed. Secondly, the fundamental equations that govern the motion of Newtonian and non-Newtonian fluids, are presented, followed by a review of numerical methods for fluid flow analysis. Lastly, an outline of the thesis is given at the end of this chapter.

1.1 Motivation, significance and objectives

Treatment of realistic computational fluid dynamics (CFD) models in engineering and science problems such as weather prediction, flooding simulation or industrial design requires a large number of degrees of freedom (DoF). It is desirable to have robust and efficient numerical methods to solve these problems within a reasonable time frame. However, serial numerical methods are limited in practice by computer resources such as the speed of processor and memory. One of the most promising solution to this resource limitation is parallelisation. With the advance of high performance computing (HPC), intensive computation tasks may be distributed to many parallel CPUs. Each CPU can communicate with other CPUs to exchange data during the computation. There has been a considerable amount of work focused on developing parallel algorithms for solving CFD problems, for example, Tezduyar et al. (1994), Johan et al. (1992), Hou and Wu (1997) with Finite Element method; Balls and Colella (2002) with Finite Difference method; Marshall et al. (1997), Bui (2000) with Finite Volume method; Shirazaki and Yagawa (1999) with Free-Mesh method; Ingber et al. (2004), Singh and Jain (2005) with mesh-free approach using Radial Basis Function (RBF) method or element-free Galerkin method; Tran et al. (2009) with macro-micro approach using Brownian Configuration Field - RBF. High efficiency is reported in (Singh and Jain, 2005; Tran et al., 2009). However, most of parallel methods focused heavily on solving large-scale problems by overcoming the memory limitation on a single computer without much attention to the question of efficiency. As a result, efficiency of these methods is usually low or disregarded. So the motivation of the present work is to develop parallel numerical methods with high efficiency to solve complex CFD problems. The methods presented in

this thesis are based on Domain Decomposition methods (DDM) and Integrated Radial Basis Function (IRBF) method, achieving the following advantages: (i) ease of load balancing among CPUs; (ii) low message complexity thanks to the fact that each CPU communicates with only four neighbouring CPUs; and (iii) high scalability associated with a distributed memory parallel computing model.

In a distributed system, if a process terminates arbitrarily, other processes will not be able to exchange data with it and the whole system might hang up forever. Hence, every process has to be aware of the status of all other processes so that it can terminate properly. This is actually an important research topic in distributed computing called Distributed Termination Detection (DTD) (Dijkstra and Scholten, 1980). Although there is a number of DTD algorithms available in the literature, it is necessary to develop a new DTD algorithm that can be incorporated efficiently into parallel methods based on DD and IRBF. The presently proposed DTD algorithm has four main advantages including symmetric detection mechanism, decentralised control, low message complexity and optimal termination detection delay.

The developed parallel methods and DTD algorithm can be applied to solve various CFD problems. This will give researchers a powerful and flexible tool to deal with problems that are currently difficult to investigate because of the limitation of computing resource. Furthermore, the DTD algorithm has a broader range of applications as it can be used in many distributed systems other than CFD without any major alteration.

In summary, main objectives of the thesis includes

1. Develop a DTD algorithm for an efficient and proper termination of parallel methods;
2. Develop parallel methods based on DD method and IRBF method to solve CFD problems;
3. Apply the developed methods to solve various CFD problems, including flows of Newtonian and non-Newtonian fluids in both rectangular and non-rectangular domains.

1.2 Fluid dynamics

Everyday life abounds with examples of fluid motion, ranging from blood flow in human bodies, to industrial flows such as flow of polymers and other chemicals in pipes, to environmental flows such as underground water, oceanic and atmospheric flows. Fluid dynamics is a branch of applied science that helps understand fluid flow phenomena, particularly through mathematical modelling where the fluid flow behaviour is governed by a set of fundamental partial differential equations (PDEs), the constitutive equation (or equivalent description of material property) and the boundary conditions. The fundamental governing equations are derived from three conservation laws, i.e., the conservation of

mass, the conservation of momentum and the conservation of energy. For an incompressible fluid, the conservation of mass is expressed by

$$\nabla \cdot \mathbf{v} = 0, \quad \mathbf{x} \in \Omega, \tag{1.1}$$

where $\mathbf{v}$ is the velocity vector, $\mathbf{x}$ the position vector, and Ω the domain of interest.

The conservation of momentum is described as

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g}, \qquad \mathbf{x} \in \Omega, \tag{1.2}$$

where t is the time, ρ the density, $\boldsymbol{\sigma}$ the total stress tensor and $\mathbf{g}$ the force per unit mass due to gravity (Tanner, 2000; Reddy and Gartling, 1994).

The total stress tensor is given by $\boldsymbol{\sigma} = -p\mathbf{I} + \boldsymbol{\tau}$, where p is the hydrostatic pressure, $\mathbf{I}$ the identity tensor and $\boldsymbol{\tau}$ the extra stress tensor. Eq. (1.2) can now be rewritten as

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{g}, \qquad \mathbf{x} \in \Omega. \tag{1.3}$$

The pressure term and gravity term can be combined into one term called modified pressure, $\nabla P = \nabla p - \rho \mathbf{g}$, and Eq. (1.3) becomes

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla P + \nabla \cdot \boldsymbol{\tau}, \qquad \mathbf{x} \in \Omega. \tag{1.4}$$

The mechanical behaviour of a fluid can be described by a constitutive equation. For Newtonian fluids, the constitutive equation has the form of

$$\boldsymbol{\tau} = 2\eta_0 \mathbf{D}, \tag{1.5}$$

where η_0 is the constant viscosity, and $\mathbf{D}$ the rate of deformation tensor, defined as $\mathbf{D} = \frac{1}{2}\left(\nabla \mathbf{v}^T + \nabla \mathbf{v}\right)$.

For non-Newtonian fluids, many different constitutive models have been developed. Some of them are given as follows.

Upper-Convected Maxwell (UCM) model

$$\boldsymbol{\tau} + \lambda_1 \overset{\nabla}{\boldsymbol{\tau}} = 2\eta_0 \mathbf{D}, \tag{1.6}$$

where λ_1 is the characteristic relaxation time of the fluid and the upper-convected derivative $\overset{\nabla}{[]}$ is defined as

$$\overset{\nabla}{[]} = \frac{\partial []}{\partial t} + \mathbf{v} \cdot \nabla [] - (\nabla \mathbf{v})^T \cdot [] - [] \cdot \nabla \mathbf{v}. \tag{1.7}$$

Oldroyd-B model

$$\boldsymbol{\tau} + \lambda_1 \overset{\nabla}{\boldsymbol{\tau}} = 2\eta_0 \left(\mathbf{D} + \lambda_2 \overset{\nabla}{\mathbf{D}} \right), \tag{1.8}$$

where λ_2 is the characteristic retardation time of the fluid. Let α be the ratio of the retardation time to the relaxation time ($\alpha = \lambda_2/\lambda_1$). When $\alpha = 0$ the Oldroyd-B model yields the UCM one.

The extra stress tensor $\boldsymbol{\tau}$ consists of two components, the solvent and polymeric contributions.

$$\boldsymbol{\tau} = 2\eta_s \mathbf{D} + \boldsymbol{\tau}_v, \tag{1.9}$$

where η_s is the solvent viscosity and $\boldsymbol{\tau}_v$ the elastic stress, which is given by

$$\boldsymbol{\tau}_v + \lambda_1 \overset{\nabla}{\boldsymbol{\tau}}_v = 2\eta_p \mathbf{D}, \tag{1.10}$$

where η_p is the polymeric viscosity, $\eta_0 = \eta_s + \eta_p$, $\eta_s = \alpha\eta_0$, and $\eta_p = (1-\alpha)\eta_0$.

As found in Eq. (1.9) if the value of η_s is equal to zero, then $\boldsymbol{\tau} = \boldsymbol{\tau}_v$, the Oldroyd-B model becomes UCM one (Covas et al., 1995; Phan-Thien and Tanner, 1977).

Giesekus-Leonov model

$$\boldsymbol{\tau}_v + \lambda_1 \overset{\nabla}{\boldsymbol{\tau}}_v - \frac{\lambda_1}{2\eta_p}\boldsymbol{\tau}_v\boldsymbol{\tau}_v = 2\eta_p \mathbf{D}. \tag{1.11}$$

Phan-Thien Tanner (PTT) model 1

$$\exp\left(\frac{\lambda_1 \varepsilon}{\eta_p}\mathrm{tr}(\boldsymbol{\tau}_v)\right)\boldsymbol{\tau}_v + \lambda_1 \overset{\nabla}{\boldsymbol{\tau}}_v + \xi\lambda_1 (\mathbf{D}\boldsymbol{\tau}_v + \boldsymbol{\tau}_v\mathbf{D}) = 2\eta_p \mathbf{D}, \tag{1.12}$$

where ε and ξ are the material parameters, and 'tr' denotes the trace operation.

Phan-Thien Tanner (PTT) model 2

$$\left(1 + \frac{\lambda_1 \varepsilon}{\eta_p}\mathrm{tr}(\boldsymbol{\tau}_v)\right)\boldsymbol{\tau}_v + \lambda_1 \overset{\nabla}{\boldsymbol{\tau}}_v + \xi\lambda_1 (\mathbf{D}\boldsymbol{\tau}_v + \boldsymbol{\tau}_v\mathbf{D}) = 2\eta_p \mathbf{D}, \tag{1.13}$$

where the coefficients are defined as before. For more details about PTT models, please refer to (Phan-Thien, 1978, 1984; Tanner, 2000).

1.3 Numerical methods

In practical problems, the above governing equations can rarely be solved analytically. Instead, their solution is normally sought in the form of approximation given by numerical methods. Common numerical methods are presented briefly below.

1.3.1 Finite Difference method (FDM)

Finite difference method is believed to be the earliest numerical method and can be simply derived from the Taylor expansion of a function at a set of collocation

points (Smith, 1978). The derivatives of the function are approximated using the nodal values of the function. The differential equation is discretised into a system of algebraic equations whose unknowns are the nodal function values. The method is well-known for its speed thanks to the tri-diagonal form of the system matrix (Conte and Dames, 1958; Gupta and Manohar, 1979; Bjorstad, 1983). However, the method has low accuracy due to the fact that the conservation is not enforced inherently. High-order schemes have been successfully introduced to increase the accuracy of the method, such as compact FD scheme (Lele, 1992; Li et al., 1995), high-order FD upwind scheme (Ferreira et al., 2009), pseudo-spectral FDM (Pilitsis and Beris, 1989). More details can be found in Mitchell and Griffiths (1980) and Crochet et al. (1984), for example.

1.3.2 Finite Volume method (FVM)

In finite volume method, the computation domain is divided into contiguous control volumes (CV). The governing equation is integrated over the whole CV. By applying Green formulae, the surface integration is transformed to line integration. With appropriate quadrature rule, the line integration is approximated in terms of nodal values, i.e., a linear equation is achieved (Patankar, 1980; Huilgol and Phan-Thien, 1997). FVM is conservative by construction (Eymard et al., 2000). In addition, the method is suitable for many types of grid. However, FVMs of order higher than second are more difficult to develop because FVM requires three levels of approximation, namely interpolation, differentiation and integration.

1.3.3 Finite Element method (FEM)

Finite element method is the favoured discretisation method in structural mechanics (Hughes, 1987b; Rannacher, 1999; Reddy, 2005). Thanks to the ability to model singularities through mesh refinement, FEMs have been used in many non-Newtonian fluid flow simulations (Yurun and Crochet, 1995; Fan et al., 1999; Sun et al., 1999). However, the matrices of the linearised equations are not well structured so FEM normally requires highly efficient solution methods (Pastor et al., 1991). In addition, generating a FE mesh is a costly process and the solution has a slow convergence in high gradient regions (Pastor et al., 1991; Emdadi et al., 2008). Galerkin FEM and mixed FEM (Baaijens, 1998) and the streamline upwind Petrov-Galerkin (SUPG) (Brooks and Hughes, 1982; Hughes, 1987a) are among the most popular schemes.

1.3.4 Boundary Element method (BEM)

Boundary element method is any method that solves PDEs by finding the approximate numerical solution to the equivalent boundary integral equation of these PDEs. In BEM, only the boundary of the domain needs to be discretised.

Hence, BEM can be more efficient than other methods in terms of computational resources for problems where the surface-to-volume ratio is small. BEM is very useful for problems where the physically relevant data are given not by the solution in the interior of the domain but rather on the boundary (Brebbia et al., 1984). The accuracy of the solution by BEM on the boundary is superior to those by FEM. However, BEM is only applicable to problems, for which a Green's function is available. Thus, problems with inhomogeneities or governed by non-linear differential equations normally are not solved straightforwardly by BEM itself but rather in combination with other methods. BEM normally produces fully populated matrices. As a result, memory and computation time required by BEM will grow as the square of the problem size (Tanner and Xue, 2002).

1.3.5 Radial Basis Function (RBF) method

Radial Basis Functions have traditionally been used to provide a continuous interpolation of scattered data sets (Franke, 1982; Haykin, 1999). Details on several well-known radial basis functions are given in Appendix A. For differential RBF (DRBF) method, the function values are first approximated by weighted RBFs. Such closed form can then be differentiated analytically to obtain derivatives of the function. On the other hand, in integrated RBF (IRBF) method the highest-order derivatives of the ODE/PDE is approximated by weighted RBFs first. Subsequently, its lower-order derivatives and function values are obtained through integration (Mai-Duy and Tran-Cong, 2001). RBF method is considered as a high order method (Kansa, 1990a). Furthermore, RBF method is able to solve problems with complex boundary. The main drawback of RBF methods is the fact that coefficient matrices are fully populated and may be ill-conditioned. To overcome these problems, local and compact schemes have been proposed (Shu et al., 2003; Lee et al., 2003; Kosec and Sarler, 2008; Bourantas et al., 2010; Mai-Duy et al., 2011). Details on RBF methods will be presented in the Chapter 2.

1.4 Outline of the Thesis

The remaining of the thesis is organised as follows.

- Chapter 2 consists of four parts describing the basic tools used in the present research project. The first part is to present the IRBF methods including compact local IRBF schemes. The second part is to review the DD method. The third part is to present the parallel computation together with the challenges to parallel performance. Finally, in the last part, the termination detection in parallel algorithm is discussed.

- Chapter 3 presents a rigorous analysis of the new DTD algorithm (Pham-Sy et al., 2015a). The so called Bitmap DTD is used to detect the termination of parallel methods implemented and reported in Chapters 4, 5, 6 and 7.

- Chapter 4 reports a parallel method achieved by combining the overlapping DD method and the CLIRBF approach. The method is verified by problems with available analytic solution and the benchmark problem of lid-driven cavity flow (Pham-Sy et al., 2013).
- Chapter 5 is to develop parallel method based on local IRBF Control Volume method and overlapping DD method. The method is verified through the simulation of the lid-driven flows in both rectangular and triangular cavities (Pham-Sy et al., 2014).
- Chapter 6 reports a parallel method based on non-overlapping DD method and compact local IRBF approach. The method is verified by solving two benchmark problems, which are the lid-driven cavity flow and the natural convection in concentric annuli (Pham-Sy et al., 2015b).
- Chapter 7 is to demonstrate the efficiency of the present parallel methods by simulating the benchmark 4:1 planar contraction flow of Newtonian and Oldroyd-B fluids.
- Chapter 8 gives some concluding remarks on the outcome of this research project and some possible research developments in the future.

Chapter 2

Fundamental background

This chapter describes several basic tools, which are the background for the numerical methods and their parallelisation developed in the present work. These tools are the Radial Basis Function (RBF) based approximation methods, the Domain Decomposition (DD) methods together with parallel programming and the Distributed Termination Detection (DTD) algorithms. In this dissertation, the integrated RBF approaches (IRBF), which are considered as numerical methods for function approximation and numerical solution of partial differential equations (PDEs), will be coupled with the DD technique in both collocation and control volume (CV) approaches. The DD techniques help methods take advantage of high performance computing (HPC) capability for the simulation of the fluid flows problems.

2.1 Radial Basis Function method

Numerical techniques have been developed to solve ordinary differential equations (ODEs) and PDEs arising from modelling physical phenomena, finance problems and weather forecast problems. Among traditional techniques presented in the previous chapter, RBF has recently emerged as a powerful tool owing to its nature as a universal approximator with computational advantages over traditional methods. Indeed, RBF based numerical methods can be implemented straightforwardly and possess a very high rate of convergence (Fedoseyev et al., 2002; Cheng et al., 2003). Furthermore, the RBF collocation methods are superior to other classical element type methods because they remove the huge burden of the mesh generating process. In 1990, Kansa (1990a,b) developed a meshless collocation method for solving PDEs using multi-quadric radial basis function (MQ-RBF). Following the Kansa's papers, several other improvements on RBF methods have been proposed (Sharan et al., 1997; Golberg et al., 1996; Fasshauer, 1999). The traditional RBF based method is also called differential RBF (DRBF) due to the fact that a function is approximated first and the function's derivatives are then calculated by differentiating the approximate expression for the function as follows.

$$u(\mathbf{x}) = \sum_{i=1}^{n} w_i g_i(\mathbf{x}), \tag{2.1}$$

$$\frac{\partial^k u(\mathbf{x})}{\partial x_j^k} = \sum_{i=1}^{n} w_i^{[x_j]} h_{[x_j]i}^{[k]}(\mathbf{x}), \tag{2.2}$$

where x_j is the j-component of $\mathbf{x}$ ($j = 1, 2$); superscript $[k]$ denotes the order of the derivatives of u ($k = 1, 2$), and $h_{[x_j]i}^{[k]}(\mathbf{x}) = \dfrac{\partial^k g_i(\mathbf{x})}{\partial x_j^k}$.

More recently, Mai-Duy and Tran-Cong (2001) proposed the idea of the integrated RBF method. Within the IRBF based methods, the highest order of the derivatives in an ODE/PDE is approximated first and, subsequently, its lower-order derivatives and the function itself are obtained through integration. This method can yield very accurate solutions using a relatively small number of grid nodes (Mai-Duy and Tran-Cong, 2003). IRBF methods can be global, local or compact local.

Consider the Poisson's equation in 2D.

$$\nabla^2 u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{2.3}$$

where u is the field variable; $\mathbf{x}$ the position vector; Ω the considered domain and f a known function of $\mathbf{x}$.

The domain of interest is discretised using a Cartesian grid. Let n_x and n_y be the numbers of grid lines in the x- and y-directions, respectively. Suppose the domain is rectangular, the total number of grid nodes will be $n = n_x \times n_y$.

2.1.1 Global IRBF

Two dimensional IRBF method (2D-IRBF)

In the global 2D-IRBF method, the dependent variable u and its derivatives are approximated using all grid nodes.

For the Poisson's equation (2.3), the highest order derivatives of the PDE, i.e. second order, are approximated by a weighted set of RBFs as

$$\frac{\partial^2 u(\mathbf{x})}{\partial x_j^2} = \sum_{i=1}^{n} w_i^{[x_j]} g_i^{[x_j]} = \sum_{i=1}^{n} w_i^{[x_j]} G_{[x_j]i}^{[2]}(\mathbf{x}), \tag{2.4}$$

where x_j is the j-component of $\mathbf{x}$ ($j = 1, 2$); $\{w_i\}_{i=1}^n$ the set of weights and $\{g_i(\mathbf{x})\}_{i=1}^n$ the set of RBFs. The superscript [.] is used to indicate the associated derivative order. The MQ-RBF is used in this work and given by

$$G_i(\mathbf{x}) = \sqrt{(\mathbf{x} - \mathbf{c}_i)^2 + a_i^2},$$

where $\{\mathbf{c}_i\}_{i=1}^n$ is a set of centres and $\{a_i\}_{i=1}^n$ a set of MQ-RBF widths.

To obtain first-order derivatives and field variable, Eq. (2.4) is integrated successively with respect to x_j as follows.

$$\frac{\partial u(\mathbf{x})}{\partial x_j} = \sum_{i=1}^{n} w_i^{[x_j]} G_{[x_j]i}^{[1]}(\mathbf{x}) + C_1^{[x_j]}(x_k), \quad k \neq j, \tag{2.5}$$

$$u^{[x_j]}(\mathbf{x}) = \sum_{i=1}^{n} w_i^{[x_j]} G_{[x_j]i}^{[0]}(\mathbf{x}) + x_j C_1^{[x_j]}(x_k) + C_2^{[x_j]}(x_k), \quad k \neq j, \tag{2.6}$$

where $G_{[x_j]i}^{[1]}(\mathbf{x}) = \int G_{[x_j]i}^{[2]}(\mathbf{x}) dx_j$, $G_{[x_j]i}^{[0]}(\mathbf{x}) = \int G_{[x_j]i}^{[1]}(\mathbf{x}) dx_j$, and $C_1^{[x_j]}(x_k)$ and $C_2^{[x_j]}(x_k)$ are constants of integration which are functions of the variable x_k, $k \neq i$.

Collocating equations (2.4) - (2.6) at grid points $\{\mathbf{x}_i\}_{i=1}^n$ yields

$$\frac{\partial^2 \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}_{x_j}^{[2]} \widetilde{\mathbf{w}}_{x_j}, \tag{2.7}$$

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}_{x_j}^{[1]} \widetilde{\mathbf{w}}_{x_j}, \tag{2.8}$$

$$\widetilde{\mathbf{u}}_{x_j} = \mathcal{G}_{x_j}^{[0]} \widetilde{\mathbf{w}}_{x_j}, \tag{2.9}$$

with

$$\begin{aligned}
\widetilde{\mathbf{w}}_{x_j} &= \left(w_1^{[x_j]}, w_2^{[x_j]}, \cdots, w_n^{[x_j]}, C_1^{[x_j]}, C_2^{[x_j]}\right)^T, \\
\widetilde{\mathbf{u}}_{x_j} &= \left(u_1^{[x_j]}, u_2^{[x_j]}, \cdots, u_n^{[x_j]}\right)^T, \\
\frac{\partial^k \widetilde{\mathbf{u}}}{\partial x_j^k} &= \left(\frac{\partial^k u_1}{\partial x_j^k}, \frac{\partial^k u_2}{\partial x_j^k}, \cdots, \frac{\partial^k u_n}{\partial x_j^k}\right)^T,
\end{aligned}$$

where $u_i^{[x_j]} = u^{[x_j]}(\mathbf{x}_i)$ $(i = 1, 2, \cdots, N)$; $\mathcal{G}^{[2]}, \mathcal{G}_{x_j}^{[1]}$ and $\mathcal{G}_{x_j}^{[0]}$ are known matrices.

One dimensional IRBF method (1D-IRBF)

One dimensional IRBF method shares the same principle as 2D-IRBF. The only difference is that, in 1D-IRBF the dependent variable u and its derivatives are approximated using nodes on a single line along the directions. 1D-IRBF method uses only n_x or n_y nodes, compare to n nodes of 2D-IRBF, to construct the approximation of a value at a given point. As a result, a much sparser approximation matrix is obtained. Sparse matrices generally have better condition number. They are also less expensive to compute than dense matrices. The approximation of a PDE now consists of (i) using IRBFs to approximate the variable u and its derivatives along a grid-line, and (ii) applying Kronecker tensor product to construct the approximation matrix for derivatives over a 2D domain.

For example, consider an x_j-gridline, which runs parallel to the x_j-axis. For simplicity, the grid nodes are numbered from left to right, started with $\mathbf{x}_1$ and ended with $\mathbf{x}_{n_j}$.

The second order derivative of the PDE at an arbitrary point in the considered line is approximated as follows.

$$\frac{\partial^2 u(\mathbf{x})}{\partial x_j^2} = \sum_{i=1}^{n_j} w_i g_i = \sum_{i=1}^{n_j} w_i G_i(\mathbf{x}). \tag{2.10}$$

The first-order derivatives and field variable are then obtained as follows.

$$\frac{\partial u}{\partial x_j} = \sum_{i=1}^{n_j} w_i G_i^{[1]}(\mathbf{x}) + C_1, \tag{2.11}$$

$$u = \sum_{i=1}^{n_j} w_i G_i^{[0]}(\mathbf{x}) + C_1 x_j + C_2. \tag{2.12}$$

Collocating equations (2.10) - (2.12) at grid points $\{\mathbf{x}_i\}_{i=1}^{n_j}$ yields

$$\frac{\partial^2 \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}^{[2]}\widetilde{\mathbf{w}}, \tag{2.13}$$

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}^{[1]}\widetilde{\mathbf{w}}, \tag{2.14}$$

$$\widetilde{\mathbf{u}} = \mathcal{G}^{[0]}\widetilde{\mathbf{w}}, \tag{2.15}$$

with

$$\begin{aligned}
\widetilde{\mathbf{w}} &= \Big(w_1, w_2, \cdots, w_{n_j}, C_1, C_2\Big)^T, \\
\widetilde{\mathbf{u}} &= \Big(u_1, u_2, \cdots, u_{n_j}\Big)^T, \\
\frac{\partial^k \widetilde{\mathbf{u}}}{\partial x_j^k} &= \left(\frac{\partial^k u_1}{\partial x_j^k}, \frac{\partial^k u_2}{\partial x_j^k}, \cdots, \frac{\partial^k u_{n_j}}{\partial x_j^k}\right)^T,
\end{aligned}$$

where $u_i = u(\mathbf{x}_i)$ $(i = 1, 2, \cdots, n_j)$; $\mathcal{G}^{[2]}, \mathcal{G}^{[1]}$ and $\mathcal{G}^{[0]}$ are known matrices of size $n_j \times (n_j + 2)$ as presented below.

$$\mathcal{G}^{[k]}(k = 1, 2, 3) = \begin{bmatrix}
G_1^{[k]}(\mathbf{x}_1) & G_2^{[k]}(\mathbf{x}_1) & \cdots & G_{n_j}^{[k]}(\mathbf{x}_1) & a_1^{[k]} & b_1^{[k]} \\
G_1^{[k]}(\mathbf{x}_2) & G_2^{[k]}(\mathbf{x}_2) & \cdots & G_{n_j}^{[2]}(\mathbf{x}_2) & a_2^{[k]} & b_2^{[k]} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
G_1^{[k]}(\mathbf{x}_{n_j}) & G_2^{[k]}(\mathbf{x}_{n_j}) & \cdots & G_{n_j}^{[k]}(\mathbf{x}_{n_j}) & a_{n_j}^{[k]} & b_{n_j}^{[k]}
\end{bmatrix},$$

where

$$\Big(\ a_1^{[k]},\ \ a_2^{[k]},\ \ \cdots,\ \ a_{n_j}^{[k]}\ \Big)^T = \begin{cases}
\Big(\ 0,\ \ 0,\ \ \cdots,\ \ 0\ \Big)^T, & k = 2 \\
\Big(\ 1,\ \ 1,\ \ \cdots,\ \ 1\ \Big)^T, & k = 1 \\
\Big(\ x_{j1},\ \ x_{j2},\ \ \cdots,\ \ x_{jn_j}\ \Big)^T, & k = 0
\end{cases}$$

and

$$\Big(\ b_1^{[k]},\ \ b_2^{[k]},\ \ \cdots,\ \ b_{n_j}^{[k]}\ \Big)^T = \begin{cases}
\Big(\ 0,\ \ 0,\ \ \cdots,\ \ 0\ \Big)^T, & k = 1, 2 \\
\Big(\ 1,\ \ 1,\ \ \cdots,\ \ 1\ \Big)^T, & k = 0
\end{cases}$$

2.1.2 Local and compact local IRBF

As stated in Kansa (1990a), although the global RBF methods are proved to have better accuracy in approximating a function, they are time-consuming and require large storage. More importantly, when the number of collocation points increases, the system matrix becomes ill-conditioned. To overcome these drawbacks, several solutions have been suggested including local RBF based methods (Mai-Duy and Tanner, 2007; Mai-Duy et al., 2011) and domain decomposition (DD) methods (Beatson et al., 2001; Tran et al., 2009). The difference between local and global IRBF methods is that in local methods, the approximation of a function at a grid node involves a small number of surrounding nodes only.

However, local IRBF methods are less accurate as the approximation is based on a set of local points rather than the whole grid. To recover the loss of accuracy, compact schemes have been introduced (Mai-Duy et al., 2011; Hoang-Trieu et al., 2012; Thai-Quang et al., 2012). In this work, the 2D-IRBF compact scheme proposed in Mai-Duy et al. (2011) is employed and presented in details.

For 2D problems, a 9-point stencil scheme is applied to overcome the problem of ill-conditioned system matrix, which is an inherent issue in the global approach. According to this scheme, a local 9-point stencil for an arbitrary grid-point $\mathbf{x}_{i,j}$ $(2 \leq i \leq n_x - 1; 2 \leq j \leq n_y - 1)$ is described as follows.

$$\begin{bmatrix} \mathbf{x}_{i-1,j+1} & \mathbf{x}_{i,j+1} & \mathbf{x}_{i+1,j+1} \\ \mathbf{x}_{i-1,j} & \mathbf{x}_{i,j} & \mathbf{x}_{i+1,j} \\ \mathbf{x}_{i-1,j-1} & \mathbf{x}_{i,j-1} & \mathbf{x}_{i+1,j+1} \end{bmatrix}.$$

For simplicity, a specific local stencil is used here as

$$\begin{bmatrix} \mathbf{x}_3 & \mathbf{x}_6 & \mathbf{x}_9 \\ \mathbf{x}_2 & \mathbf{x}_5 & \mathbf{x}_8 \\ \mathbf{x}_1 & \mathbf{x}_4 & \mathbf{x}_7 \end{bmatrix},$$

Applying the 2D-IRBF procedure to this stencil leads to a system of algebraic equations for unknown nodal values of the field variable as follows.

$$\begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \end{pmatrix} = \underbrace{\begin{bmatrix} \mathcal{G}_x^{[0]}, & \mathcal{O} \\ \mathcal{G}_x^{[0]}, & -\mathcal{G}_y^{[0]} \end{bmatrix}}_{\mathcal{C}} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}, \tag{2.16}$$

where $\mathcal{C}$ is the conversion matrix, $\widetilde{\mathbf{w}}_x$ and $\widetilde{\mathbf{w}}_y$ the RBF weight vectors of length 15; $\widetilde{\mathbf{u}}$ the vector of length 9, and $\widetilde{\mathbf{0}}$ the zeros vector of length 9; $\mathcal{O}$ the zeros matrix of dimension 9×15, and $\mathcal{G}_x^{[0]}$ and $\mathcal{G}_y^{[0]}$ the known matrices of dimensions 9×15. Furthermore, $\widetilde{\mathbf{u}}$, $\widetilde{\mathbf{w}}_x$ and $\widetilde{\mathbf{w}}_y$ are given by

$$\widetilde{\mathbf{u}} = (u_1, \ldots, u_9)^T, \tag{2.17}$$

$$\widetilde{\mathbf{w}}_x = (w_{x_1}, \ldots, w_{x_9}, C_1^x(y_1), C_1^x(y_2), C_1^x(y_3), C_2^x(y_1), C_2^x(y_2), C_2^x(y_3))^T, \tag{2.18}$$

$$\widetilde{\mathbf{w}}_y = (w_{y_1}, \ldots, w_{y_9}, C_1^y(x_1), C_1^y(x_2), C_1^y(x_3), C_2^y(x_1), C_2^y(x_2), C_2^y(x_3))^T, \tag{2.19}$$

$$\mathcal{G}_x^{[0]} = \begin{bmatrix} G_{[x]1}^{[0]}(\mathbf{x}_1) & \cdots & G_{[x]9}^{[0]}(\mathbf{x}_1) & x_1 & 0 & 0 & 1 & 0 & 0 \\ G_{[x]1}^{[0]}(\mathbf{x}_2) & \cdots & G_{[x]9}^{[0]}(\mathbf{x}_2) & 0 & x_2 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & x_3 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & x_4 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & x_5 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & x_6 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & x_7 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & x_8 & 0 & 0 & 1 & 0 \\ G_{[x]1}^{[0]}(\mathbf{x}_9) & \cdots & G_{[x]9}^{[0]}(\mathbf{x}_9) & 0 & 0 & x_9 & 0 & 0 & 1 \end{bmatrix}, \tag{2.20}$$

$$\mathcal{G}_y^{[0]} = \begin{bmatrix} G_{[y]1}^{[0]}(\mathbf{x}_1) & \cdots & G_{[y]9}^{[0]}(\mathbf{x}_1) & y_1 & 0 & 0 & 1 & 0 & 0 \\ G_{[y]1}^{[0]}(\mathbf{x}_2) & \cdots & G_{[y]9}^{[0]}(\mathbf{x}_2) & y_2 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & y_3 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & y_4 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & y_5 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & y_6 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & y_7 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & 0 & 0 & y_8 & 0 & 0 & 1 \\ G_{[y]1}^{[0]}(\mathbf{x}_9) & \cdots & G_{[y]9}^{[0]}(\mathbf{x}_9) & 0 & 0 & y_9 & 0 & 0 & 1 \end{bmatrix}, \tag{2.21}$$

where $G_{[x]i}^{[0]}$ and $G_{[y]i}^{[0]}$ $(i = 1..9)$ were defined in Section 2.1.1 in the x and y-directions and x_i, y_i are the two components of $\mathbf{x}_i$. It is noted that in Eq. (2.16)

$$\widetilde{\mathbf{u}} = [\mathcal{G}_x^{[0]}, \mathcal{O}] \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}$$

is obtained by collocating the field variable over a local stencil, and

$$\widetilde{\mathbf{0}} = [\mathcal{G}_x^{[0]}, -\mathcal{G}_y^{[0]}] \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}$$

is derived from the consistency condition $\int\int \frac{\partial^2 u}{\partial x^2} dx \Big|_{\mathbf{x}_i} = \int\int \frac{\partial^2 u}{\partial y^2} dy \Big|_{\mathbf{x}_i}$.

The conversion of the network-weight space into the physical space is achieved by first inverting Eq. (2.16)

$$\begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C}^{-1} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \end{pmatrix}. \tag{2.22}$$

Then by substituting Eq. (2.22) into Eqs. (2.7) and (2.9) the first order derivatives of u with respect to x and y and the function itself over a local stencil are determined.

2.2 Domain decomposition method

The first idea of domain decomposition (DD) method was introduced by Schwarz (1972). In its early definition, DD is the process of splitting of the considered domain of a problem into smaller sub-domains. In a broader view, the idea of DD can be used in different ways. From the point of view of physics, DD technique aims to divide the domain into regions where different physical models may be applied. From the numerical modelling point of view, it is sometimes helpful to use different approximation schemes in different regions. The three most important motivations for the DD technique include

- high potential for parallelisation, some good parallel performances have been observed;
- simplification of problems on complicated geometry; and
- superior convergence properties.

Thanks to these motivations, DD technique is applied in many numerical methods, for example in FDM (Dawson and Dupont, 1994; Balls and Colella, 2002), FVM (Marshall et al., 1997), FEM (Marini and Quarteroni, 1989; Cowsar et al., 1995; Becker et al., 2003) and recently in RBF methods (Beatson et al., 2001; Zhou et al., 2003; Li and Hon, 2004; Mai-Duy and Tran-Cong, 2004; Tran et al., 2009). DD is also widely used in molecular dynamics (MD) where simulations normally concern the movement and interaction of hundreds of thousands of molecules. Notable applications of DD in MD can be found in (Jabbarzadeh et al., 1997; Koradi et al., 2000; Ilnytskyi and Wilson, 2002; Jabbarzadeh et al., 2003).

In particular, for solving PDEs, discretisation methods like FDM or other methods transform the problem to the solution of a system of algebraic equations. Such system is often so big that direct factorisation is difficult or not effective. In addition, the use of basic iterative methods, such as the conjugate gradient, could result in slow convergence. The first advantage of DD idea is that in many cases it is the most practicable way to solve large scale problems. Moreover, for some problems, proper implementation of DD will significantly increase the convergence rate of the solution. And last but not least, by replacing the original problem by a set of sub-problems of reduced size, DD technique provides an effective way to alleviate numerical difficulties associated with large populated matrices such as ill-conditioning.

The original DD method proposed by Schwarz (1972) was supposed to solve classical boundary value problems (BVPs) involving harmonic functions. This is probably the earliest DD method called Schwarz alternating method. In this method, the domain is divided into overlapping sub-domains with Dirichlet boundary condition on the artificial boundary (AB). In a consecutive step, a sub-domain is considered as a well posed BVP and is solved independently. The obtained results in this sub-domain is used to update the boundary condition of its adjacent sub-domains. This Schwarz's method defines a class of DD methods called overlapping DD methods. Overlapping DD method have the advantage of intuitive

presentation and implementation. The value on AB of one sub-domain is obtained directly from the value of corresponding points inside adjacent sub-domains and thus neither interpolation nor extrapolation is needed. The Dirichlet type of AB allows insignificant change in the solving algorithm as well as guarantees the convergence over the interface (Quarteroni and Valli, 1999). However, the fact that sub-domains must overlap, makes the method less applicable in irregular domains. This method also adds an overhead of degrees of freedom (DoF) to each sub-domain.

The second class consists of non-overlapping DD methods, which can be divided into two sub-classes, namely the Schur complement method and the Steklov-Poincare method. The Schur complement method or sub-structuring method came from an approach proposed by Przemieniecki to divide a structure into substructures with the assumption that all ABs are fixed (Gander and Tu, 2014). By this approach, the problem firstly is divided into sub-problems with non-overlapping ABs. Then the solutions on ABs are determined by solving a system of equations created by the Schur complement method. After the AB solutions are obtained, each sub-domain becomes a separate BVP which can be solved by using a suitable numerical method. The Schur complement method has high potential for parallelisation because once the AB solutions are obtained, all sub-domains are totally independent. Nevertheless, obtaining AB solution is a time consuming process that cannot be parallelised. As a result, the time efficiency of the method is not very impressive.

Apart from Schur complement method, Steklov-Poincare method focuses on constructing an equivalent problem by introducing transmission conditions. The transmission conditions are, in fact, the continuity of the function or its derivatives across the ABs. Similar to the overlapping method, sub-domains become separate BVPs which can be solved in parallel within a time step. The results obtained from sub-domains are used to approximate the boundary condition on ABs. It is clear that the non-overlapping method creates less DoF overhead than the overlapping one since the overlapping area is reduced to a single point for 1D problems or a single line for 2D problems. For this method, the potential of parallelisation is high as all sub-domains are independent within a time step.

2.2.1 Overlapping domain decomposition method

Schwarz multiplicative overlapping method

Consider a second-order PDE with boundary conditions as follows.

$$\begin{cases} \mathcal{L}u &= f, \quad \mathbf{x} \in \Omega, \\ \mathcal{B}u &= g, \quad \mathbf{x} \in \partial\Omega, \end{cases} \tag{2.23}$$

where $\mathcal{L}$ is a second order differential operator; $\mathcal{B}$ - an operator imposed as boundary conditions such as Dirichlet, Neumann or a mixture of both; u - an unknown function; f and g - given functions; Ω and $\partial\Omega$ - the domain under consideration and its boundary, respectively.

An example of a 2D domain with two overlapping sub-domains is provided in Fig. 2.1. The domain Ω is divided into two sub-domains Ω_1 and Ω_2. Let $\partial\Omega_1$ and $\partial\Omega_2$ and Γ_1, Γ_2 be the real and the artificial boundaries of Ω_1 and Ω_2 respectively. The boundary condition imposed on Γ_1, Γ_2 is of Dirichlet type.

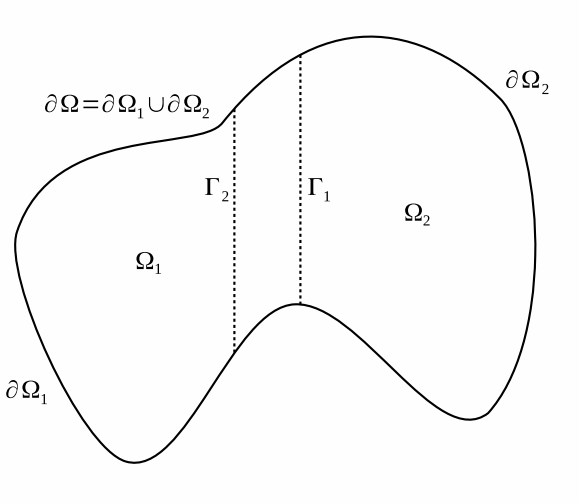


Figure 2.1 Non-overlapping DD method with two sub-domains Ω_1 and Ω_2 in 2D.

The Schwarz multiplicative overlapping method is described at a step k as follows.

$$\begin{cases} \mathcal{L}u_1^k = f, & \mathbf{x} \in \Omega_1 \\ \mathcal{B}u_1^k = g, & \mathbf{x} \in \partial\Omega_1 \\ u_1^k = u_2^{k-1}, & \mathbf{x} \in \Gamma_1 \\ \mathcal{L}u_2^k = f, & \mathbf{x} \in \Omega_2 \\ \mathcal{B}u_2^k = g, & \mathbf{x} \in \partial\Omega_2 \\ u_2^k = u_1^k, & \mathbf{x} \in \Gamma_2 \end{cases} \tag{2.24}$$

The Schwarz multiplicative method is a sequential method because the values on ABs in each sub-domain are updated from the most recent results of its neighbouring sub-domains (NSs). Indeed, as can be seen from Eq. (2.24) sub-domain Ω_2 has to wait until sub-domain Ω_1 finish the computation in order to obtain the value on the AB Γ_2. Clearly, the computations on Ω_1 and Ω_2 cannot be performed in parallel. To solve Eq. (2.24) in parallel a black-white colouring technique (Smith et al., 1996) can be utilised, so that all non-adjacent sub-domains are put in the same group marked by a colour. Normally, two groups namely black and white will be formed. By this way, in the first half-step, operations in one group, e.g. the white one, can be executed in parallel. In the next half-step, the other group operations (the black one) will be executed in the same manner.

Schwarz additive overlapping method

In this method, the values on AB in each sub-domain problem at step k are updated from the results of its NS's in previous step $k-1$. In other words, Eq.

(2.24) is changed to

$$\begin{cases} \mathcal{L}u_1^k &= f, & \mathbf{x} \in \Omega_1 \\ \mathcal{B}u_1^k &= g, & \mathbf{x} \in \partial\Omega_1 \\ u_1^k &= u_2^{k-1}, & \mathbf{x} \in \Gamma_1 \\ \mathcal{L}u_2^k &= f, & \mathbf{x} \in \Omega_2 \\ \mathcal{B}u_2^k &= g, & \mathbf{x} \in \partial\Omega_2 \\ u_2^k &= u_1^{k-1}, & \mathbf{x} \in \Gamma_2 \end{cases} \tag{2.25}$$

As shown in Eq. (2.25) the data in each sub-domain are independent within an iterative step. The method is suitable for parallelisation. However, as sub-domains do not use the newest values on the ABs, the additive method suffers from deterioration of convergence rate. Fortunately, in most of the cases this deterioration is negligible. The overlapping DD method will be used in combination with the local and compact local IRBF schemes to develop parallel methods presented in Chapters 4, 5, 7.

2.2.2 Non-overlapping Dirichlet-Neumann DD method

The non-overlapping DD method is based on the multi-domain formulation of a PDE and Steklov-Poincare interface equation (Quarteroni and Valli, 1999). There are several types of non-overlapping DD method, such as Dirichlet-Neumann method, Neumann-Neumann method, Robin method or a method by Agoshkov and Lebedev (1985). These methods differ from each other by the way the transmission conditions over the interface are defined. In this work, the Dirichlet-Newman method is considered as it provides a great compatibility to IRBF method in the way boundary condition on ABs is imposed. The mathematical background as well as the convergence analysis of the Dirichlet-Neumann method are extensive topics, which are out of the scope of this thesis. Those information can be found in a comprehensive book by Quarteroni and Valli (1999).

An example of non-overlapping DD method with two sub-domain and an AB is shown in Fig. 2.2. Details of non-overlapping DD method will be presented in Chapter 6.

2.2.3 High performance computing for large scale problems using parallel DD methods

In recent decades, parallel computing has become more and more popular in the world of numerical methods owing to its promising ability to solve large-scale and complex practical problems. This big potential of parallel computing has motivated researchers to investigate DD method more intensively. Because of DD's parallel capability, there are many attempts to apply DD technique with RBF methods. Kansa (1990a) showed an example of using DD and multi-quadric MQ-RBF method to solve a hyperbolic problem. He stated that by incorporating DD into the MQ-RBF method, the problem can be solved much more efficiently. Another implementation of DD within RBF based interpolation problem for large scale problems by Beatson et al. (2001) also yielded a very good convergence rate.

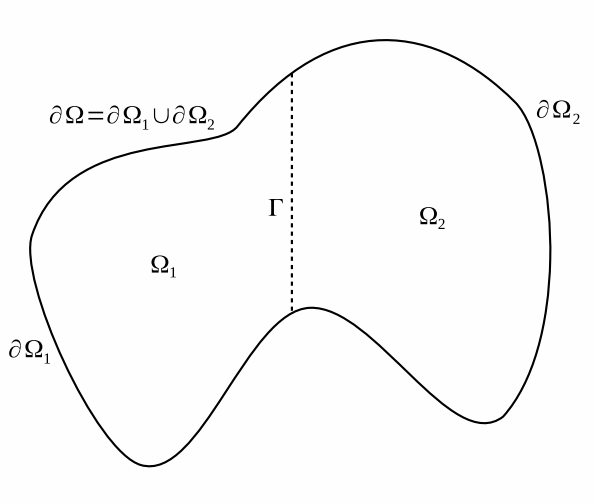


Figure 2.2 Non-overlapping DD method with two sub-domains Ω_1 and Ω_2 in 2D

Numerical results from Li and Hon (2004) indicated that the non-matching grid based DD scheme has similar convergence rate and accuracy as the matching grid case. A parallel DD technique for the hybrid simulation of dilute polymer solution flows using Brownian dynamics and RBF based methods was reported in Tran et al. (2009). The method has utilised the parallel-capable property of DD for the solution of both stochastic differential equations (SDEs) and PDEs and thus significantly decreased the computational time and enjoyed good convergence rate. In this project, a HPC solution for large scale problems is proposed via the combination of the DD technique and CLIRBF methods.

2.3 Parallel programming

Parallel computing has become a popular trend in science community. The complexity of new and practical problems requires an astonishing amount of computing resources that could not be provided by a single computer. Supercomputing systems have been introduced with many CPUs connected by a fast network. The primary goal of parallel programming is to achieve high computational performance. However, even when the parallel computers can provide the increased speed and abundance of memory, applications may not be able to make use of all the available resources if the underpinning algorithms are not appropriate. Unlike hardware, software cannot be automatically scaled up to work with parallel computers. As a result, the scalability of parallelism is an interesting topic for software developers.

2.3.1 Development of parallel computers

The development of parallel computers can be tracked back to the second half of the 1970s when vector computers were first introduced. Vector computers offered higher machine performance but the lack of application compatibility and paral-

lel algorithms made these computers less attractive to industrial customers. In the first years of the 1980s, parallel programming environments, operating systems and applications were included in vectors computers. Performance was also increased by improved chip technologies and the availability of shared-memory multi-processor systems. With new chip technologies, new generation of applications was formed, leading to the adoption of parallel computing by industrial customers. Later in 1980s, scalable parallel computing using distributed memory started to gain attention as it solved the hardware scalability limitation of shared memory systems. In the early 1990s, massively parallel processor (MPP) systems came to the market and began to gain popularity. MPP systems are, in fact, distributed memory systems with multiple nodes connected by fast networks.

2.3.2 Parallel computing architectures

From the point of view of computer architectures, parallel computing can be divided into memory parallelism and CPUs parallelism.

Memory parallelism

One model of parallel computer is shared memory, in which there is only one big memory and access to this memory is given by virtual address space (Fig. 2.3). CPUs are granted an equal access to this shared memory through a high-speed bus. This model has very low access latency and high bandwidth. However, when two or more CPUs try to access the same memory address, the race condition appears and may be severe. To prevent race condition, programmers need to provide explicit block mechanisms, e.g. the critical region, where only one CPU is allowed to be in the critical region at a time, that CPU has highest priority in accessing the memory, other CPUs have to wait until the critical region is released to enter. Particular attention also needs to be paid to maintain the consistency and coherency of the memory.

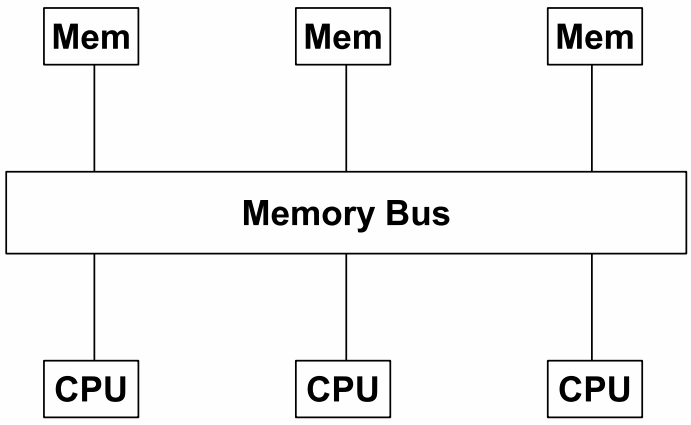


Figure 2.3 Shared memory parallel computing model.

Another model of parallel computers is distributed memory (Fig. 2.4). In this model, each CPU has its own physical memory, i.e. local memory. The communication between CPUs is performed by message passing mechanism. Connections between CPUs are handled by network interface cards (NIC) equipped in each CPU. Distributed memory model suffers from high latency communication but it is free of resource contention. Systems based on distributed memory are also easy to assemble, maintain and scale up.

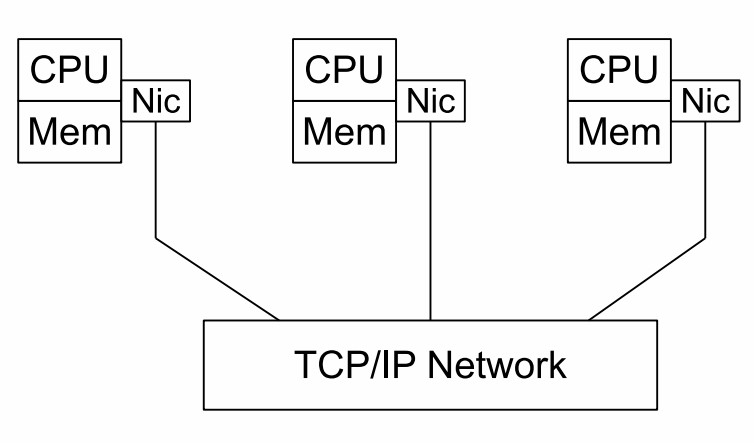


Figure 2.4 Distributed memory parallel computing model. Nic is network interface card.

Other important memory parallel architecture is vector computers. In vector computers, data is organised in vectors, which are groups of 64 floating point numbers. All operations are performed on vectors, hence the 64 operations are executed in parallel. Vector computers are popular in scientific applications where data are favourably presented in vectors. The high throughput of vector computers is supported by wide memory bandwidth. Vector computers are highly application specific. Their efficiency is high only when vector applications and vector data are used.

CPU parallelism

Super-scalar processing

When there are two or more instructions in the stack and they are going to use different functional units, e.g. Floating Point Unit (FPU) and Algebraic Unit (ALU), and different register sets, e.g. Floating Point Register (FPR) and General Purpose Register (GPR), a super-scalar computer can execute both instructions at the same time. This type of parallelism is also called instruction-level parallelism. The main advantage of super-scalar processing is that parallelism is done inside CPU and no change needs to be applied to the existing code nor the algorithm. However, it is a complex and intensive task to identify possible parallel instructions in pre-processing step. The efficiency of super-scalar processing highly depends on the nature of the code, and for many cases, is not high.

Explicit parallel instructions

During the compilation, the compiler uses sub-instructions to explicitly tell the

CPU to execute the code using different functional units and registers. The resultant instructions are way longer than the original ones hence the name Very Long Instruction Words (VLIW). The compiler is solely responsible for the parallelism. As compiler is designed to take all advantages of a specific CPU architecture, porting the code to other machines with different CPU architectures will likely eliminate the parallelism and in the worst case might cause fatal runtime error.

Multi-threading

Suppose that, there are four independent instruction sets that need to be executed and four functional units, each instruction set can be handled by one functional unit at the same time. This model is called simultaneous multi-threading with each instruction set is a thread.

Fine-grained multi-threading runs only one thread at a time while keeping other thread on hold. The benefit of this is when some thread performs a slow operation, e.g. hard drive access, it can be put on hold and give other thread a time window to run. This type of multi-threading is also seen in single CPU computer where the operating system schedules threads to achieve the best throughput of the CPU.

Graphics processing unit (GPU) parallel computing

Graphics processing unit is a specialised unit that handles computation only for computer graphics. At the beginning, the GPU is designed to work with vectors and matrices data because images are normally stored the same way. That means GPU is a vector computer by nature. The parallel potential of GPU has recently been increased by adding multiple GPUs into one computer. The underlying programming interface allows data to transfer from CPU to GPU for parallel computation, and the result is then transferred back to the CPU. This process is called general-purpose computing on graphics processing units (GPGPU). The advantage of GPU over CPU is that, GPU can perform operations on graphical data at a speed much faster than CPU. However, data on CPU must be converted to graphical-like data before they can be processed by GPUs.

It is considerably relevant to mention the well-known Flynn's taxonomy of parallel computers. Flynn (1966) categorised parallel computers based on their data streams and instruction streams. A sequential computer has a single data stream and single instruction stream, for that it is called SISD. A vector computer is an example of single instruction-multiple data streams machine or SIMD. A later and more popular variation of SIMD is simple program-multiple data (SPMD), which covers broader range of parallel applications. Many parallel computing systems, e.g. MPP systems, have multiple instruction streams as well as multiple data streams. These systems form the MIMD group. The last group MISD, which means multiple instructions, single data, can be subtly referred to as shared-memory parallel computers. However, to exactly reflect the MISD idea, applications on these computers need to perform different actions on the same set of data.

Parallelism today is a combination of many approaches and techniques. The most

popular model is distributed memory clusters of nodes, in which each node is a shared-memory multi-processor machine.

2.3.3 Decomposing programs for parallelism

There are three main steps of the development of a program for parallel computers (i) identification; (ii) decomposition and (iii) implementation (Dongarra et al., 2003).

Identification

The first step is to identify parts of the program that can be executed in parallel. Based on Bernstein (1966), two parts P_1 and P_2 of a program can run concurrently if none of the following holds

1. P_1 writes into a location that is later read by P_2 - a read-after-write race;
2. P_1 reads from a location that is later written into by P_2 - a write-after-read race;
3. P_1 writes into a location that is later overwritten by P_2 - a write-after-write race.

Decomposition

There are two common strategies to decompose a program for parallel computing. The first one is to decompose a program into independent tasks so that different CPUs can handle these tasks in parallel. This strategy is called task parallelism or functional parallelism. Task parallelism is commonly found in multi-core machines where each core is assigned a specific task.

The other strategy is called data parallelism, which decomposes a program in a way that the same code runs on different portions of the data. Data parallelism is more popular in practice owing to its stable load balancing. Once the data is divided, the load is even between CPUs until the computation is finished.

Implementation

Before writing the code, a developer needs to decide what memory model is best suited for the problem in hand. The shared-memory model offers fast access to global memory but requires blocking mechanism to prevent race conditions. The distributed-memory model is blocking free since each CPU is equipped with a local memory. However, communication between CPUs is needed to exchange

data. In addition, synchronisation is strictly required to maintain the consistency of the data on all CPUs.

There are several ways to implement parallel programs. The simplest way is to create a parallel loop, in which iterations are executed in parallel instead of sequentially. All iterations must be independent, in other words, the computation of one iteration does not require data from other iterations. Consider following loop

```
for i = 1:n
        a(i) = a(i) + i;
end
```

Where a is an array of size n. It is clear that in each iteration the value of $a(i)$ is calculated using only the value of itself and the index i. Therefore, the code can be rewritten in parallel form as bellow

```
parallel_for i = 1:n
        a(i) = a(i) + i;
end
```

Actually, there is no changes in the code except the keyword for being replaced by $parallel\_for$. In fact, it is the keyword $parallel\_for$ that tells the compiler to add special sub-instructions to the original execution code to command the CPUs to execute the loop in parallel. Parallel loops are very handy and easy to use. However, if those loops are not dominant in computation, the efficiency of parallelism will be very limited.

Single program-multiple data is another approach to implement parallel programs. SPMD aims to divide the problem by data and run the same code on these multiple data in parallel. Basically, parallel loop can also be considered a SPMD program. As seen in the example above, the same summation operation is performed on different elements of a. But in general, SPMD provides a more flexible way to decompose problems into parallel parts. In SPMD, at the beginning of the computation, all global data need to be replicated on every CPUs, and at the end of an iteration, explicit communication needs to be specified to exchange data between CPUs.

Parallel programs can also be implemented by recursive task programming. With a tree-like data structure, every time the computation reaches a branch, a new thread is created to handle the computation on that particular branch.

2.3.4 Challenges to parallel performance

Early models of parallel computers and applications did not seem attractive to customers due to their low performance-to-cost ratio. There are four main factors that affect the performance of parallel programs including scalability, communication, load balancing and irregularity (Dongarra et al., 2003).

To measure the performance of a parallel program, the speed-up is calculated as follows.

$$S_{(n)} = \frac{T_{(1)}}{T_{(n)}},$$

where $T_{(n)}$ is the computation time using n processor(s). A scalable parallel program is the one that has $S_{(n)} \approx n$. However, due to the present of a part of program that cannot be parallelised $S_{(n)} < n$. Suppose T_s and T_p are the time spent on sequential and parallel part respectively then the speed-up is given by Amdahl's Law (Amdahl, 1967)

$$S_{(n)} = \frac{T_s + T_p}{T_s + T_p/n} \leq \frac{T_{(1)}}{T_s}$$

It can be seen that the speed-up is limited by the ratio of the time of sequential part to the time of the parallel part in that program. Suppose that 20% of the time of sequential program is non-parallel, the maximum speed-up is 5 no matter how many CPUs are used.

The only problem with Amdahl's law lies in the fact that regardless of number of CPUs is used, the maximum speed-up of a parallel algorithm is always bounded by a factor of $\frac{1}{T_s}$, which is not always true. Gustafson (1988) pointed out this controversial prediction where they were able to obtain speed-up up to 1021 using 1024 CPUs with the sequential part of the algorithm contributing 0.4% of the total simulation time. This contradiction inspired the Gustafson-Barsis's law. In this law, the problem scale is considered as an important parameter. To measure the efficiency of a parallel algorithm, the actual times run on parallel CPUs are chosen as reference. Let T_s^* and T_p^* be the sequential part and parallel part of runtime on parallel CPUs. Then a hypothetical runtime $T_{(1)}^*$ on a serial processor is calculated as

$$T_s^* + NT_p^*$$

The so-called scaled speed-up becomes

$$S^* = \frac{T_s^* + NT_p^*}{T_s^* + T_p^*}$$

It is clear that with this formula, if T_s^* is small or diminished, the speed-up grows proportionally with the number of CPUs. Hence, scaled speed-up is particularly useful in the estimation of the efficiency of parallel algorithms that solve large-scale problems where it is unlikely that the computation time on a single CPU can be determined. In addition, as the scaled speed-up suggests, the limitation caused by the sequential part of a program can be mitigated by increasing the amount of computation which can be carried out in parallel.

Communication is also an important issue in parallel computing, especially with distributed memory model. It is understandable that the communication time

should take a fraction of the computation time. However, as the scale of the program grows, poor designed communication scheme may end up with exponential increase of number of messages being passed.

Another big challenge to parallel computing programmers is load balancing, which require an even distribution of load between CPUs. Bad load balancing is often the reason for low efficiency of parallel program. As a quick example, if in a parallel system one CPU takes half of the parallel work, the speed-up will never excess 2 regardless of number of CPUs used. The most common reason for poor load balancing is the irregularity of the problem under consideration. With irregular problems, the load is often not available before run-time, and even if the load is available, it can change during run-time. A possible solution to this problem is to employ a dynamic load balancing mechanism that monitors the load on all CPUs and rebalances the loads if necessary.

2.4 Distributed Termination Detection

In distributed computing, a often encountered problem is how to detect whether all processes are passive and the system is ready to be terminated. This problem was discovered by Francez (1980); Dijkstra and Scholten (1980) and called as the Distributed Termination Detection. The DTD problem might look trivial to sequential computing where a single process can terminate when the computation is finished. However, in parallel computing even when all processes have finished the computation, there may still be some messages being in transit. These messages, upon arrival, will reactivate other processes and that actually means the computation has not finished yet. Generally, a DTD algorithm aims to detect a system's quiescence. Quiescence is a state, in which no process is active and no message is in transmission. The difficulty of DTD comes from the fact that, in distributed computing, there is no shared clock or shared memory. Thus, a process has no way to know the status of other processes. So far, there are three classes of DTD, including wave algorithms, parental responsibility algorithms and credit and recovery algorithms. In this chapter, these classes are presented briefly. More details can be found in Matocha and Camp (1998) and Raynal (2013).

The wave DTD algorithms is initially based on a token ring network topology. A root process creates a token and sends it into the ring. The token is passed through all other processes in the ring and comes back to the root process. The root process then analyses the information in the token and tries to detect the termination. As the token ring network is enclosed, the token travels the whole network and comes back to its root in just one round. Thus, for token ring network, wave algorithms are very efficient. However, as modern network topologies have become very diverse, wave algorithms have changed to utilise the network graphs. With this change, wave algorithm now consists of two phases. In the first phase, many tokens are passed from the root process through a graph to all processes in the network. In the second phase, tokens are passed back to the root process for analysis. The wave algorithms have a major drawback: it is repetitive. Tokens are sent out and collected from beginning until termination is detected.

This adds a significant overhead to the main computing task. In addition, the message complexity of wave algorithms is high. In many wave algorithms, the message complexity is $O(M \times n)$ in the worst case, where M is the number of basic messages, i.e. the messages of the main computation task, and n is the number of processes in system.

In parental responsibility algorithm, a set of rules is predefined before the system starts to operate. Basically, the process that activates other processes immediately becomes a father process and the one that is activated becomes child process. The father process is responsible for the status of its child processes. Father process can become passive only after all of his child processes have become passive. In such of a system, the root process is the first one to be active and also the last one to be passive. Once the root process becomes passive it can declare that the termination is detected. The number of messages sent by DTD algorithm is minimised, only one message is sent when a process is activated or becomes passive. However, as the algorithm suggests, there is no recovery mechanism. In other words, if any father process fails, the algorithm has no way to resume.

For the credit and recovery algorithm, there is a total credit which is a number equal to 1 in general case. The root process initially holds all credits. When an active process activates another process, it divides its credit into two halves and send one half to the activated process. Process that is going to become passive will send its credit to another active process. In this way, all credits are kept in active processes. At any time, when a process collects all credits, it can detect the termination. In this algorithm, any node can detect termination. The last active process needs not necessary be the first active one. However, an obvious problem with credit and recovery algorithm is the allocation of credit. When the number of processes increases, the value of credit will decrease very fast and over the recognisable range of a computer. Therefore, some credits will be lost and the system will never be able to collect enough credit to detect the termination. Moreover, most of credit and recovery algorithms have message complexity far beyond the optimal value $O(M)$, where M is the number of basic messages.

In this research project, the characteristics of the credit and recovery class is considered to build up a new DTD algorithm which is used to detect the termination. The proposed algorithm, named the Bitmap DTD, possesses some important advantages, including (i) it allows any node to detect termination, i.e. a symmetric algorithm; (ii) it does not require any central control agent; and (iii) the message complexity of the proposed algorithm is optimal. The algorithm will be presented in Chapter 3.

2.5 Hardware and software specification

Most of simulations in this thesis were performed on a HPC cluster at University of Southern Queensland (USQ). Some experiments were carried out on HPC clusters at University of Queensland (UQ) and Queensland University of Technology

(QUT) which are members of the Queensland Cyber Infrastructure Foundation. The specifications of the USQ HPC cluster are as follows

- 30 nodes in Sun X2200 servers, each of which contains two quad-core 2.7Ghz AMD Opteron CPU and 16Gb DDR2-667 memory.
- Total of 480Gb of RAM.
- Approximately 30Tb of global storage.

There are two standards for parallel programming, namely Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). PVM was designed to work on heterogeneous networks. Therefore, it possesses very good interoperability between different hosts as well as powerful dynamic resource manager and process control function. On the other hand, MPI was proposed later as a standard for massive parallel computer vendor to implement parallel applications on their products. MPI provides rich communication routines and an ability to specify communication topologies. MPI also supports the design of safe parallel software libraries. The main tool used in this thesis is the MATLAB distributed computing toolbox (MDCT) and MATLAB distributed computing engine (MDCE). In MATLAB, all low level operations are handled by message-passing routines based on an MPI standard library (MPICH2). The use of MDCT is to create a parallel pool on local machines. This pool allows for designing, implementing and debugging parallel algorithms. The local parallel pool is particularly useful for debugging where an error can be tracked back to the line of code that triggered it. MDCE is similar to MDCT because it provides an environment to run the same code and produce the same results. The only difference is that MDCE is deployed on a massive distributed computing cluster. In our simulations, MDCE is used to verify the efficiency of our parallel algorithms.

2.6 Conclusion

Some brief reviews of the basic tools for the work of the thesis, such as RBF approximation methods, DD techniques, parallel computation and DTD algorithms, were presented in this chapter.

The universal and effective IRBF approximation method has a high order convergence rate and does not require an expensive meshing procedure associated with other numerical methods such as FVMs and FEMs. The method is enhanced with local and compact schemes which help make the system matrix sparse and hence improve its condition number.

Domain decomposition technique has gained more and more attention since the first introduction of parallel computing in the 1980's. With the advantage of being able to solve large scale problems and high potential for parallelisation, DD methods have been employed to solve several problems in mechanics. The two main classes of DD techniques are overlapping and non-overlapping ones. In this work, both methods are considered and employed in combination with IRBF to develop parallel algorithms to solve CFD problems.

The termination of distributed system has been a non-trivial problem in the parallel computing community because there is no in-built mechanism in a distributed system to detect when all processes in the system have finished the computation. There have been many efforts in designing an effective algorithm to perform this job including wave algorithms, parental responsibility algorithms and credit-recovery algorithm. An optimal DTD algorithm has to be able to detect the global status of the system with a small delay and low message complexity. In this work, such an algorithm, namely the Bitmap DTD, is developed and incorporated into parallel methods. The Bitmap DTD itself is presented in the next chapter.

Chapter 3

Bitmap distributed termination detection algorithm with applications in parallel domain decomposition computation

In this chapter, a new Distributed Termination Detection (DTD) technique will be presented in detailed. The technique will be incorporated into parallel algorithms developed and reported in subsequent chapters. In distributed computing, the DTD problem concerns the discovery whether all processes in a distributed system have finished their job. This problem is not a trivial problem since in distributed system there is neither a global synchronised clock nor shared memory. In fact, the DTD is related to a more general problem, namely the detection of global property of a distributed system. In this chapter, we propose a new algorithm called Bitmap DTD. This algorithm is designed to work with domain decomposition (DD) method in solving PDEs. The DD method adds more strict requirements to a DTD, such as the communication must be synchronous and the termination on individual process must be synchronised with the whole system. The algorithm is inspired by the weight-throwing DTD (or Credit/Recovery). However, in our algorithm, a bitmap is used to carry the snapshot of the system from process to process. The proposed algorithm possesses the following desired characteristics

- it allows any process to detect termination;
- it does not require any central control agent;
- the termination detection delay is low;
- the message complexity of the proposed algorithm is nearly optimal.

3.1 Introduction

Back in the 1980s, when distributed computing was first introduced the DTD problem has been raised and investigated continuously. The pioneer in this field

of parallel computing is Dijkstra, who proposed two methods for solving the DTD problem (Dijkstra and Scholten, 1980; Dijkstra et al., 1983). Based on these very first methods, two main categories of DTD, namely wave algorithm and parental responsibility algorithm, were defined. Later on, Mattern (1989) proposed a credit and recovery algorithm, which forms the third class. A very good taxonomy and comparison of DTD algorithms can be found in (Matocha and Camp, 1998).

For wave algorithms, each iteration consists of two phases. In the first phase, tokens are passed from a process, called root, to all other processes in the network. In the second phase, the tokens are returned to the root. Based on the information from the returned token the root can detect the termination. Wave algorithms were proposed to work with token ring network topology, in which all processes are connected in an enclosed ring. With this topology the algorithm is visibly simple. The root process initiates only one token and sends it on to the ring. After passing all the processes in the ring, the token comes back to the root in just one round (Dijkstra and Scholten, 1980). However, the token ring topology has become less common in practise. Instead, other network topologies, such as mesh, star, bus, tree, line and hybrid ones are more popular. In these networks, wave algorithms need to rely on a graph to function properly. The idea remains the same, in the first wave, many tokens are passed from the root of the tree through all intermediate processes and finally reach all the leaves. After that, tokens are passed back to the root in the second wave (Huang, 1988). The biggest drawback of wave algorithms is that it is a repetitive method, i.e. waves are sent out one after another until termination is detected. In general, the message complexity is $O(M \times n)$, where M is the number of basic messages and n is the number of processes in the system.

In parental responsibility algorithms (Dijkstra and Scholten, 1980), all processes in the system are always linked to their predecessor and/or successor by a relationship called parental relationship. The predecessor is responsible for its successors whenever it sends a message and activates successor, it also is not allowed to become passive until all of its successors are passive. A successor sends a signal to its predecessor right before it becomes passive. In this scheme, the root is the first process that becomes active and also the last process that becomes passive and detects the termination. Although the message complexity of these algorithms is optimal $O(M)$, they are asymmetric. An asymmetric algorithm requires a graph to be constructed before it actually works. In addition, the root is the only process that can detect the termination. If the root fails, the system will not be able to terminate.

The third class of DTD is credit and recovery in which each active process is assigned a portion of a predefined number of credits. This portion of credit will be returned to another active process right before a process becomes passive. This mechanism guarantees that no passive process holds any credit. When a process holds all credits, termination is detected. The main drawback of this type of DTD is the allocation of credits. As an example, in (Mattern, 1987), the initial credit is chosen to be 1. When a process is activated, the credit is divided into two halves and one half is sent out. By this way, with a large number of processes, the credit will decrease tremendously and some portion of

the credit can be lost. In (Huang, 1989), a space-efficient credit encoding scheme was introduced. The credit is defined by a specific length and an index. This index indicates the position of the window of that credit in a full-range credit, which is hold by a controlling agent. Although the scheme solves the problem related to the shortage of credit, it involves the use of a centralised controlling agent. Hence, the algorithm will be broken if the controlling agent fails to operate. Most of credit and recovery algorithms have optimal message complexity, which is $O(M)$.

The DTD algorithm presented in this chapter belongs to the third category. However, the presented DTD uses a bitmap rather than a credit to detect the termination. The use of bitmap makes the algorithm symmetric as all processes will run the same code. Owing to this, any process can detect the termination. The present algorithm requires neither a central process to monitor the state of the system nor a spanning tree graph. The message complexity of our DTD is optimal $O(M)$ and the termination delay is D steps, where D is the diameter of the network. The algorithm is presented in Section 3.3.

3.2 The DTD problem in DD method based parallel computation

Domain decomposition method is an efficient approach for solving large scale problems in mechanics and specifically in computational fluid dynamics (CFD). This method has been drawing more and more attention in recent decades owing to its high potential for parallel computing. The basic idea of DD method is to split the domain of a problem into smaller sub-domains and solve the problem in each of these sub-domains. A parallel program based on DD method can be seen as a distributed system, in which each process runs on a separate CPU and does not share any data with other processes. In this system, since processes do not directly share any data they cannot know the status of other processes. Therefore, in order to detect whether all processes are ready to terminate, a dedicated DTD algorithm is required.

When solving CFD problems using DD method on a parallel distributed system, it is crucial to keep the computational task in all sub-domains consistent. Within a DD method, the values on artificial boundaries (ABs), i.e. the boundaries used to separate sub-domains, are unknown. In overlapping DD methods, the values on ABs are exchanged between adjacent sub-domains after one iteration. These exchanges are done through synchronous communication supported by Message Passing Interface (MPI).

A summary of important properties of a distributed system based on DD method is as follows.

1. A process is considered passive if the solution of the problem being solved has converged. Passive is a stable property, i.e. once a process reaches this state it will remain in this state forever. A process in passive state is ready

to terminate. However, it has to keep running to exchange with other active processes as well as maintain the consistency of its solution with the whole system.

2. A system is considered passive when all processes are passive and all channels are empty.
3. The computation is synchronous, i.e. in one iteration, all processes must finish their computation task and exchange data with their neighbouring processes before advancing to the next iteration. The communication is synchronous, i.e. message delivery is guaranteed by MPI.

The condition 3 guarantees that all communication channels are empty after each iteration. It is because, in synchronous communication, a process always knows the number of messages that it expects to receive. Therefore, once the process has received all expected messages, there will be no message in transit heading to it.

3.3 Bitmap DTD algorithm

In this section, the Bitmap DTD algorithm is presented. As the name suggests, in this algorithm, a bitmap is used to store the state of all processes in the system. Before presenting algorithm, it is necessary to define some terminologies.

Bitmap: Bitmap is a group of binary bits. Each bit can have only one of two values 0 and 1, which are equivalent to active and passive states of a process, respectively. The length of a bitmap is equal to the number of processes in the system. For example, for a system of 32 processes, it is sufficient to use a bitmap of 32 bits. Then the bitmap could have any value in the range from 0 to $(2^{32}-1)$.

Process-Index: is a numeric index of a process in distributed system enumeration. This index also defines the position of the bit, representing that process in a bitmap (Fig. 3.1);

Stop-Code: a bitmap with all bits of 1, which means all processes are passive;

Step-To-End (STE): the number of iterations left for each process to execute before termination.

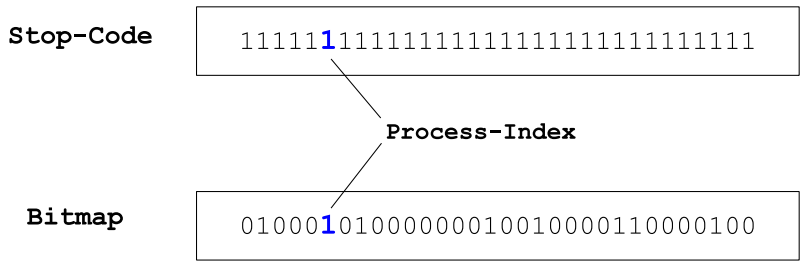


Figure 3.1 Example of bitmap and Ready-Code for a process with Process-Index 6 in a system of 32 sub-domains

3.3.1 The algorithm

Bitmap DTD algorithm consists of two parts: (i) Termination Detection and (ii) Synchronous Termination. The Termination Detection part aims to detect whether all processes in the system are passive. Once the globally passive state is detected, the Synchronous Termination is activated to get all processes terminated simultaneously at the same step.

Synchronous Termination is a very important feature in synchronous distributed system. If one process terminates while some processes are still active, those active processes cannot exchange information with the terminated process and thus cannot finish their work. Consequently, the system will end up with deadlock.

The Termination Detection algorithm is expressed by a flowchart in Fig. 3.2.

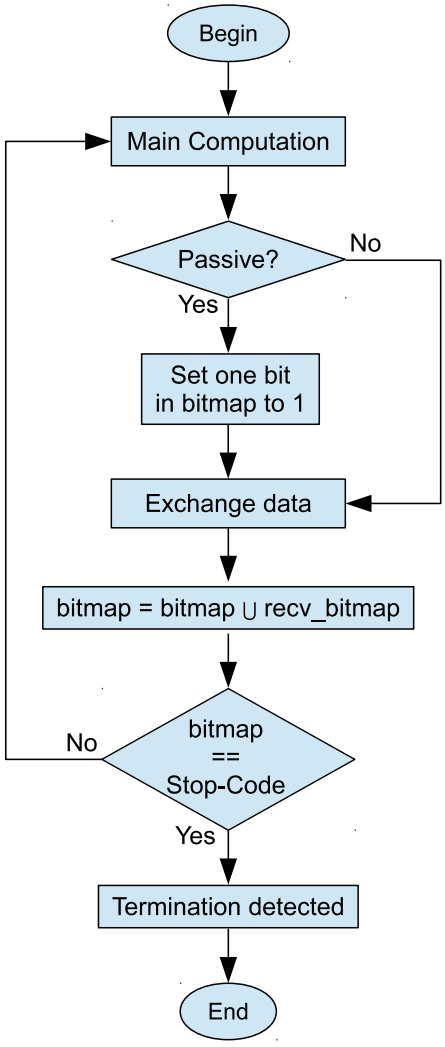


Figure 3.2 Bitmap Termination Detection algorithm. Recv_bitmap is the bitmap received from a neighbour

The algorithm starts by setting a zero bitmap. After each iteration of the underlying computation, if a process becomes passive, the bit corresponding to its

Process-Index will be set to 1. And then the bitmap will be exchanged with neighbouring processes. When a process receives a bitmap from one of neighbouring processes, it will update its own bitmap by a binary union of its current bitmap and the received bitmap. The process then checks the value of its bitmap. If its bitmap is equal to the **Stop-Code**, the process detects termination. Otherwise, the algorithm is repeated.

When the termination is detected, the Synchronous Termination algorithm is started. A flowchart of Simultaneous Termination is shown in Fig. 3.3.

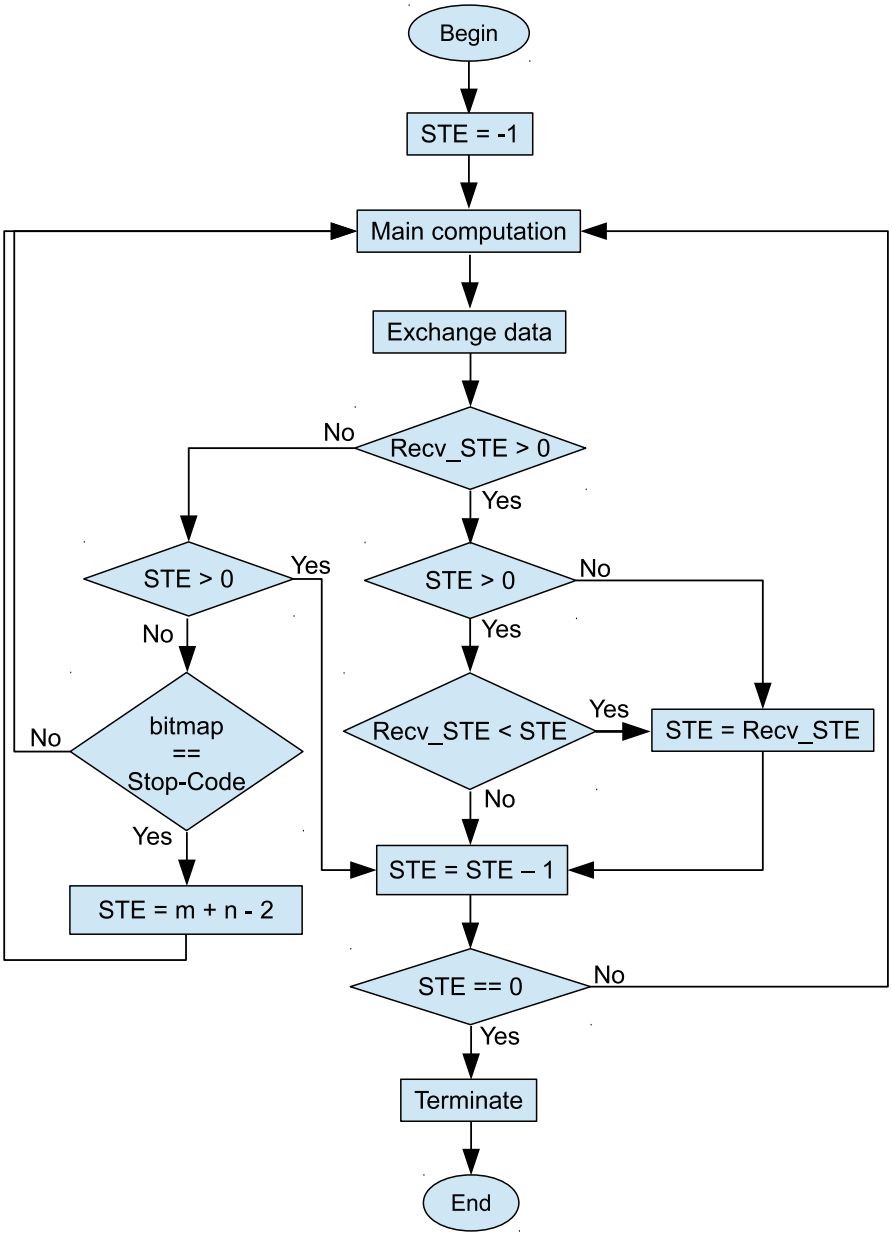


Figure 3.3 Synchronous Termination algorithm. Recv_STE is the STE received from a neighbour

In this flowchart, a STE is used to keep track of whether a process can actually terminate. STEs are initially set to (-1) in all processes. Once the termination

is detected on a process, its STE is set to a positive integer which is decreased by one after each iteration or message exchange. The process will terminate when its STE reaches zero.

In a distributed system described by the model in Section 3.2, there are two types of message. The first type is basic message which is produced by the underlying computation. The second type is control message which is produced by a DTD algorithm. Generally, basic messages and control messages are packed and handled separately. However, as a control message consists of a bitmap and a STE (Fig. 3.4), its size is small. Small messages are expensive due to protocol overhead so that, in Bitmap DTD algorithm, basic message and DTD message are aggregated into a single message as seen in Fig. 3.5. This technique, namely message combining, is commonly used in DTD implementations to reduce the communication overhead (Mattern, 1989).

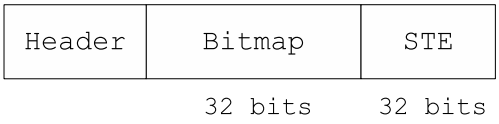


Figure 3.4 Control message format in a system of 32 processes

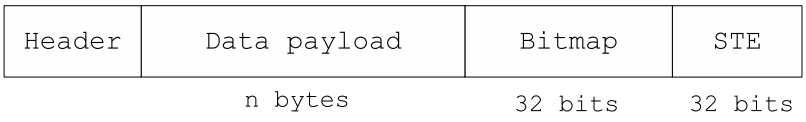


Figure 3.5 Combined message format in a system of 32 processes

3.3.2 Proof of correctness

Lemma 1: Given that a distributed system described in Section 3.2 has entered a quiescent state, the Bitmap Termination Detection algorithm will detect this state after a finite number of iterations.

Proof: As the whole system is a connected graph, there always exists at least one path from any two processes. By unidirectional communication, information from one process will be spread to all other processes within a finite number of steps. In the case that all processes have become passive, their corresponding bits in the bitmap are set to 1 and sent out to other processes. After a finite number of iteration, some process will gather all bits 1 and obtain the **Stop-Code**, that process will detect the termination. Proof ended.

To prove the correctness of Synchronous Termination algorithm, let consider the following prerequisites, which are taken from the algorithm itself in Fig. 3.3. Let

D be the diameter of a graph, i.e. the greatest distance between any pair of vertices.

P 1 *When a process detects the termination, its STE is set to D and this process is called trigger process.*

P 2 *When a process receives a non-negative STE, if its STE is negative or if its STE is bigger than the received STE, its STE will be replaced by the newly received STE.*

P 3 *If STE of a process is positive, that STE will be decreased by* 1 *after each iteration. When STE of a process is equal to* 0*, the process terminates.*

At this point we have to prove the following lemma

Lemma 2: Given that a trigger process has detected termination, using the Synchronous Termination algorithm, all processes in the system will finally terminate exactly at the same step D.

Proof:

For the trigger process, according to P1 and P3, after termination is detected, STE is set to D and passed from one process to another and it is decreased by 1 per iteration. Obviously, after D iterations the trigger process will terminate.

To reach the farthest process, the STE of the trigger process has to go through D processes and after D iterations. As stated by P2 and P3, STE is decreased by one after passing from one process to another. So when the STE of the trigger process reaches the farthest process its value will be $D - D = 0$. Upon receiving this STE, the farthest process terminates and this happens exactly after D iterations from the detection of termination.

For an intermediate process, let assume its shortest path to the trigger process is K. Then after K iterations, it will receive the STE equal to $(D - K)$. After that, following P3 this STE will be decreased by one after each iteration and the process will terminate after $(D - K)$ iterations. In summary, this process terminates after $(D - K) + K = D$ iterations from the detection of termination.

There may be cases where two or more processes detect the termination at different iterations and become trigger processes. Then, in an arbitrary trigger process, according to P1, the STE is set to D, which means the process has to execute D iterations before termination. During this period, if a message with lower STE arrives, the STE of current process will be replaced by the new lower STE, according to P2. By this way, the earliest trigger process will win out all other later trigger processes and they will terminate altogether after D step since the earliest trigger process detected the termination. Proof ended.

Table 3.1 Comparison of various termination detection algorithms. M: the number of basic messages, n: the number of processes, E: the number of links, k: the number of failed processes.

Algorithm	Message Complexity	Detection Latency	Symmetry
Mattern (1987)	$O(M)$	$O(1)$	N
Venkatesan (1989)	$O(M+E)$	N/A	N
Huang (1989)	$O(M)$	$O(D)$	N
Tseng (1995)	$O(M+kn+n)$	$O(k+1)$	N
Present algorithm	$O(M)$	$O(D/2)$	Y

3.4 Performance analysis

3.4.1 Termination detection delay

Since a distributed system is considered as a connected graph with diameter D, the number of steps required for a bitmap of one process to spread throughout the system is D. In the worst case, when all processes become passive simultaneously, some intermediate processes will need about $D/2$ steps to gather all bitmaps and detect the termination. Thus, the detection delay is $D/2$.

3.4.2 Message complexity

Chandy and Misra (1986) showed that, in a worst case, lower bound of the message complexity of any DTD in asynchronous network is $O(M)$, where M is the number of basic messages. In Chandrasekaran and Venkatesan (1990), a lower bound $O(E)$ was proved for asynchronous DTD algorithms and $O(M+E)$ for general DTD algorithms, where E is the number of links in the system.

In Bitmap DTD algorithm, for an iteration, a process needs to send as much control messages as basic messages to its neighbours. In total, the number of control messages is equal to the number of basic messages. In other words, the message complexity of the algorithm is $O(M)$

A comparison of termination detection delay and message complexity by various credit/recovery DTD algorithms is provided in Table 3.1. It can be seen that, the present Bitmap DTD algorithm has optimal message complexity and a very good detection latency. This table also shows that the present algorithm is symmetric while others are not. Symmetric algorithms are easy to implement and are more efficient as they perform identically on all processes and do not require a spanning tree to be constructed before hand.

3.5 Conclusion

In this chapter, a new DTD algorithm, namely Bitmap DTD, is introduced. The algorithm evolves from the Credit/Recovery class, but with the credit being replaced by a bitmap. The proposed algorithm has following advantages in comparison with a typical Credit/Recovery DTD

- it is an symmetric algorithm which allows any process to detect termination;
- it is a decentralised algorithm which does not require any central control agent as well as rooted tree or graph;
- the termination detection delay is low ($D/2$);
- the message complexity of the algorithm is optimal $O(M)$.

It is noted that the Bitmap DTD method is implemented to be used with DD method for solving fluid flow problems. Therefore, the communication topology is required to be synchronous and message delivery is guaranteed.

Chapter 4

Compact local IRBF based parallel domain decomposition method for the solution of PDEs

In this chapter, a parallel method based on domain decomposition (DD) technique and compact local integrated radial basis function (CLIRBF) is presented (Pham-Sy et al., 2013). The overlapping DD method is used to separate the problem under consideration into smaller, separated problems with actual and artificial boundaries (ABs). The conditions on these ABs are guessed initially and may attain the actual solution if the iterative algorithm converges. Within each iterative step, these sub-problems are independent and solved in parallel by a CLIRBF scheme. The present algorithm, namely parallel CLIRBF, achieves higher throughput in solving large scale problems by, firstly, parallel processing of sub-problems and, secondly, accelerating the convergence rate using high order compact local scheme in each sub-problem. The procedure is illustrated with several numerical examples, including the lid-driven cavity fluid flow problem using Message Passing Interface supported by MATLAB.

4.1 Introduction

Radial Basis Functions (RBFs) have traditionally been used to provide a continuous interpolation of scattered data sets (Franke, 1982; Kansa, 1990a). The differential RBF (DRBF) based methods have been successfully used to solve a wide variety of differential equations. For this approach, once the field variables are known, its derivatives can be calculated through differentiation (Kansa, 1990a; Zerroukat et al., 1998; Tran-Canh and Tran-Cong, 2004). Another approach, namely the integrated RBF method (IRBF), which was proposed by Mai-Duy and Tran-Cong (2001), is based on the approximation of the highest-order derivatives of the differential equations using RBF at the first step, and subsequently its lower-order derivatives and the dependent variable itself are obtained by integration. The IRBF based methods can outperform other approximation methods

based on the DRBF owing to its ability to produce very accurate solutions using relatively small number of data nodes.

Although full-domain IRBF methods are highly flexible and exhibit high order convergence rates in their basic implementation, the associated fully-populated matrix systems can lead to poor numerical conditioning as the scale of a problem increases (Mai-Duy et al., 2008). The problem becomes critical with increasingly large data sets. Many techniques have been developed to reduce the effect of the problem, including DD methods (Ingber et al., 2004; Tran et al., 2009), adaptive selection of data centres (Ling et al., 2006), RBF preconditioners (Brown, 2005) and RBF based compact local stencil methods (Mai-Duy et al., 2011). While a reliable method of controlling numerical ill-conditioning and particularly computational cost, as problem scale increases, can be based on DD method, the use of compact local approximations facilitates the solution of a differential equation without having to deal with large systems of global equations. In this chapter, a parallel algorithm based on compact local integrated RBF and DD techniques is developed for the solution of boundary value problems (BVP). A large problem is firstly decomposed into many smaller problems, each of which is analysed in parallel, and secondly the acceleration of the convergence rate within each subregion using groups of CLIRBF stencils is carried out by parallel CPUs. For the ease of presentation, in this chapter the terms sub-domain and process are used interchangeable.

One common problem associated with distributed computing is asymmetric termination of processes over the system. This problem occurs more frequently when dealing with asymmetric problems, in which one process may converge faster than the others. In fact, this situation was first noticed by Francez (1980) and independently by Dijkstra and Scholten (1980), and named Distributed Termination Detection (DTD). In general, the goal of DTD is to detect whether a system is in its quiescent state. Quiescence is defined as a state, in which no process is active and no message is in transmission. As the system is distributed, there is no shared clock or memory involved. Moreover, if the system is required to be synchronous, the existed problem becomes more complex. In our parallel approach, DD method is used to divide the computational work among distributed processes, or CPUs. Because the results obtained from each process must be consistent within the whole domain, all processes must stop at the same step. In this chapter, the Bitmap DTD algorithm presented in Chapter 3 is used owing to some important advantages (i) it allows any process to detect termination (symmetry); (ii) it does not require any central control agent (decentralisation); and (iii) the message complexity of the proposed algorithm is nearly optimal.

This chapter is organised as follows. In Section 4.2, a brief review of a CLIRBF method is presented. The DD method as well as DTD are described in Section 4.3 and Section 4.4, respectively. Numerical examples are then discussed in Section 4.5 with a conclusion in Section 4.6.

4.2 Review of the IRBF collocation method

Consider a second-order ODE with boundary condition as follows.

$$\mathcal{L}u = f, \quad x \in \Omega \tag{4.1}$$
$$\mathcal{B}u = g, \quad x \in \partial\Omega \tag{4.2}$$

where $\mathcal{L}$ is a second order differential operator; $\mathcal{B}$ - an operator imposed as boundary conditions such as Dirichlet, Neumann or a mixture of both; u - an unknown function; f and g - given functions; Ω and $\partial\Omega$ - the domain under consideration and its boundary. For simplicity, the 1D-IRBF scheme for discretisation of ODEs is presented.

4.2.1 1D-IRBF collocation method

The function u along an x-gridline is represented in the IRBF form (Mai-Duy and Tran-Cong, 2001) as

$$\frac{d^2u}{dx^2} = \sum_{i=1}^{n} w_i g_i = \sum_{i=1}^{n} w_i G_i^{[2]}, \tag{4.3}$$

where $\{w_i\}_{i=1}^n$ is the set of RBF weights; and $\{g_i(x)\}_{i=1}^n$ the set of RBFs. In this work, the Multi-quadric (MQ) RBF is used and given by (Haykin, 1999)

$$G_i(x) = \sqrt{(x - c_i)^2 + a_i^2},$$

where $\{c_i\}_{i=1}^n$ is a set of centres and $\{a_i\}_{i=1}^n$ a set of MQ-RBF widths.

The corresponding first-order derivative and function are then determined through integration as follows.

$$\frac{du}{dx} = \sum_{i=1}^{n} w_i G_i^{[1]} + C_1, \tag{4.4}$$

$$u = \sum_{i=1}^{n} w_i G_i^{[0]} + C_1 x + C_2, \tag{4.5}$$

where $G_i^{[1]}(x) = \int G_i^{[2]}(x)dx$, $G_i^{[0]}(x) = \int G_i^{[1]}(x)dx$ and C_1 and C_2 are unknown constants of integration. The superscript [.] is used to indicate the associated derivative order.

Collocating equations (4.3) - (4.5) at grid points $\{x_i\}_{i=1}^n$ yields the following set of algebraic equations

$$\frac{d^2\widetilde{\mathbf{u}}}{dx^2} = \mathcal{G}^{[2]}\widetilde{\mathbf{w}}, \tag{4.6}$$

$$\frac{d\widetilde{\mathbf{u}}}{dx} = \mathcal{G}^{[1]}\widetilde{\mathbf{w}}, \tag{4.7}$$

$$\widetilde{\mathbf{u}} = \mathcal{G}^{[0]}\widetilde{\mathbf{w}}, \tag{4.8}$$

where

$$\mathcal{G}^{[2]} = \begin{bmatrix} G_1^{[2]}(x_1) & G_2^{[2]}(x_1) & \cdots & G_n^{[2]}(x_1) & 0 & 0 \\ G_1^{[2]}(x_2) & G_2^{[2]}(x_2) & \cdots & G_n^{[2]}(x_2) & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ G_1^{[2]}(x_n) & G_2^{[2]}(x_n) & \cdots & G_n^{[2]}(x_n) & 0 & 0 \end{bmatrix},$$

$$\mathcal{G}^{[1]} = \begin{bmatrix} G_1^{[1]}(x_1) & G_2^{[1]}(x_1) & \cdots & G_n^{[1]}(x_1) & 1 & 0 \\ G_1^{[1]}(x_2) & G_2^{[1]}(x_2) & \cdots & G_n^{[1]}(x_2) & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ G_1^{[1]}(x_n) & G_2^{[1]}(x_n) & \cdots & G_n^{[1]}(x_n) & 1 & 0 \end{bmatrix},$$

$$\mathcal{G}^{[0]} = \begin{bmatrix} G_1^{[0]}(x_1) & G_2^{[0]}(x_1) & \cdots & G_n^{[0]}(x_1) & x_1 & 1 \\ G_1^{[0]}(x_2) & G_2^{[0]}(x_2) & \cdots & G_n^{[0]}(x_2) & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ G_1^{[0]}(x_n) & G_2^{[0]}(x_n) & \cdots & G_n^{[0]}(x_n) & x_n & 1 \end{bmatrix},$$

$$\begin{aligned} \widetilde{\mathbf{w}} &= (w_1, w_2, \cdots, w_n, C_1, C_2)^T, \\ \widetilde{\mathbf{u}} &= (u_1, u_2, \cdots, u_n)^T, \\ \frac{d^k \widetilde{\mathbf{u}}}{dx^k} &= \left(\frac{d^k u_1}{dx^k}, \frac{d^k u_2}{dx^k}, \cdots, \frac{d^k u_n}{dx^k} \right)^T, \end{aligned}$$

where $u_i = u(x_i)$ with $i = 1, 2, \cdots, n$.

Because of the presence of integration constants of the IRBF based approximation, the system of equations (4.8) will become under-determined. In this case, one can beneficially introduce in the algebraic equation system additional constraints such as nodal or derivative values. Thus, the algebraic equation system (4.8) can be reformulated as follows.

$$\begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{h}} \end{pmatrix} = \begin{bmatrix} \mathcal{G}^{[0]} \\ \widetilde{L} \end{bmatrix} \widetilde{\mathbf{w}} = \mathcal{C} \widetilde{\mathbf{w}}, \tag{4.9}$$

where $\widetilde{\mathbf{h}} = \widetilde{L} \widetilde{\mathbf{w}}$ are additional constraints. The conversion of the network-weight space into the physical space yields

$$\widetilde{\mathbf{w}} = \mathcal{C}^{-1} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{h}} \end{pmatrix}, \tag{4.10}$$

where $\mathcal{C}^{-1}$ is the conversion matrix. By substituting equation (4.10) into equations (4.6) and (4.7), the second and first-order derivatives of u will be expressed in terms of nodal variable values as follows.

$$\begin{aligned} \frac{d^2 u}{d^2 x} &= \mathcal{D}_2 \tilde{\mathbf{u}} + k_2, \\ \frac{du}{dx} &= \mathcal{D}_1 \tilde{\mathbf{u}} + k_1, \end{aligned} \tag{4.11}$$

where $\mathcal{D}_1$ and $\mathcal{D}_2$ are known vectors of length n; and k_2 and k_1 scalars determined by $\widetilde{h}$. Applying (4.11) at each and every collocation point on the grid-line yields

$$\begin{aligned} \frac{d^2\tilde{\mathbf{u}}}{d^2x} &= \widetilde{\mathcal{D}}_2\tilde{\mathbf{u}} + \widetilde{\mathbf{k}}_2, \\ \frac{d\tilde{\mathbf{u}}}{dx} &= \widetilde{\mathcal{D}}_1\tilde{\mathbf{u}} + \widetilde{\mathbf{k}}_1, \end{aligned} \tag{4.12}$$

where $\widetilde{\mathcal{D}}_2$ and $\widetilde{\mathcal{D}}_1$ are known matrices of dimension $n \times n$; and $\widetilde{\mathbf{k}}_2$ and $\widetilde{\mathbf{k}}_1$ are known vectors of length n.

4.2.2 Compact local IRBF methods

The 1-DIRBF approximation, presented in Section 4.2.1, is now developed into compact local 3-point stencil to discretise differential equations, following Mai-Duy et al. (2011).

Consider local stencils $LS_i = [x_{i-1}, x_i, x_{i+1}]$ associated with grid point x_i ($2 \leq i \leq n-1$) in a typical global 1D Cartesian grid line. The same procedure of Eqs. (4.3)-(4.9) is applied for each and every LS_i.

In the context of the present 3-point local stencils, additional constraints are chosen to be the imposition of the governing equation at x_{i-1} and x_{i+1}. Thus, Eq. (4.9) includes (i) a set of three equations representing nodal values of u over the LS_i and (ii) a set of two algebraic equations obtained by evaluating the governing differential equation (4.1) at x_{i-1} and x_{i+1}. Following Eqs. (4.10) - (4.11), the governing differential equation (4.1) is discretised at x_i by a weighted combination of function values (u_{i-1}, u_i, u_{i+1}). Eventually, a tri-diagonal system matrix is obtained and can be solved by a proper iterative method.

4.3 Overlapping DD Method

Although the CLIRBF methods are expected to (i) enhance the computational accuracy and convergence rate, and (ii) reduce the computational time, they could suffer from the problem of numerical ill-condition for large scale problems. In order to deal with this issue, a combination of the present CLIRBF approach and DD method is used to solve differential equations.

DD method aims to split the domain under consideration into smaller sub-domains while guaranteeing the continuity at the splitting boundary. DD methods can be grouped into two classes: (i) non-overlapping methods and (ii) overlapping methods (Smith et al., 1996).

For the multiplicative Schwarz overlapping method, since the values on AB of each sub-domain are updated from the most recent results of its neighbouring sub-domains (NSs), the sub-domains have to be solved sequentially. Thus, it has

little potential for parallel programming. One way to make it parallelisable is by grouping sub-domains through black-white colouring technique (Quarteroni and Valli, 1999), so that all any two NSs have different colours. By this way, in the first half-step only one group, for example the white one, executes simultaneously because its sub-domains do not share any common boundary. In the next half-step, the other group (the black one) will execute in the same manner.

For the additive Schwarz overlapping method, the values on AB of each sub-domain at step $k+1$ are updated from the results of its NS's at step k. Since the data of each sub-domain are independent of others during the computational process, the approach is naturally parallel-capable and its implementation is quite straightforward. In this chapter, the additive overlapping method is implemented on the domain under consideration. We will present this method in more detail below.

4.3.1 Additive Schwarz overlapping method

For illustrative purposes, the domain Ω is divided into two sub-domains Ω_1 and Ω_2. Let $\partial\Omega_1$ and $\partial\Omega_2$ be the actual boundaries and Γ_1, Γ_2 ABs of Ω_1 and Ω_2 respectively (Fig. 4.1). The boundary condition imposed on this interface can be Dirichlet-Dirichlet, Dirichlet-Neumann or otherwise.

In this work, the boundary condition imposed on the ABs $\Gamma_{1,2}$ is Dirichlet-Dirichlet type. As the algorithm is iterative, Eq. (4.1) is written for the sub-domains Ω_1 and Ω_2 respectively at a step k as follows.

$$\left\{\begin{array}{rcll} \mathcal{L}u_1^k & = & f, & x \in \Omega_1 \\ \mathcal{B}u_1^k & = & g, & x \in \partial\Omega_1 \\ u_1^k & = & c_{1\Omega_2}^{k-1}, & x \in \Gamma_1 \end{array}\right. \tag{4.13}$$

and

$$\left\{\begin{array}{rcll} \mathcal{L}u_2^k & = & f, & x \in \Omega_2 \\ \mathcal{B}u_2^k & = & g, & x \in \partial\Omega_2 \\ u_2^k & = & c_{2\Omega_1}^{k-1}, & x \in \Gamma_2 \end{array}\right. \tag{4.14}$$

where $c_{i\Omega_j}^{k-1}$ $(i,j=\{1,2\},\, i \neq j)$ is extracted from Ω_j at the step $k-1$ and given by

$$c_{i\Omega_j}^{k-1} = \theta u_{i\Omega_j}^{k-2} + (1-\theta)u_{i\Omega_j}^{k-1}; \quad 0 \leq \theta \leq 1$$

where $u_{i\Omega_j}^{k-1}$ $(i,j=\{1,2\},\, i \neq j)$ is the value of u in Ω_i obtained from the solution in Ω_j at the step $k-1$ (see Fig. 4.1); θ relaxation factor which, in our case, is chosen to be equal to 1.

One important character of parallel DD methods is shared memory. In such systems all processes have access to a shared memory and each process has to update its own record in that memory. Because of the nature of shared memory as the number of processes increased, memory access becomes a bottleneck of the whole system. Another drawback of this approach is its high message complexity. Each process has to access the share memory at least once per iteration to update its state, which produces considerably large amount control messages in the network.

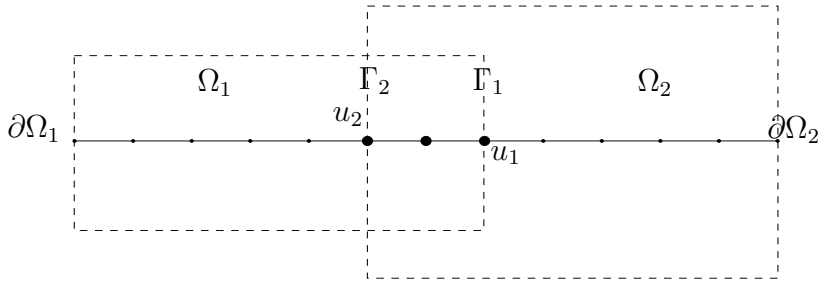


Figure 4.1 DD with two sub-domains Ω_1 and Ω_2 for a 1D problem

In order to get rid of these disadvantages of shared memory, in our method, a distributed memory is used. However, to keep all processes synchronised, each message between processes carries a bitmap which is a snapshot of the system at that moment. This idea will be discussed in Section 4.4.

4.3.2 Algorithm of the present procedure

The present method can now be described in a more detailed algorithm whose flowchart is shown in Fig. 4.2 and consists of four main steps

1. Divide the analysis domain into a number of sub-domains. Guess initial boundary condition on ABs;
2. Solve the boundary value problem in each and every sub-domain using CLIRBF method;
3. Exchange the field variables across ABs;
4. Check for the convergence on the overlapping area.

For the case of two sub-domains, it is quite easy to determine neighbour and corresponding AB of a sub-domain. However, with more than two sub-domains, more effort is required as follows.

1. Enumerate sub-domains;
2. Determine the neighbours;
3. Determine real and artificial boundaries.

An example of this procedure is provided in Fig. 4.3

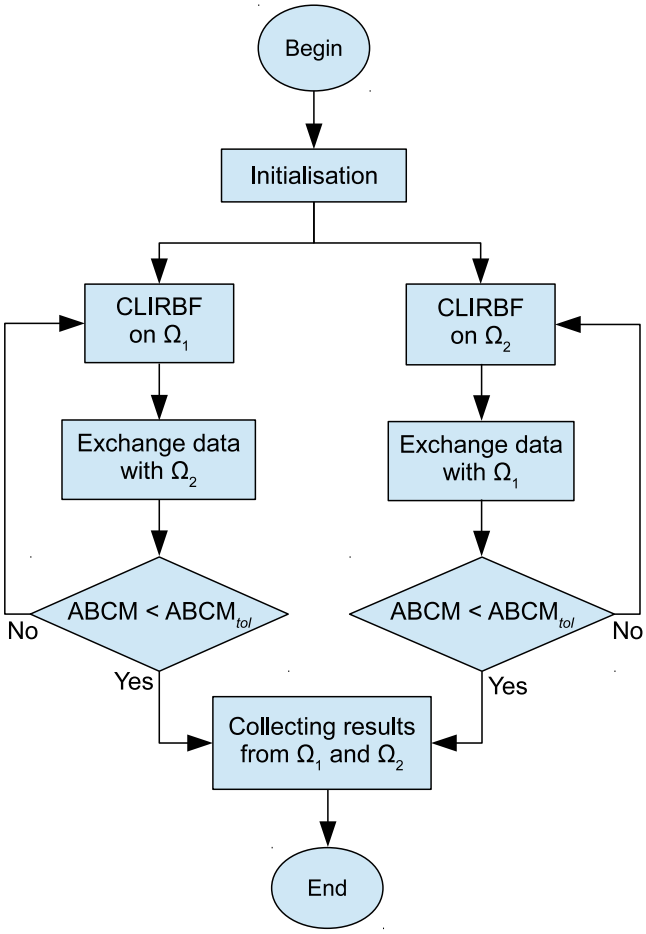


Figure 4.2 Parallel DD with 2 sub-domains. $ABCM$ - convergence measure on ABs; $ABCM_{tol}$ - predefined tolerance.

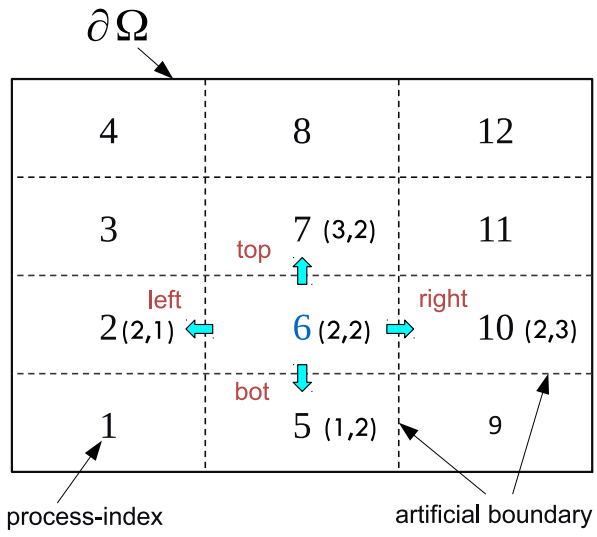


Figure 4.3 Enumeration in system within 4×3 sub-domains.

4.4 Parallel algorithm based on DTD

In a general distributed system, any process can terminate if it has finished its assigned task and the system is fully terminated only when all processes are terminated. However, when solving a PDE using DD method based parallel algorithm, as the result obtained from all sub-domains must be consistent, all processes are required to terminate strictly at the same step. Clearly, if one process terminates while some other processes are still active, those active processes cannot exchange information with their terminated neighbours. Consequently, the system will end up with communication errors. In the current parallel algorithm, a Bitmap DTD method presented in Chapter 3 is employed to synchronise the computation and detect the termination of the system.

During the DTD process, an extra amount of data called bitmap is spread throughout the system and reflects the current state of all processes. The use of bitmap makes the present algorithm symmetric because any process can now detect the termination at any time. Furthermore, the algorithm requires neither a central process nor a spanning tree/graph to monitor the state of the system.

Bitmap DTD algorithm consists of two components: (i) Termination Detection and (ii) Synchronised Termination. The purpose of Termination Detection is to detect whether the system is in quiescent state, i.e. all the processes are ready to terminate. When that state is achieved, the Synchronised Termination is activated to get all processes terminated simultaneously at the same step.

Bitmap DTD runs along with the main computation as the data and the bitmap are combined into one message. During Termination Detection, each process keeps a record of its current bitmap which is initially zero. When a process receives a bitmap from one of its neighbours it updates its own bitmap by doing a binary union between its bitmap and the received bitmap. At any stage, a process will detect termination if its bitmap consists of all bits 1. Otherwise, it continues the main computation. When the termination is detected, the Synchronous Termination is started. In this phase, a Step-To-End (STE), i.e the number of steps each process needs to perform the computation before terminating, is introduced. Initially, all STEs are set to (-1). Once the termination is detected by a process, its STE is set to a positive value of D which is the diameter of the graph of the distributed system. The STE and other data are then transferred to its neighbours. The neighbours, upon receiving a positive STE, will update their own STE by the received STE. A positive STE is decreased by one after each iteration. A process will terminate when its STE reaches zero. The mechanism of Bitmap DTD algorithm is presented in details in Chapter 3.

4.5 Numerical results

The proposed parallel method based on the combination of a CLIRBF method and parallel DD method is verified using 1D and 2D problems with different boundary conditions. The capability and efficiency of the present method are

then demonstrated with the simulation of the lid-driven cavity (LDC) flow of a viscous fluid.

4.5.1 One dimensional problem

Consider the following second-order ODE.

$$\frac{d^2u}{dx^2} + \frac{du}{dx} + u = -\exp(-5x)\left[9979\sin(100x) + 900\cos(100x)\right], \quad 0 \le x \le 1, \tag{4.15}$$

with an analytic solution $u = \exp(-5x)\sin(100x)$.

This problem is solved using the present method with two different types of boundary condition. The domain is partitioned into 2 sub-domains and a wide range of grids $(201, 303, \ldots, 501)$ is considered.

Dirichlet boundary condition

The Dirichlet conditions are $u(0) = 0$ and $u(1) = \sin(100)\exp(-5)$.

The results show that the present method achieves the same accuracy level as the CLIRBF method. In fact, the relative L_2 error of the present method is $O(h^\alpha)$, showing a convergence rate $\alpha = 4.12$ while that for the CLIRBF is 4.26 (see Fig. 4.4 (bottom figure)). Figure 4.4 (top figure) depicts a comparison of the results obtained by the present method, the CLIRBF and the analytic ones.

Dirichlet and Neumann boundary conditions

The Dirichlet condition is imposed on the left end $u(0) = 0$ and the Neumann conditions on the right end $\dfrac{du(1)}{dx} = 5\exp(-5)\left[20\cos(100) - \sin(100)\right] = 0.598$.

While the result described in Fig. 4.5 (top) by the present method is in very good agreement with the analytic solution, the convergence rate displayed in Fig. 4.5 (bottom) shows that the present parallel method yields a higher accuracy in comparison with the CLIRBF method. Generally, the 1D example shows that the present parallel scheme based on CLIRBF and DD method can attain the numerical accuracy of corresponding schemes using single domain.

4.5.2 Two dimensional problem

Consider the following 2D problem

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 4(1-\pi^2)\sin(2\pi x)\sinh(2y) + 16(1-\pi^2)\cosh(4x)\cos(4\pi y). \tag{4.16}$$

This problem is solved in the analysis domain and with the Dirichlet and Dirichlet/Neumann boundary conditions given in Fig. 4.6. A sample program code of

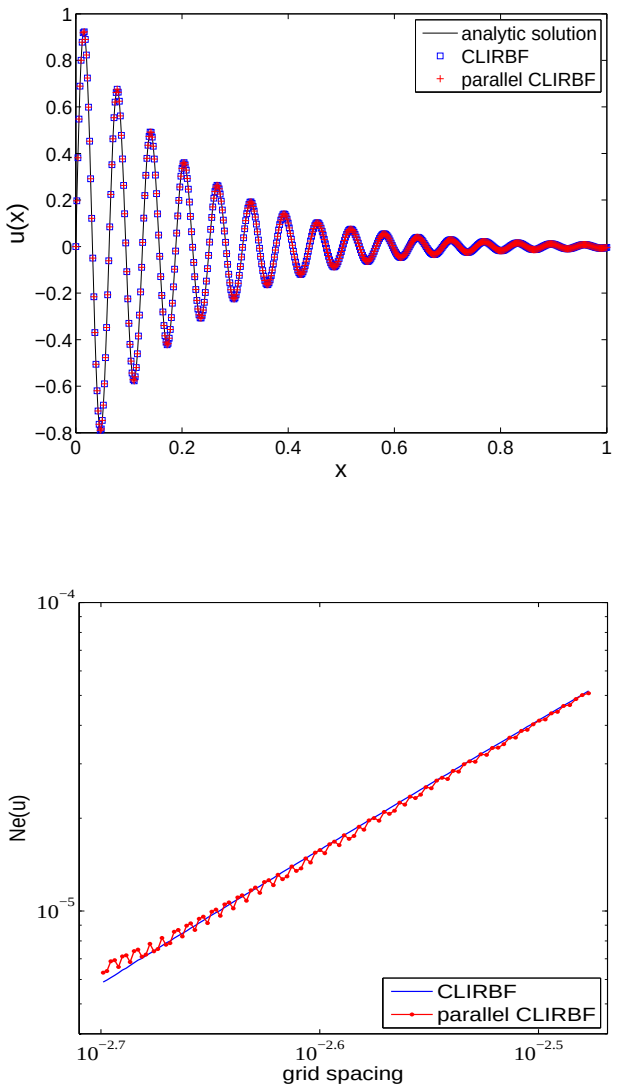


Figure 4.4 Second order problem with Dirichlet boundary condition. Solutions obtained by the 3-point CLIRBF method, the present parallel method and and the analytic solution (top figure); Relative L_2 errors of the solution u against the grid density by the 3-point CLIRBF method and the present method (bottom figure).

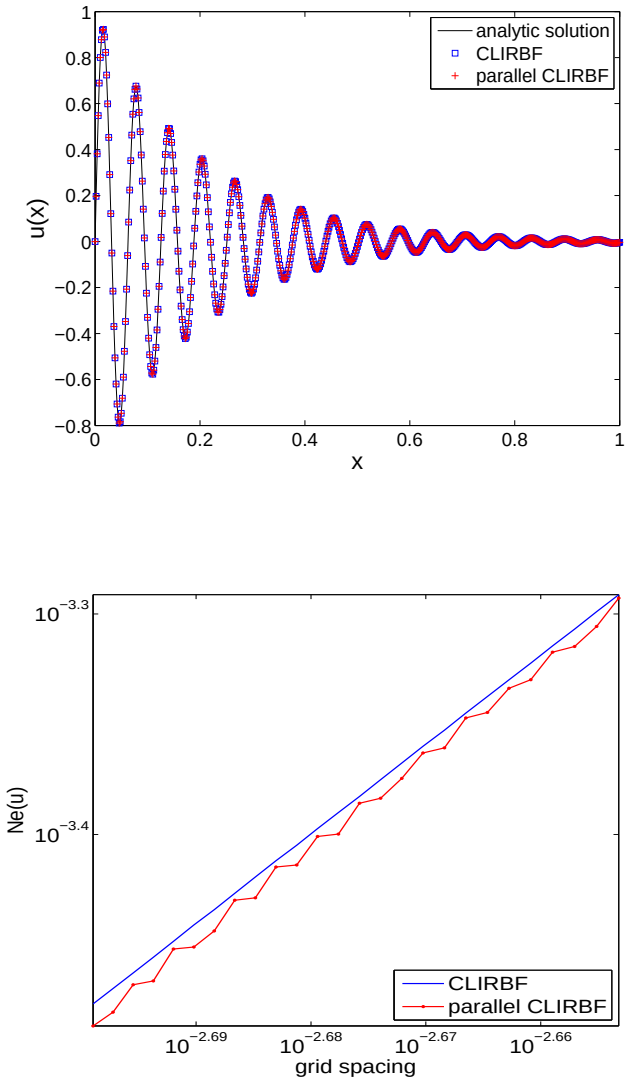


Figure 4.5 Second order problem with Dirichlet-Neumann boundary condition. Solutions obtained by the 3-point CLIRBF method, the present parallel method and and the analytic solution (top figure); Relative L_2 errors of the solution u against the grid density by the 3-point CLIRBF method and the present method (bottom figure).

the parallel algorithm for 2D problems with Dirichlet boundary conditions can be found in Appendix B.

The analytic solution is given by $u(x, y) = \sin(2\pi x)\sinh(2y) + \cosh(4x)\cos(4\pi y)$ and presented in Fig. 4.7(a).

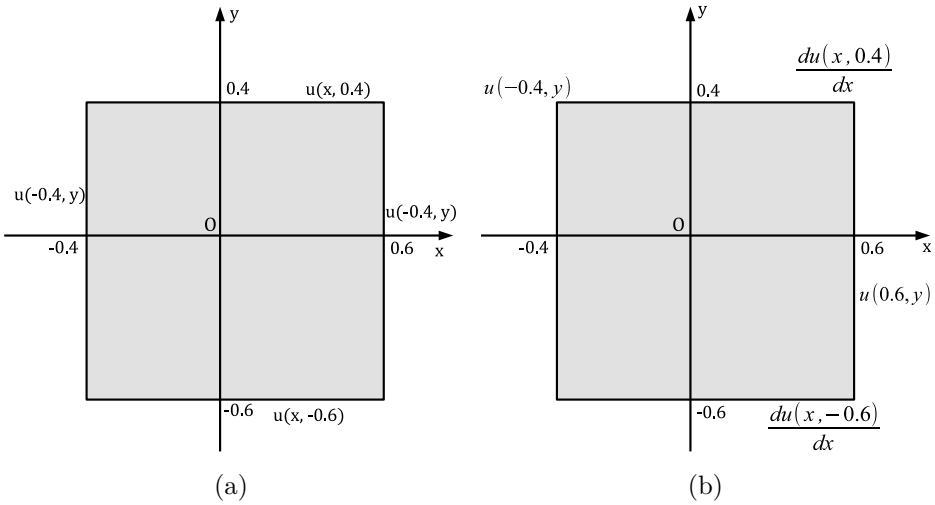
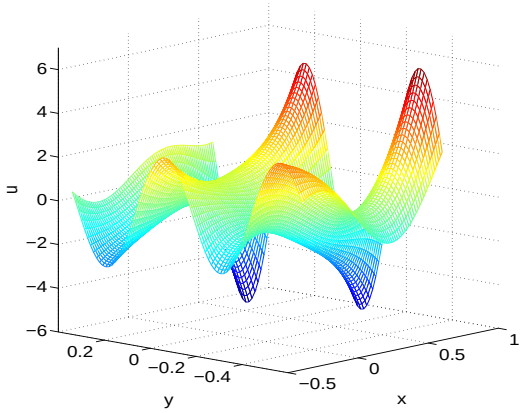


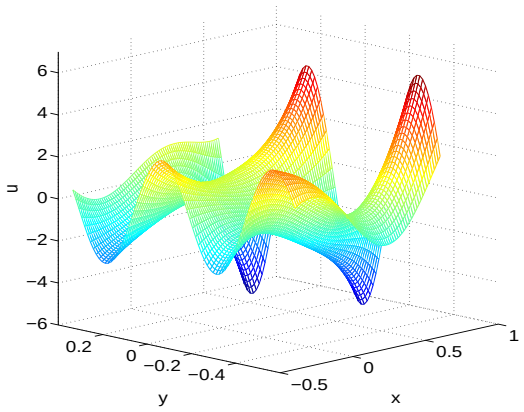
Figure 4.6 2D problem. Geometry of the analysis domain with Dirichlet boundary conditions (a) and Dirichlet-Neumann boundary conditions (b)

A range of uniform grids (71×71, 113×113, 149×149, 185×185, 237×237) and CPUs (4, 9, 16, 25) are considered in the simulation. The results show that the numerical solution by the present parallel method whose computational parameters are given in Tables 4.1 and 4.2 is in very good agreement with the analytic solution (see Fig. 4.7). Indeed, Fig. 4.7 presents the numerical solution of field variable u by the present parallel method with Dirichlet-Dirichlet boundary condition (Fig. 4.7(b)) and Dirichlet-Neumann boundary condition (Fig. 4.7(c)) using a grid of 77×77 collocation points and four CPUs in comparison with the analytic solution.

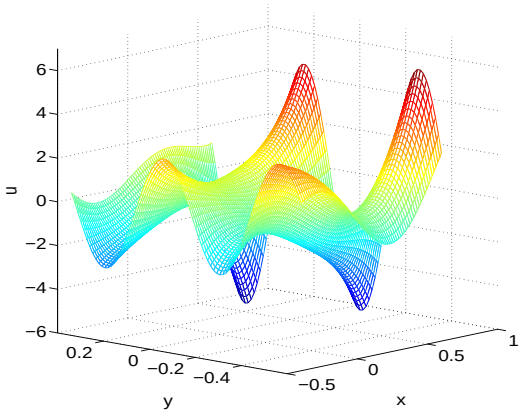
The results presented in Tables 4.1 and 4.2 show that when grid density increases the throughput of the present parallel method grows significantly. Indeed, the computation time is reduced significantly in comparison with non-parallel computation when increasing the number of collocation points to 185×185 as shown in Tables 4.1-4.2. Furthermore, the parallel algorithm is really efficient for solving large scale problems which require a large number of calculations as described in the next example.



(a)



(b)



(c)

Figure 4.7 2D problem. (a) - Analytic solution; (b) - Present method with Dirichlet-Dirichlet boundary condition; (c) - Present method with Dirichlet-Neumann boundary condition.

Table 4.1 2D problem with Dirichlet Boundary conditions. CPUs: number of CPUs; $ABCM_{tol}$: the tolerance of convergence measure on the artificial interfaces; N_i: number of iterations; N_{es}: error norm for the single domain CLIRBF method; N_{ep}: error norm for the parallel CLIRBF-DD method; T_s: sequential computation time (second); T_p: parallel computation time (second).

	Single Domain		Multi Domains - Parallel computation				
Grid	N_{es}	T_s	CPUs	$ABCM_{tol}$	N_i	N_{ep}	T_p
77×77	$2.963E-6$	25	4	$1.E-6$	124	$3.174E-6$	428
113×113	$1.882E-6$	104	9	$1.E-6$	217	$1.831E-6$	744
149×149	$1.443E-6$	383	16	$1.E-6$	384	$1.413E-6$	1311
185×185	$1.268E-6$	11234	25	$1.E-6$	515	$1.408E-6$	1739
237×237	n/a	n/a	25	$1.E-7$	577	$2.877E-6$	3873

Table 4.2 2D problem with Neumann and Dirichlet Boundary conditions. CPUs: number of CPUs; $ABCM_{tol}$: the tolerance of convergence measure on the artificial interfaces; N_i: number of iterations; N_{es}: error norm for the single domain CLIRBF method; N_{ep}: error norm for the parallel CLIRBF-DD method; T_s: sequential computation time (second); T_p: parallel computation time (second).

	Single Domain		Multi Domains - Parallel computation				
Grid	N_{es}	T_s	CPUs	$ABCM_{tol}$	N_i	N_{ep}	T_p
77×77	$1.523E-6$	19	4	$1.E-5$	132	$1.356E-6$	469
113×113	$5.898E-6$	54	9	$1.E-6$	660	$3.949E-6$	2294
149×149	$2.708E-6$	145	16	$5.E-7$	1150	$1.945E-6$	4039
185×185	$1.443E-6$	16560	25	$5.E-7$	921	$1.129E-6$	3193
237×237	n/a	n/a	25	$1.E-7$	661	$1.307E-6$	4379

4.5.3 Lid-driven cavity fluid flow problem

The LDC flow is commonly used as a typical example to benchmark numerical methods and therefore is also employed here to investigate the accuracy as well as the efficiency of the present parallel scheme. The problem is defined in the stream-function and vorticity formulation as follows.

$$\frac{\partial \omega}{\partial t} + (\frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y}) = \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right), \tag{4.17}$$

$$-\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}, \tag{4.18}$$

where ψ is the stream function, ω the vorticity, Re the Reynolds number defined as $Re = \frac{UL}{\nu}$, where U is the characteristic speed of the lid; L the characteristic side of the cavity and ν the kinematic viscosity of the fluid. The geometry of the analysis domain with the chosen coordinate system is shown in Fig. 4.8. The velocity components by x and y-direction u and v are given by $u = \frac{\partial \psi}{\partial y}$ and $v = -\frac{\partial \psi}{\partial x}$.

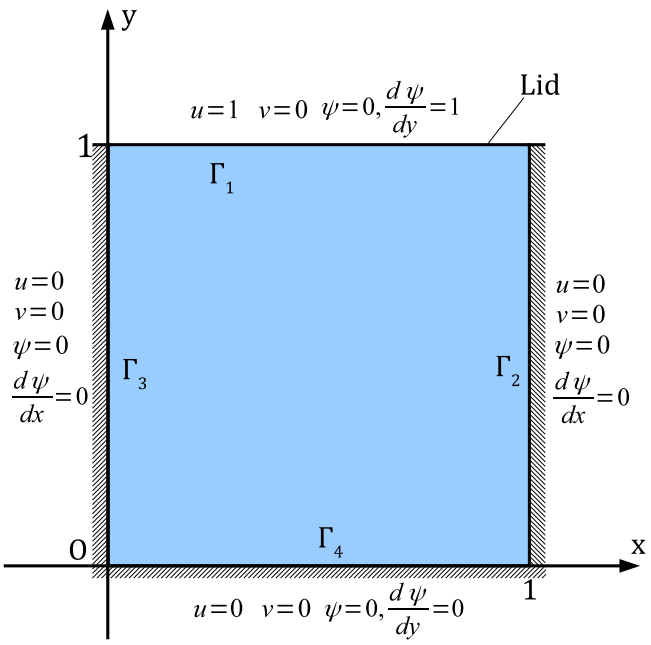


Figure 4.8 The LDC fluid flow problem. Geometry of the analysis domain and boundary conditions in terms of the stream function.

The boundary conditions are given in terms of the stream function as

$$\psi = 0, \quad \frac{\partial \psi}{\partial x} = 0 \quad \forall (x, y) \in \Gamma_2 \cup \Gamma_3; \tag{4.19}$$

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 0 \quad \forall (x, y) \in \Gamma_4; \tag{4.20}$$

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 1 \quad \forall (x, y) \in \Gamma_1. \tag{4.21}$$

The problem is solved by the present parallel algorithm using the 9-point 2D CLIRBF - DD method. Since each sub-domain is handled by a separate CPU, the number of the sub-domain is also the number of CPUs used for the present parallel method. After dividing the analysis domain into sub-domains, the algorithm for solving LDC problem in each sub-domain can be described as follows.

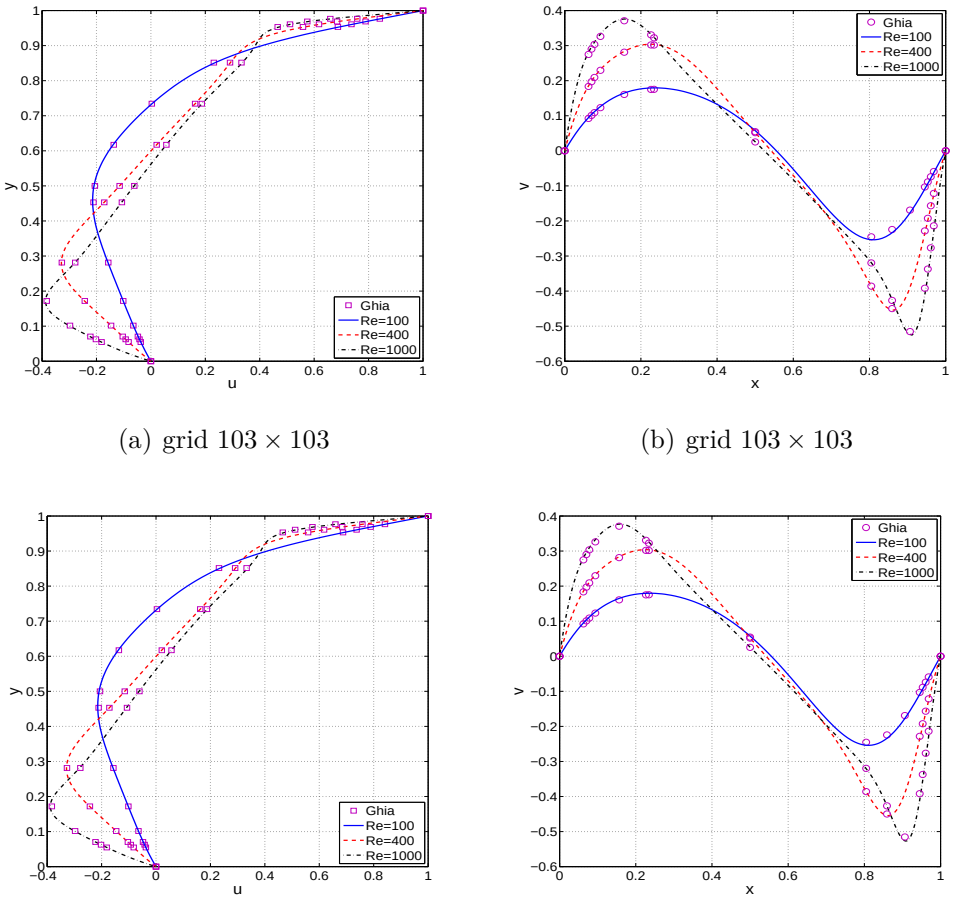
1. Guess initial boundary condition at ABs and initial values of ω;
2. Solve the LDC problem using CLIRBF method;
   (a) Solve Eq. (4.18) for ψ;
   (b) Approximate the values of ω on boundaries and the convective terms;
   (c) Solve Eq. (4.17) for ω;
3. Exchange the values of ψ and ω at interfaces with neighbours;
4. Calculate convergence measure in overlapping areas and check for termination condition;

   If the procedure is not yet converged, replace values of ψ and ω at ABs with those received from neighbours and go back to step 2;
5. Stop the procedure.

A range of $Re = \{100, 400, 1000\}$ and uniform grids are considered in the simulation. The time step Δt is chosen in the range from 10^{-4} to 10^{-6} and based on Re value and spatial grid size. Smaller time step is required for finer grids and/or higher Re. The detailed results by the present parallel method are provided in Tables 4.3 - 4.5.

Fig. 4.9 depicts profiles of the velocities u and v along the vertical and horizontal centre-lines, respectively for several Reynolds numbers with grids of 113×113 collocation points (Figs 4.9(a) - 4.9(b)) and 295×295 collocation points (Figs 4.9(c) - 4.9(d)) using the present parallel method with 25 CPUs. Results from the benchmark solution by Ghia et al. (1982) using a grid of 129×129 points are also included for comparison.

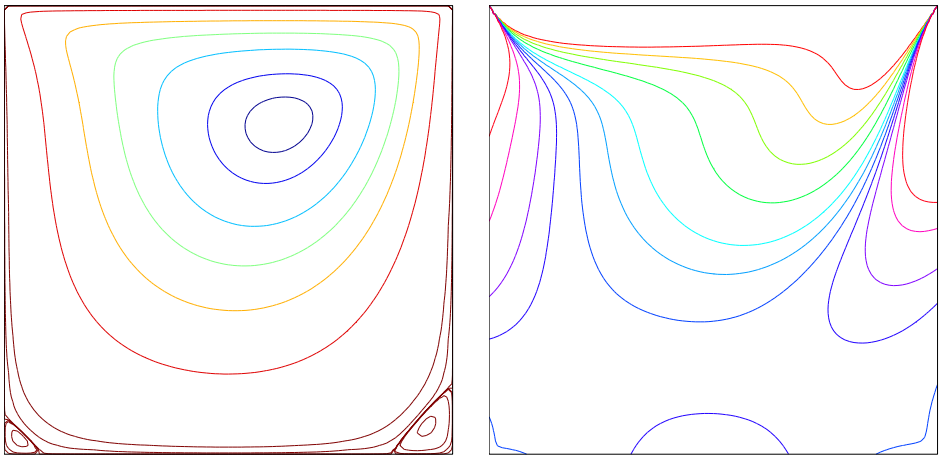
In Fig. 4.10, the streamlines and vorticity contours of the flow are shown to be in a very good agreement with benchmark solution by Ghia et al. (1982) and Botella and Peyret (1998). Furthermore, secondary vortices at the bottom corners (Figs. 4.10(a), 4.10(c) and 4.10(e) using 103×103 grid points and Figs. 4.11(a), 4.11(c) and 4.11(e) using 295×295 grid points) are well captured by the present method.



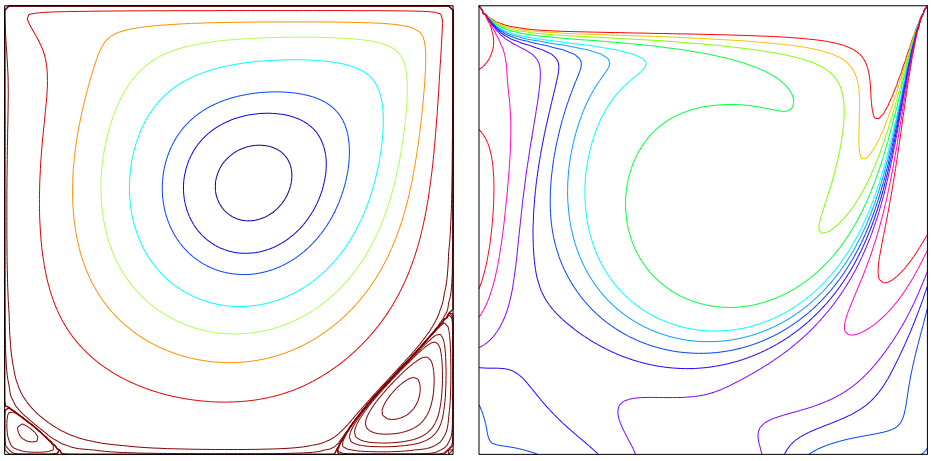
(a) grid 103×103 (b) grid 103×103

(c) grid 295×295 (d) grid 295×295

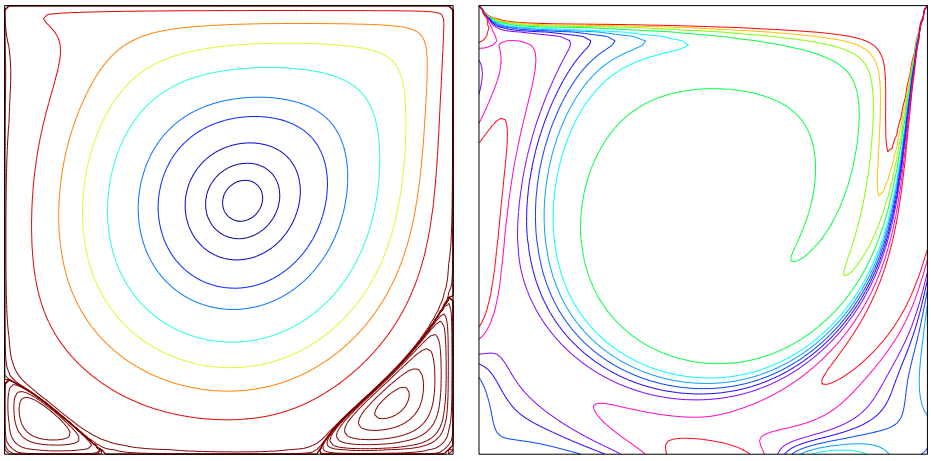
Figure 4.9 The LDC fluid flow problem. Profiles of the u velocity along the vertical centre lines (a, c) and the v velocity along the horizontal centre lines (b, d) by the present parallel method with several Reynolds numbers $Re = \{100, 400, 1000\}$ and two grids 103×103 (a, b) and 295×295 (c, d) in comparison with the corresponding Ghia's results



(a) $Re = 100$ (b) $Re = 100$

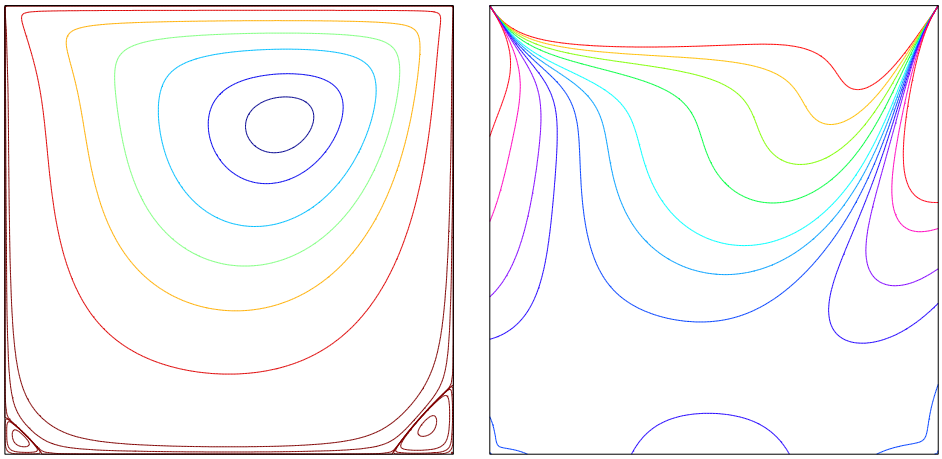


(c) $Re = 400$ (d) $Re = 400$

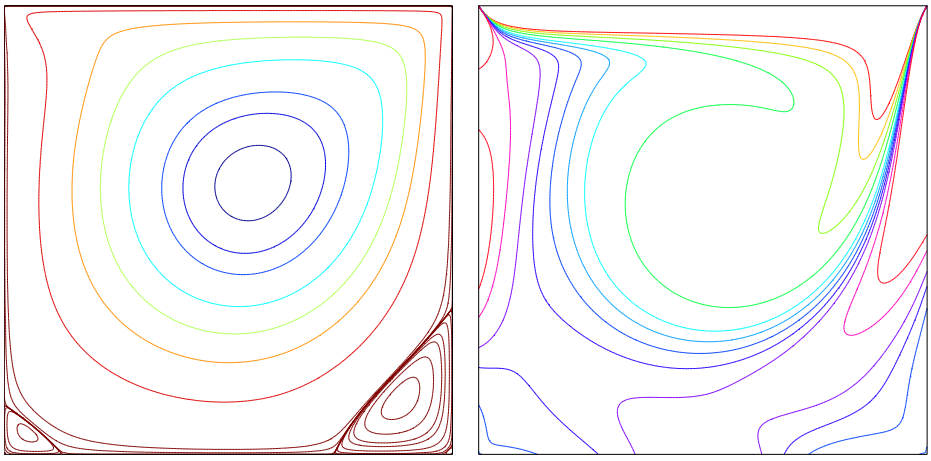


(e) $Re = 1000$ (f) $Re = 1000$

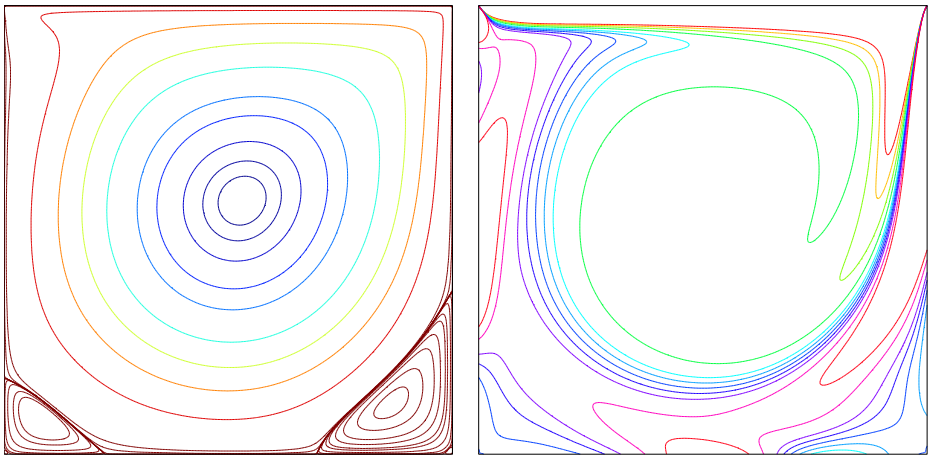
Figure 4.10 The LDC fluid flow problem. Streamlines (left figures) and vorticity contour (right figures) of the flow for several Reynolds numbers $Re = \{100, 400, 1000\}$ by the present parallel method using a grid of 103×103 of points.



(a) $Re = 100$ (b) $Re = 100$



(c) $Re = 400$ (d) $Re = 400$

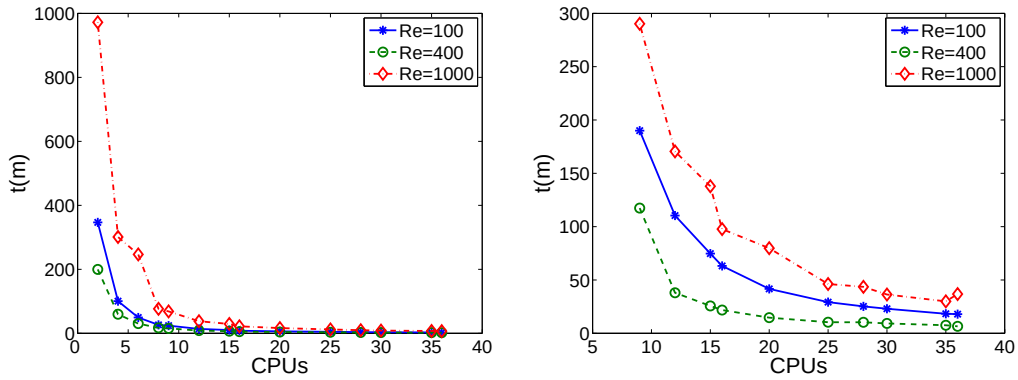


(e) $Re = 1000$ (f) $Re = 1000$

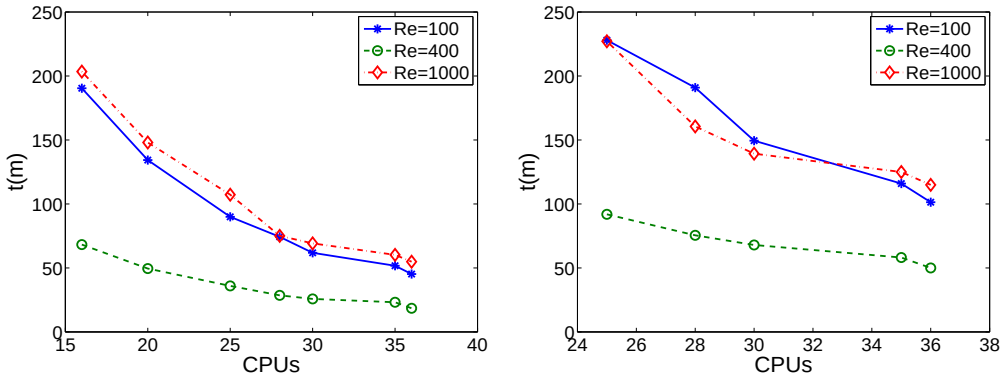
Figure 4.11 The LDC fluid flow problem. Streamlines (left figures) and vorticity contours (right figures) of the flow for several Reynolds numbers $Re = \{100, 400, 1000\}$ using a grid of 295×295 points.

The efficiency of the present parallel method is considered. The general goals are (i) to assign each process an equal amount of work by dividing the original domain into equal size sub-domains and (ii) to reduce the communication between processes by minimising the surface area of the sub-domains. As can be seen in Tables 4.3 - 4.5 the throughput of the present parallel method has been greatly improved.

In Fig. 4.12, the results show that the simulation time of the present parallel algorithm decreases tremendously as the number of CPUs is increased. This reflects the scalability the algorithm. However, there are always some thresholds, over which the increase of number of CPUs influences insignificantly on the simulation time. For example, the improvement of time efficiency of computation is not significant anymore as the number of CPUs is more than 10 for a grid of 101×101 (Fig. 4.12(a)), 25 for a grid of 155×151 (Fig. 4.12(b)) and $32 - 35$ for a grid of 209×209 (Fig. 4.12(c)) and 253×253 (Fig. 4.12(d)).



(a) grid 101×101 (b) grid 155×155



(c) grid 209×209 (d) grid 253×253

Figure 4.12 The LDC fluid flow problem. Simulation time versus number of CPUs for the present parallel method with different grids.

Furthermore, the parallel method removes the ill-conditioning of the global system matrix evidenced by a significant decrease of the condition number of the system matrix associated with the IRBF method (see the column 3 of Table 4.6 and Fig. 4.13 (bottom figure)).

Fig. 4.14 depicts the influence of the grid density on the time efficiency of the

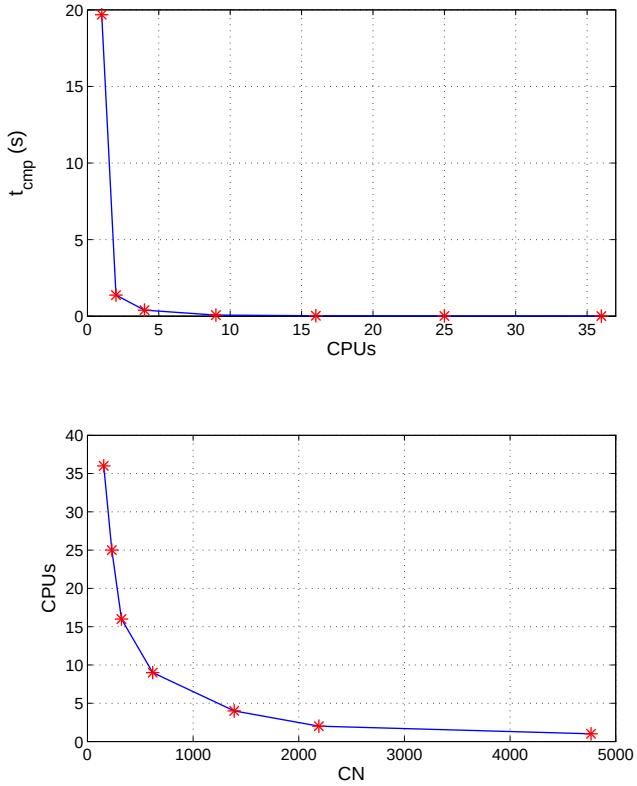


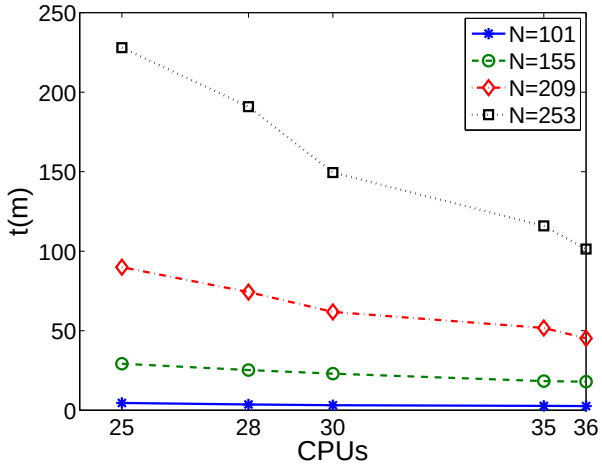
Figure 4.13 The LDC fluid flow problem with $Re = 100$, grid 101×101. Top figure: T_{cmp} - average computation time in one iteration. Bottom figure: CN - the condition number of the system matrix for solving ψ in each sub-domain.

present parallel algorithm. In this figure, it is observed that with higher grid size the simulation time decreases faster for all considered Reynolds numbers. The results confirm that the time efficiency of parallel methods will be higher for larger numbers of collocation points, i.e., larger scale problems.

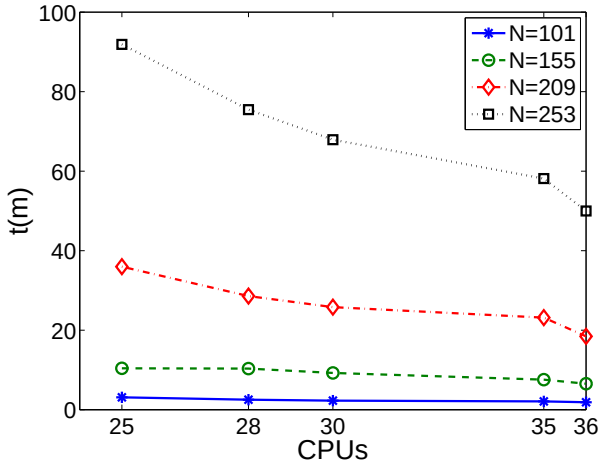
It is worth noting that the throughput of the present parallel method is very high when solving the LDC fluid flow problem. A reason for this is the reduction of the computation time within each sub-domain. As can be seen in Fig. 4.13 (top figure), the average computation time in a sub-domain decreases exponentially with respect to number of CPUs.

Indeed, although the communication time among processes is generally order of magnitude slower than calculations within a process, due to a great number of iterations, the computation time for each sub-domain plays a key role in improving the throughput of the present parallel method. One typical numerical example is given in Table 4.6 and the extensive improvement of throughput of the present parallel method can be explained as follows.

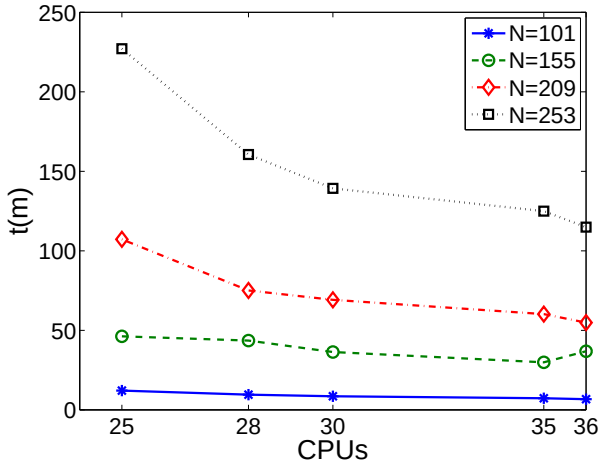
Let N_i be the number of iterations in the whole domain, T_{cmp} the average computational time (seconds) in sub-domains for each iteration; T_{cmm} the total communication time (minutes). The total parallel computation time T_p (minutes) is



(a) $Re = 100$



(b) $Re = 400$



(c) $Re = 1000$

Figure 4.14 The LDC fluid flow problem. Simulation time versus number of CPUs when solving LDC using the present parallel method with different Reynolds numbers: $Re = 100$ (Fig. 4.14(a)); $Re = 400$ (Fig. 4.14(b)); $Re = 1000$ (Fig. 4.14(c)).

given by

$$T_p = \frac{N_i \times T_{cmp}}{60} + T_{cmm} \tag{4.22}$$

Table 4.6 shows that while the number of iterations for different cases of parallel computation is similar (approximately 3000), the average computation time T_{cmp} for each iteration in sub-domains decreases quickly with respect to the problem size in each sub-domain (Figure 4.13 (top)), resulting in a very high rate of throughput.

Also, it is worth adding that as a result of DD, the problem size in a sub-domain is smaller. Consequently, the complexity of the problem is greatly reduced. To be more specific, the number of floating point operations (flops) required to do a singular value decomposition (SVD) for a matrix $(m \times n)$ is of $\mathrm{O}(mn^2)$. If the size of the matrix is reduced to a half, i.e. $(m \times \frac{n}{2})$, the complexity of the SVD operation becomes $\mathrm{O}(m\frac{n^2}{4})$, which is a fourth of the complexity of the original problem. Thus this can be considered as one of the main factors contributing to the achievement of super linear speed-up of the simulation.

Table 4.3 Parallel computation of the LDC fluid flow problem with $Re = 100$; CPUs: number of CPUs; Δt: time step; CM_{tol}: tolerance of convergence measure at interfaces; N_i: number of iterations; T_p: parallel computation time (minutes); CM: average convergence measure of the whole analysis domain.

grid	CPUs	Δt	CM_{tol}	N_i	T_p	CM
101×101	2	$1.E-5$	$1.E-6$	2983	346.66	$3.3862E-07$
–	4	$1.E-5$	$1.E-6$	3039	100.48	$3.2612E-07$
–	6	$1.E-5$	$1.E-6$	3130	49.44	$3.0900E-07$
–	8	$1.E-5$	$1.E-6$	3127	26.32	$3.1188E-07$
–	9	$1.E-5$	$1.E-6$	3153	23.90	$2.6812E-07$
–	12	$1.E-5$	$1.E-6$	3141	13.69	$2.7585E-07$
–	15	$1.E-5$	$1.E-6$	3140	10.22	$2.5592E-07$
–	16	$1.E-5$	$1.E-6$	3142	7.96	$2.5326E-07$
–	20	$1.E-5$	$1.E-6$	3139	5.97	$2.3442E-07$
–	25	$1.E-5$	$1.E-6$	3128	4.54	$2.2448E-07$
–	28	$1.E-5$	$1.E-6$	3125	3.59	$2.1013E-07$
–	30	$1.E-5$	$1.E-6$	3123	3.10	$2.1792E-07$
–	35	$1.E-5$	$1.E-6$	3119	2.70	$2.0436E-07$
–	36	$1.E-5$	$1.E-6$	3118	2.61	$2.0544E-07$
155×155	9	$5.E-6$	$1.E-6$	5806	190.00	$2.8840E-07$
–	12	$5.E-6$	$1.E-6$	5822	110.39	$2.8021E-07$
–	15	$5.E-6$	$1.E-6$	5828	74.76	$2.7005E-07$
–	16	$5.E-6$	$1.E-6$	5837	63.17	$2.5711E-07$
–	20	$5.E-6$	$1.E-6$	5842	41.61	$2.4535E-07$
–	25	$5.E-6$	$1.E-6$	5837	29.23	$2.3962E-07$
–	28	$5.E-6$	$1.E-6$	5764	25.24	$2.2321E-07$
–	30	$5.E-6$	$1.E-6$	5820	23.00	$2.1753E-07$
–	35	$5.E-6$	$1.E-6$	5813	18.27	$2.0763E-07$
–	36	$5.E-6$	$1.E-6$	5732	17.94	$2.1531E-07$
209×209	16	$5.E-6$	$1.E-6$	5862	190.41	$3.4286E-07$
–	20	$5.E-6$	$1.E-6$	5968	134.30	$2.8124E-07$
–	25	$5.E-6$	$1.E-6$	5835	89.89	$3.3027E-07$
–	28	$5.E-6$	$1.E-6$	5823	74.26	$2.6927E-07$
–	30	$5.E-6$	$1.E-6$	5854	61.78	$3.0911E-07$
–	35	$5.E-6$	$1.E-6$	5820	51.69	$2.7399E-07$
–	36	$5.E-6$	$1.E-6$	5820	45.19	$2.2564E-07$
253×253	25	$5.E-6$	$1.E-6$	6715	227.97	$2.9365E-07$
–	28	$5.E-6$	$1.E-6$	6598	190.89	$2.8132E-07$
–	30	$5.E-6$	$1.E-6$	6673	149.42	$2.6219E-07$
–	35	$5.E-6$	$1.E-6$	6619	115.89	$2.4176E-07$
–	36	$5.E-6$	$1.E-6$	6537	101.33	$2.5564E-07$
295×295	36	$4.E-6$	$1.E-6$	8437	313.05	$2.4598E-07$
299×299	42	$4.E-6$	$1.E-6$	8485	207.59	$2.0221E-07$

Table 4.4 Parallel computation of the LDC fluid flow problem with $Re = 400$; CPUs: number of CPUs; Δt: time step; CM_{tol}: tolerance of convergence measure at interfaces; N_i: number of iterations; T_p: parallel computation time (minutes); CM: average convergence measure of the whole analysis domain.

grid	CPUs	Δt	CM_{tol}	N_i	T_p	CM
101×101	2	$1.E-5$	$1.E-6$	1769	199.67	$3.6469E-07$
–	4	$1.E-5$	$1.E-6$	1809	59.51	$3.4120E-07$
–	6	$1.E-5$	$1.E-6$	1907	30.38	$3.1543E-07$
–	8	$1.E-5$	$1.E-6$	1943	17.09	$3.0031E-07$
–	9	$1.E-5$	$1.E-6$	1949	15.34	$2.6732E-07$
–	12	$1.E-5$	$1.E-6$	1980	8.55	$2.6361E-07$
–	15	$1.E-5$	$1.E-6$	1988	6.55	$2.6674E-07$
–	16	$1.E-5$	$1.E-6$	2044	5.06	$2.5307E-07$
–	20	$1.E-5$	$1.E-6$	2092	3.89	$2.2655E-07$
–	25	$1.E-5$	$1.E-6$	2163	3.10	$2.0506E-07$
–	28	$1.E-5$	$1.E-6$	2184	2.51	$2.1807E-07$
–	30	$1.E-5$	$1.E-6$	2199	2.26	$1.9648E-07$
–	35	$1.E-5$	$1.E-6$	2330	2.09	$1.4480E-07$
–	36	$1.E-5$	$1.E-6$	2359	1.86	$1.3151E-07$
155×155	9	$1.E-5$	$1.E-6$	1957	117.43	$2.2188E-07$
–	12	$1.E-5$	$1.E-6$	1978	38.02	$2.0113E-07$
–	15	$1.E-5$	$1.E-6$	1995	25.64	$1.9335E-07$
–	16	$1.E-5$	$1.E-6$	1993	21.82	$2.1742E-07$
–	20	$1.E-5$	$1.E-6$	1996	14.64	$2.1968E-07$
–	25	$1.E-5$	$1.E-6$	2047	10.41	$1.8317E-07$
–	28	$1.E-5$	$1.E-6$	2303	10.34	$1.7704E-07$
–	30	$1.E-5$	$1.E-6$	2292	9.23	$1.7548E-07$
–	35	$1.E-5$	$1.E-6$	2398	7.54	$1.4143E-07$
–	36	$1.E-5$	$1.E-6$	2577	6.54	$1.0865E-07$
209×209	16	$1.E-5$	$1.E-6$	2088	68.21	$3.5483E-07$
–	20	$1.E-5$	$1.E-6$	2192	49.50	$3.5207E-07$
–	25	$1.E-5$	$1.E-6$	2305	35.96	$4.3792E-07$
–	28	$1.E-5$	$1.E-6$	2245	28.56	$3.1715E-07$
–	30	$1.E-5$	$1.E-6$	2293	25.80	$3.5021E-07$
–	35	$1.E-5$	$1.E-6$	2453	23.16	$2.6163E-07$
–	36	$1.E-5$	$1.E-6$	2393	18.43	$2.2997E-07$
253×253	25	$8.E-6$	$1.E-6$	2814	91.87	$3.4725E-07$
–	28	$8.E-6$	$1.E-6$	2980	75.47	$4.1144E-07$
–	30	$8.E-6$	$1.E-6$	3015	67.91	$3.6694E-07$
–	35	$8.E-6$	$1.E-6$	3195	58.14	$2.4023E-07$
–	36	$8.E-6$	$1.E-6$	3236	49.99	$2.7969E-07$
295×295	36	$4.E-6$	$1.E-6$	5164	171.74	$2.5456E-07$
299×299	42	$4.E-6$	$1.E-6$	5333	130.06	$1.9579E-07$

Table 4.5 Parallel computation of the LDC problem with $Re = 1000$; CPUs: number of CPUs; Δt: time step; CM_{tol}: tolerance of convergence measure at interfaces; N_i: number of iterations; T_p: parallel computation time (minutes); CM: average convergence measure of the whole analysis domain.

grid	CPUs	Δt	CM_{tol}	N_i	T_p	CM
101×101	2	$1.E-06$	$1.E-6$	9292	972.00	$2.7711E-07$
–	4	$1.E-6$	$1.E-6$	9146	301.05	$2.9968E-07$
–	6	$1.E-6$	$1.E-6$	8958	246.58	$3.6250E-07$
–	8	$1.E-6$	$1.E-6$	8789	76.40	$3.9178E-07$
–	9	$1.E-6$	$1.E-6$	8964	68.32	$3.3912E-07$
–	12	$1.E-6$	$1.E-6$	8936	37.71	$3.1141E-07$
–	15	$1.E-6$	$1.E-6$	8901	29.06	$2.8634E-07$
–	16	$1.E-6$	$1.E-6$	8866	21.68	$2.4941E-07$
–	20	$1.E-6$	$1.E-6$	8862	16.27	$2.2520E-07$
–	25	$1.E-6$	$1.E-6$	8881	12.11	$2.1926E-07$
–	28	$1.E-6$	$1.E-6$	9062	9.54	$1.9764E-07$
–	30	$1.E-6$	$1.E-6$	8685	8.54	$2.4913E-07$
–	35	$1.E-6$	$1.E-6$	8764	7.27	$2.3634E-07$
–	36	$1.E-6$	$1.E-6$	9326	6.71	$1.5421E-07$
155×155	9	$1.E-6$	$1.E-6$	8887	290.22	$3.5905E-07$
–	12	$1.E-6$	$1.E-6$	8973	170.57	$3.2145E-07$
–	15	$1.E-6$	$1.E-6$	8979	137.84	$3.0020E-07$
–	16	$1.E-6$	$1.E-6$	9018	97.69	$2.6282E-07$
–	20	$1.E-6$	$1.E-6$	9072	79.81	$2.3382E-07$
–	25	$1.E-6$	$1.E-6$	9094	46.22	$2.1900E-07$
–	28	$1.E-6$	$1.E-6$	9565	43.57	$1.8332E-07$
–	30	$1.E-6$	$1.E-6$	9171	36.38	$2.2463E-07$
–	35	$1.E-6$	$1.E-6$	9901	29.89	$1.5324E-07$
–	36	$1.E-6$	$1.E-6$	11704	36.81	$1.2794E-07$
209×209	16	$5.E-6$	$1.E-6$	3435	203.43	$1.7919E-07$
–	20	$5.E-6$	$1.E-6$	3738	148.03	$1.6568E-07$
–	25	$5.E-6$	$1.E-6$	3944	107.23	$1.6116E-07$
–	28	$5.E-6$	$1.E-6$	3890	75.06	$1.6030E-07$
–	30	$5.E-6$	$1.E-6$	3988	69.16	$1.5540E-07$
–	35	$5.E-6$	$1.E-6$	4118	60.20	$1.3527E-07$
–	36	$5.E-6$	$1.E-6$	4268	54.84	$1.0258E-07$
253×253	25	$5.E-6$	$1.E-6$	3798	227.09	$1.5637E-07$
–	28	$5.E-6$	$1.E-6$	4043	160.56	$1.5123E-07$
–	30	$5.E-6$	$1.E-6$	4042	139.30	$1.4436E-07$
–	35	$5.E-6$	$1.E-6$	4131	124.96	$1.3336E-07$
–	36	$5.E-6$	$1.E-6$	4486	114.92	$8.8219E-08$
295×295	36	$5.E-6$	$1.E-6$	4550	271.60	$2.3908E-07$
299×295	42	$5.E-6$	$1.E-6$	4849	212.26	$2.0420E-07$

Table 4.6 Parallel computation using the present method for the LDC problem with $Re = 100$, grid 101×101 and $\Delta t = 1.E-5$. $n_x \times n_y$: grid points; $CPUs$: number of CPUs; CN: condition number of the system matrix; CM_p: average convergence measure of the whole analysis domain; N_i: number of iterations on the whole domain; T_{cmp}: computational time for one iteration in sub-domain (seconds); T_{cmm}: total communication time (minutes); T_p: total parallel computation time (minutes).

$CPUs$	CN	CM_p	N_i	T_{cmp}	T_{cmm}	T_p
2	2190.99	$3.3862E-7$	2983	6.891	4.06	346.66
4	1389.97	$3.2612E-7$	3039	1.971	0.63	100.48
9	617.80	$2.6812E-7$	3153	0.350	5.48	23.90
16	322.27	$2.5326E-7$	3142	0.143	0.45	7.96
25	230.75	$2.2448E-7$	3128	0.065	1.14	4.54
36	154.49	$2.0544E-7$	3118	0.037	0.67	2.61

4.6 Conclusion

In this chapter, a parallel method based on the combination of the overlapping DD technique and CLIRBF methods has been presented. Advantages of the new method include (i) to alleviate the ill-condition problem associated with IRBF methods; (ii) to avoid the reduction in convergence rate caused by differentiation and (iii) to achieve much higher throughput in solving large scale problems. The method has been successfully verified through numerical solution of problems with available analytic solution and the benchmark LDC fluid flow problem using Matlab Distributed Computing Engine. While, in this chapter, point collocation has been the mechanism for the discretisation of PDEs, the next chapter will discuss the control volume approach.

Chapter 5

Parallel control-volume method based on local integrated RBFs for solving fluid flow problems

In this chapter, a high performance computing (HPC) method based on the integrated Radial Basis Function (IRBF), Control Volume method (CVM) and Domain Decomposition (DD) method for solving Partial Differential Equations (PDEs) is presented (Pham-Sy et al., 2014). The goal is to develop an efficient parallel algorithm based on the local IRBF method using the control volume approach, especially for problems defined on non-rectangular domains. The results showed that the goal is attained as the achieved parallel efficiency is high. For the case of square lid-driven cavity (LDC) flow problem with Reynolds number equal to 100, super-linear speed-up is also achieved. The parallel algorithm is implemented in the MATLAB environment using Parallel Computing Toolbox and Distributed Computing Engine.

5.1 Introduction

Integrated radial basis function method was proposed by Mai-Duy and Tran-Cong (2001) as an alternative to the RBF interpolation of scattered data by Kansa (1990a), which is here referred to as the differential RBF (DRBF) method. Thanks to the integration approach, IRBF method is shown to have higher order of accuracy than the DRBF method. The convergence rate was then boosted by using local/compact local schemes to form tridiagonal matrix (Hoang-Trieu et al., 2012; Chandhini and Sanyasiraju, 2007).

The IRBF method is further developed in combination with CVM (Patankar, 1980) to improve the accuracy of the solution in non-rectangular domains (Mai-Duy and Tran-Cong, 2010). With the inherent conservation of mass, momentum and energy over control volumes, this method has been shown to be a very effective way to deal with domains with complex boundary. In this chapter, the CVM is employed in combination with two-dimensional (2D) IRBF to simulate

fluid flow in the triangular LDC flow problem (Kohno and Bathe, 2006).

Since the scale of practical engineering problems is huge in terms of degrees of freedom (DoF), modern computational mechanics has begun to embrace parallel paradigms. With the help of parallelisation, the shortage of memory and computational power is being addressed. What remains challenging is parallel algorithms, which is the main focus of this chapter. The method being considered is a DD method, which is one of the most popular methods for solving large scale problems (Quarteroni and Valli, 1999). The DD method is used to split the computational domain into smaller sub-domains which are solved separately. Originally, sub-domains are solved sequentially one after another. In order to improve the throughput of the simulation, in this chapter, the sub-domains are solved in parallel. The DD method being used is the Schwarz additive overlapping DD method and the communication between parallel sub-domains are handled by MATLAB supported Message Passing Interface (MPI).

The chapter is organised as follows. In Sections 5.2 and 5.3, a brief review of local IRBF and CVM is presented. The DD method is described in Section 5.4. Numerical results are then given and discussed in Section 5.5 with a conclusion in Section 5.6.

5.2 Local methods based on integrated radial basis function

In this section, a brief review of several IRBF local approaches, including one-dimension (1D) IRBF and two-dimension (2D) local 9-point IRBF stencils, is provided.

5.2.1 1D-IRBF method

Consider following 2D PDE

$$\nabla^2 u(\mathbf{x}) = f(\mathbf{x}), \mathbf{x} \in \Omega \tag{5.1}$$

where u is the field variable; $\mathbf{x}$ the position vector; Ω the considered domain and f a known function of $\mathbf{x}$.

By means of IRBF, the highest order derivatives of the PDE, which is second order in this case, are approximated by a weighted set of RBFs as

$$\frac{\partial^2 u(\mathbf{x})}{\partial x_j^2} = \sum_{i=1}^{n} w_i^{[x_j]} G_{[x_j]i}^{[2]}(\mathbf{x}), \tag{5.2}$$

where x_j is the j-component of $\mathbf{x}$ ($j = 1, 2$); $\{w_i\}_{i=1}^n$ the set of weights and $\{G_i^{[2]}(\mathbf{x})\}_{i=1}^n$ the set of RBFs. The multi-quadric (MQ) RBF is used in this chapter and given by

$$G_i(\mathbf{x}) = \sqrt{(\mathbf{x} - \mathbf{c}_i)^2 + a_i^2},$$

where $\{\mathbf{c}_i\}_{i=1}^n$ is a set of centres and $\{a_i\}_{i=1}^n$ a set of MQ-RBF widths.

To obtain first-order derivatives and field variable, Eq. (5.2) is integrated successively with respect to x_j as follows.

$$\frac{\partial u}{\partial x_j} = \sum_{i=1}^{n} w_i^{[x_j]} G_{[x_j]i}^{[1]}(\mathbf{x}) + C_1, \tag{5.3}$$

$$u = \sum_{i=1}^{n} w_i^{[x_j]} G_{[x_j]i}^{[0]}(\mathbf{x}) + C_1 x_j + C_2, \tag{5.4}$$

where $G_{[x_j]i}^{[1]}(\mathbf{x}) = \int G_{[x_j]i}^{[2]}(\mathbf{x}) dx_j$, $G_{[x_j]i}^{[0]}(\mathbf{x}) = \int G_{[x_j]i}^{[1]}(\mathbf{x}) dx_j$ and C_1 and C_2 are constants of integration in the sense that $C_i = C_i(x_k)$, $k \neq i$.

Collocating equations (5.2) - (5.4) at grid points $\{\mathbf{x}_i\}_{i=1}^n$ yields

$$\frac{\partial^2 \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}_{x_j}^{[2]} \widetilde{\mathbf{w}}_{x_j}, \tag{5.5}$$

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}_{x_j}^{[1]} \widetilde{\mathbf{w}}_{x_j}, \tag{5.6}$$

$$\widetilde{\mathbf{u}}_{x_j} = \mathcal{G}_{x_j}^{[0]} \widetilde{\mathbf{w}}_{x_j}, \tag{5.7}$$

with

$$\begin{aligned}
\widetilde{\mathbf{w}}_{x_j} &= \left(w_1^{[x_j]}, w_2^{[x_j]}, \cdots, w_n^{[x_j]}, C_1, C_2\right)^T, \\
\widetilde{\mathbf{u}}_{x_j} &= \left(u_1^{[x_j]}, u_2^{[x_j]}, \cdots, u_n^{[x_j]}\right)^T, \\
\frac{\partial^k \widetilde{\mathbf{u}}}{\partial x_j^k} &= \left(\frac{\partial^k u_1}{\partial x_j^k}, \frac{\partial^k u_2}{\partial x_j^k}, \cdots, \frac{\partial^k u_n}{\partial x_j^k}\right)^T,
\end{aligned}$$

where $u_i = u(\mathbf{x}_i)$ $(i = 1, 2, \cdots, n)$; $\mathcal{G}_{[x_j]}^{[2]}, \mathcal{G}_{[x_j]}^{[1]}$ and $\mathcal{G}_{[x_j]}^{[0]}$ are known matrices of size $n \times (n+2)$ as presented below.

$$\mathcal{G}_{[x_j]}^{[k]} = \begin{bmatrix}
G_{[x_j]1}^{[k]}(\mathbf{x}_1) & G_{[x_j]2}^{[k]}(\mathbf{x}_1) & \cdots & G_{[x_j]n}^{[k]}(\mathbf{x}_1) & a_1^{[k]} & b_1^{[k]} \\
G_{[x_j]1}^{[k]}(\mathbf{x}_2) & G_{[x_j]2}^{[k]}(\mathbf{x}_2) & \cdots & G_{[x_j]n}^{[2]}(\mathbf{x}_2) & a_2^{[k]} & b_2^{[k]} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
G_{[x_j]1}^{[k]}(\mathbf{x}_n) & G_{[x_j]2}^{[k]}(\mathbf{x}_n) & \cdots & G_{[x_j]n}^{[k]}(\mathbf{x}_n) & a_n^{[k]} & b_n^{[k]}
\end{bmatrix},$$

where $k = [0, 1, 2]$ and let $A^{[k]} = \left(a_1^{[k]}, a_2^{[k]}, \cdots, a_n^{[k]}\right)^T$, $B^{[k]} = \left(b_1^{[k]}, b_2^{[k]}, \cdots, b_n^{[k]}\right)^T$ then

$$A^{[k]} = \begin{cases}
\begin{pmatrix} 0 & \cdots & 0 \end{pmatrix}^T, & k = 2 \\
\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^T, & k = 1 \\
\begin{pmatrix} x_{j1} & \cdots & x_{jn} \end{pmatrix}^T, & k = 0
\end{cases}$$

and

$$B^{[k]} = \begin{cases}
\begin{pmatrix} 0 & \cdots & 0 \end{pmatrix}^T, & k = 1, 2 \\
\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}^T, & k = 0
\end{cases}$$

5.2.2 2D-IRBF local stencil scheme

In this chapter, a 9-point stencil scheme is employed to overcome the problem of ill-conditioned system matrix, which is an crucial problem in global IRBF approach. According to this scheme, a local 9-point stencil for an arbitrary grid-point $\mathbf{x}_5$ is described in (Fig. 5.1).

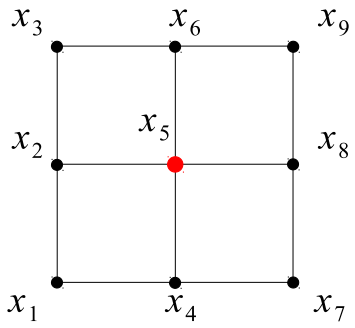


Figure 5.1 A 9-point local stencil.

Within this stencil, a local IRBF approximation is employed to discretise the governing equation at $\mathbf{x}_5$. The local IRBF approximation, coupled with the CVM, will be presented in the next section.

5.3 A control volume method based on 2D-IRBF

In this approach, each grid point is surrounded by a CV and the conservative governing equations are integrated within this volume. Fig. 5.2 shows the CV formation for a regular 2D domain. In this figure, CVs are bounded by lines parallel to grid lines through the middle points between the reference point and its neighbours. The figure shows the full CV for the point in the middle $\mathbf{x}_5$ and half CV for the point on the left $\mathbf{x}_2$.

Consider the 2D Poisson equation (5.1). This equation is integrated over a CV, and then by applying the divergence theorem to the resultant equation one gets

$$\begin{aligned}\oint_{\Gamma_s} \nabla u(\mathbf{x}) \cdot \hat{n} d\Gamma_s &= \int_{y_b}^{y_t} \left.\frac{\partial u(\mathbf{x})}{\partial x}\right|_r dy - \int_{y_b}^{y_t} \left.\frac{\partial u(\mathbf{x})}{\partial x}\right|_l dy + \int_{x_l}^{x_r} \left.\frac{\partial u(\mathbf{x})}{\partial x}\right|_t dx - \int_{x_l}^{x_r} \left.\frac{\partial u(\mathbf{x})}{\partial x}\right|_b dx \\ &= \int_{\Omega_s} f(\mathbf{x}) d\Omega_s,\end{aligned} \tag{5.8}$$

where Ω_s and Γ_s are the CV under consideration and its surface, respectively; $\hat{n}$ the outward normal unit vector and $(.)|_{(s)}$, $(s = l, r, t, b)$ depicts integrals over the left, right, top and bottom faces of the CV respectively.

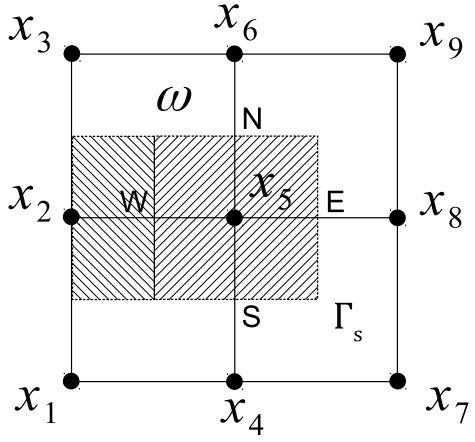


Figure 5.2 CV formation in 2D.

Using 5-point Gaussian quadrature scheme to discretise Eq. (5.8) yields

$$\frac{\Delta y}{2}\sum_{j=1}^{5}\alpha_j\left.\frac{\partial u\left(y\left(\eta_j\right)\right)}{\partial x}\right|_r-\frac{\Delta y}{2}\sum_{j=1}^{5}\alpha_j\left.\frac{\partial u\left(y\left(\eta_j\right)\right)}{\partial x}\right|_l+\frac{\Delta x}{2}\sum_{i=1}^{5}\alpha_i\left.\frac{\partial u\left(x\left(\eta_i\right)\right)}{\partial y}\right|_t$$
$$-\frac{\Delta x}{2}\sum_{i=1}^{5}\alpha_i\left.\frac{\partial u\left(x\left(\eta_i\right)\right)}{\partial y}\right|_r=\frac{\Delta x\Delta y}{4}\sum_{i=1}^{5}\sum_{j=1}^{5}\alpha_i\alpha_j f\left(x\left(\eta_i\right)y\left(\eta_j\right)\right), \tag{5.9}$$

where α_k and η_k $(k=i,j)$ are the weights and Gauss points, respectively.

The 2D local IRBF approximation scheme mentioned in Section 5.2.2 is used to approximate the field variable and its derivatives in Eq. (5.9). Thus, in this approach the governing equations are forced to be satisfied locally over CVs while the boundary conditions are directly imposed using the 1D-IRBF approximant. The procedure leads to an algebraic equation system for unknown nodal values of the field variable as follows.

$$\left(\begin{array}{c}\widetilde{\mathbf{u}}\\ \widetilde{\mathbf{0}}\end{array}\right)=\underbrace{\left[\begin{array}{cc}\mathcal{G}_x^{[0]}, & \mathcal{O}\\ \mathcal{G}_x^{[0]}, & -\mathcal{G}_y^{[0]}\end{array}\right]}_{\mathcal{C}}\left(\begin{array}{c}\widetilde{\mathbf{w}}_x\\ \widetilde{\mathbf{w}}_y\end{array}\right)=\mathcal{C}\left(\begin{array}{c}\widetilde{\mathbf{w}}_x\\ \widetilde{\mathbf{w}}_y\end{array}\right), \tag{5.10}$$

where $\mathcal{C}$ is the conversion matrix, $\widetilde{\mathbf{w}}_x$ and $\widetilde{\mathbf{w}}_y$ the RBF weight vectors of length 15; $\widetilde{\mathbf{u}}$ the vector of length 9, and $\widetilde{\mathbf{0}}$ the zeros vector of length 9; $\mathcal{O}$ the zeros matrix of dimension 9×15, and $\mathcal{G}_x^{[0]}$ and $\mathcal{G}_y^{[0]}$ the known matrices of dimensions 9×15. Furthermore, $\widetilde{\mathbf{u}}$, $\widetilde{\mathbf{w}}_x$ and $\widetilde{\mathbf{w}}_y$ are given by

$$\widetilde{\mathbf{u}}=(u_1,\ldots,u_9)^T, \tag{5.11}$$
$$\widetilde{\mathbf{w}}_x=(w_1^{[x]},\ldots,w_9^{[x]},C_1^x(y_1),C_1^x(y_2),C_1^x(y_3),C_2^x(y_1),C_2^x(y_2),C_2^x(y_3))^T, \tag{5.12}$$
$$\widetilde{\mathbf{w}}_y=(w_1^{[y]},\ldots,w_9^{[y]},C_1^y(x_1),C_1^y(x_2),C_1^y(x_3),C_2^y(x_1),C_2^y(x_2),C_2^y(x_3))^T, \tag{5.13}$$

$$\mathcal{G}_x^{[0]} = \begin{bmatrix} G_{[x]1}^{[0]}(\mathbf{x}_1) & \cdots & G_{[x]9}^{[0]}(\mathbf{x}_1) & x_1 & 0 & 0 & 1 & 0 & 0 \\ G_{[x]1}^{[0]}(\mathbf{x}_2) & \cdots & G_{[x]9}^{[0]}(\mathbf{x}_2) & 0 & x_2 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & x_3 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & x_4 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & x_5 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & x_6 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & x_7 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & x_8 & 0 & 0 & 1 & 0 \\ G_{[x]1}^{[0]}(\mathbf{x}_9) & \cdots & G_{[x]9}^{[0]}(\mathbf{x}_9) & 0 & 0 & x_9 & 0 & 0 & 1 \end{bmatrix}, \tag{5.14}$$

$$\mathcal{G}_y^{[0]} = \begin{bmatrix} G_{[y]1}^{[0]}(\mathbf{x}_1) & \cdots & G_{[y]9}^{[0]}(\mathbf{x}_1) & y_1 & 0 & 0 & 1 & 0 & 0 \\ G_{[y]1}^{[0]}(\mathbf{x}_2) & \cdots & G_{[y]9}^{[0]}(\mathbf{x}_2) & y_2 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & y_3 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & y_4 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & y_5 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & y_6 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & y_7 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & 0 & 0 & y_8 & 0 & 0 & 1 \\ G_{[y]1}^{[0]}(\mathbf{x}_9) & \cdots & G_{[y]9}^{[0]}(\mathbf{x}_9) & 0 & 0 & y_9 & 0 & 0 & 1 \end{bmatrix}, \tag{5.15}$$

where $G_{[x]i}^{[0]}$ and $G_{[y]i}^{[0]}$ $(i = 1..9)$ were defined in Section 5.2.1 in the x and y-directions and x_i, y_i are the two components of $\mathbf{x}_i$. It is noted that in (5.10)

$$\widetilde{\mathbf{u}} = [\mathcal{G}_x^{[0]}, \mathcal{O}] \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}$$

is obtained by collocating the field variable over a local stencil, and

$$\widetilde{\mathbf{0}} = [\mathcal{G}_x^{[0]}, -\mathcal{G}_y^{[0]}] \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}$$

is derived from the consistency condition $\int\int \frac{\partial^2 u}{\partial x^2} dx \Big|_{\mathbf{x}_i} = \int\int \frac{\partial^2 u}{\partial y^2} dy \Big|_{\mathbf{x}_i}$.

The conversion of the network-weight space into the physical space is achieved by first inverting Eq. (5.10)

$$\begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C}^{-1} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \end{pmatrix}. \tag{5.16}$$

Then by substituting Eq. (5.16) into Eqs. (5.3) and (5.4) the first order derivatives of u with respect to x and y and the function itself over a local stencil are determined.

In the case of non-rectangular domains, the CV formation for interior points is carried out in a similar way but with some extra treatments for non-rectangular boundaries, which will be detailed in Section 5.5.2.

5.4 Parallel domain decomposition method

Domain decomposition has been successfully used to overcome the resource limitation associated with large-scale problems. Its primary objective is to split a large problem domain into small ones called sub-domains in which the problem can be solved more effectively in terms of memory and computing power (Quarteroni and Valli, 1999; Tran et al., 2009). A notable advantage of DD method in solving numerical problems is that it helps to decrease the condition number of system matrices in sub-domains. As a result, DD method helps to achieve a more stable and accurate solution. Furthermore, with the advance of parallel computing, DD technique finds itself very parallel capable. That potential for parallelisation further encourages more intensive research in DD method in recent decades.

Over the last two decades, researchers have developed parallel algorithms owing to the simplicity of grid generation to significantly increase the throughput of numerical solutions. For example, Singh and Jain (2005) used an Element-Free Galerkin method with moving least-square approximant to solve fluid flow problems. They were able to achieve high efficiency, e.g. 91.27% for a 2D problem with eight CPUs. Shirazaki and Yagawa (1999) proposed a Mesh-Free method based parallel algorithm to solve incompressible viscous flow. They obtained a stable solution to a model with three-million DoF. However, as the speed-up was separated into two parts, namely the construction of system equations and the time integration, the efficiency of the first part was very high and even super-linear with high number of CPUs, the efficiency of the second part was not able to scale to high number of CPUs. Indeed, the efficiency dropped from approximately 98% with 16 CPUs to around 50% with 64 CPUs. Ingber et al. (2004) combined the method of fundamental solutions and the particular solution method to solve the transient heat conduction problems. The approach was developed using a Schwarz Neumann-Neumann DD based parallel scheme. Although the authors successfully demonstrated the accuracy of parallel algorithm in comparison with the non-parallel version, unfortunately, information regarding the efficiency of parallelisation was not given.

This algorithm is a further development of the Schwarz Additive Overlapping DD technique (Pham-Sy et al., 2013) using the local stencil IRBF approximants. Here, the local stencil 2D-IRBF based CVM, which is presented in the previous sections, is used to develop the parallel algorithm for solving fluid flow problems.

The additive overlapping DD method is a rather simple but effective method. It also has a high potential for parallelisation as the computation in each sub-domain is independent within a time step. In this approach, the original domain is divided into several overlapping sub-domains. The function values on artificial boundaries (ABs) are initially unknown and are set to zeros (initial guess). In each iterative step, the boundary value problem is solved separately in each sub-domain. Then the function values on the AB of one sub-domain are updated by the solution from other sub-domains. This procedure is repeated until a desired tolerance is achieved. Several specific details on the use of additive overlapping

DD technique will be presented briefly in the next sections.

5.4.1 Sub-domain formation and neighbour identification

The sub-domain formation task is straightforward with a rectangular domain as can be seen in Fig. 5.3. This formation has been reported in our previous work. However, it is also presented here for completeness. For example, in Fig. 5.3 the original domain is decomposed into 12 sub-domains. Each sub-domain is assigned a unique number called process-index. In this example, sub-domains are enumerated with process-index from 1 to 12. Each sub-domain also has a 2D index (i, j) that determines its position in the original domain. The 2D index is used to determine neighbours of a sub-domain. In this example, the 2D index of sub-domain 7 is $(3, 2)$ and its neighbours are sub-domains 3, 6, 8 and 11.

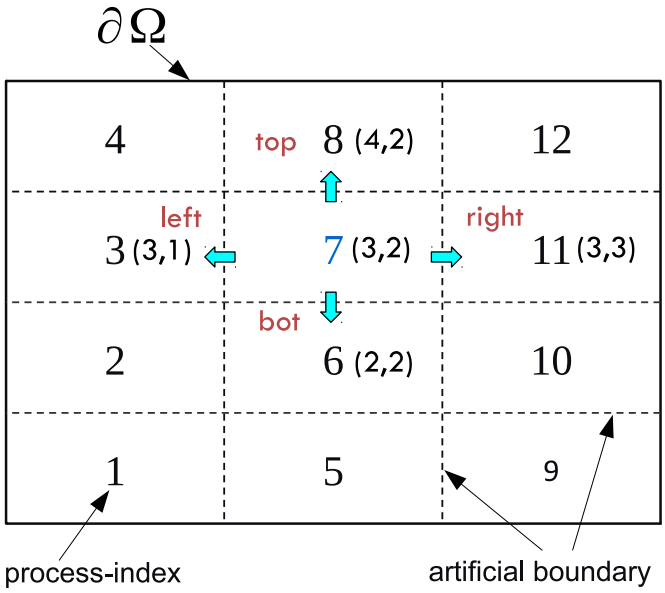


Figure 5.3 Enumeration in a system of $N_x \times N_y = 4 \times 3$ sub-domains of a rectangular domain.

For a non-rectangular domain (e.g., Fig. 5.4) the process of determining neighbours is a bit more difficult because some sub-domains will lie outside the considered domain. For example, in Fig. 5.4 sub-domains 8 and 9 are outside of the triangular domain Ω. To overcome this situation, one first needs to create a list of 2D indices of all sub-domains in the system. Then a sub-domain has left, right, top and bottom neighbours if processes with 2D indices $(i, j-1)$, $(i, j+1)$, $(i+1, j)$ and $(i-1, j)$ respectively exist in that list.

A detailed example of this process is provided in Table 5.1 with 7 sub-domains in a triangular domain presented in Fig. 5.4.

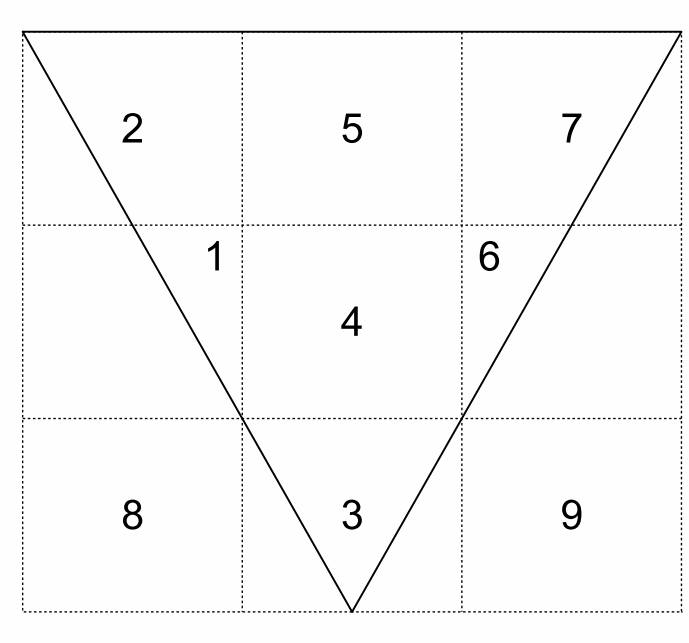


Figure 5.4 Enumeration in a system of 7 sub-domains of a triangular domain.

Table 5.1 Neighbour sub-domain (NB) determination for a triangular domain

labindex	2d-index	left NB	right NB	top NB	bottom NB
1	(2,1)	(2,0) $\sim$ nil	(2,2) $\sim$ 4	(3,1) $\sim$ 2	(1,1) $\sim$ nil
2	(3,1)	(3,0) $\sim$ nil	(3,2) $\sim$ 5	(4,1) $\sim$ nil	(2,1) $\sim$ 1
3	(1,2)	(1,1) $\sim$ nil	(1,3) $\sim$ nil	(2,2) $\sim$ 4	(0,2) $\sim$ nil
4	(2,2)	(2,1) $\sim$ 1	(2,3) $\sim$ 6	(3,2) $\sim$ 5	(1,2) $\sim$ 3
5	(3,2)	(3,1) $\sim$ 2	(3,3) $\sim$ 7	(4,2) $\sim$ nil	(2,2) $\sim$ 4
6	(2,3)	(2,2) $\sim$ 4	(2,4) $\sim$ nil	(3,3) $\sim$ 7	(1,3) $\sim$ nil
7	(3,3)	(3,2) $\sim$ 5	(3,4) $\sim$ nil	(4,3) $\sim$ nil	(2,3) $\sim$ 6

5.4.2 Communication and Synchronisation

In additive overlapping DD method, one of the critical tasks is the communication between the sub-domains as the function values on the AB of one sub-domain are obtained from the solution in its neighbouring sub-domains (NSs) in the previous step. In the present implementation, MATLAB built-in parallel communication method is utilised. MATLAB communication functions allow to send an array of data to MATLAB workers in a synchronised way, which means the sender must wait until the receiver fully receives a message. This mechanism itself guarantees the synchronisation between sub-domains and no extra care is needed to ensure that all sub-domains are always executing the same iterative step. More information about MATLAB supported MPI implementation can be found in (MATLAB, 2012).

5.4.3 Termination

Since the parallel algorithm presented in this chapter is a Distributed Computing Algorithm, it needs to have a termination detection process. This process has been investigated and classified into a unique class of algorithm called Distributed Termination Detection (DTD). In this chapter, the Bitmap DTD algorithm presented in Chapter 3 is employed. This algorithm has several advantages such as symmetric detection, decentralised control and low termination detection delay, and thus ideally suits the implementation of parallel algorithm in the present work.

5.4.4 Parallelisation

The present parallel method is based on the combination of the local stencil 2D-IRBF and CV, and the DD technique presented in the previous sections can now be described in an overall algorithm whose flowchart is shown in Fig. 5.5. In this flowchart, the termination condition is reached once the convergence measurement in every sub-domain (CM) and the convergence measurement on the overlapping area ($ABCM$) are bellow some predefined tolerances CM_{tol} and $ABCM_{tol}$, respectively.

5.5 Numerical results

The proposed method is verified through the simulation of the lid-driven cavity (LDC) fluid flow problem for two cases of rectangular and non-rectangular domains. The efficiency of the present method is analysed.

The LDC fluid flow problem has been commonly used for verification of a numerical method owing to the availability of benchmark solutions in the literature. The problem has also been quite popular among meshless community, e.g. Lin and Atluri (2001) with meshless Local Petrov-Galerkin (MLPG) method; Shu et al. (2005) with local RBF-based Differential Quadrature method; Chinchapatnam et al. (2007) with RBF; and Kim et al. (2007) with meshfree point collocation method. Therefore, in this chapter the LDC flow is also employed to investigate the accuracy as well as the efficiency of the present parallel scheme.

The problem is defined in the stream-function - vorticity formulation as follows.

$$\frac{\partial \omega}{\partial t} + \left(\frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y}\right) = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right), \tag{5.17}$$

$$-\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}, \tag{5.18}$$

where Re is the Reynolds number, ψ the stream function; ω the vorticity and t the time. The $x-$ and $y-$ velocity components are given by $u = \dfrac{\partial \psi}{\partial y}$ and $v = -\dfrac{\partial \psi}{\partial x}$.

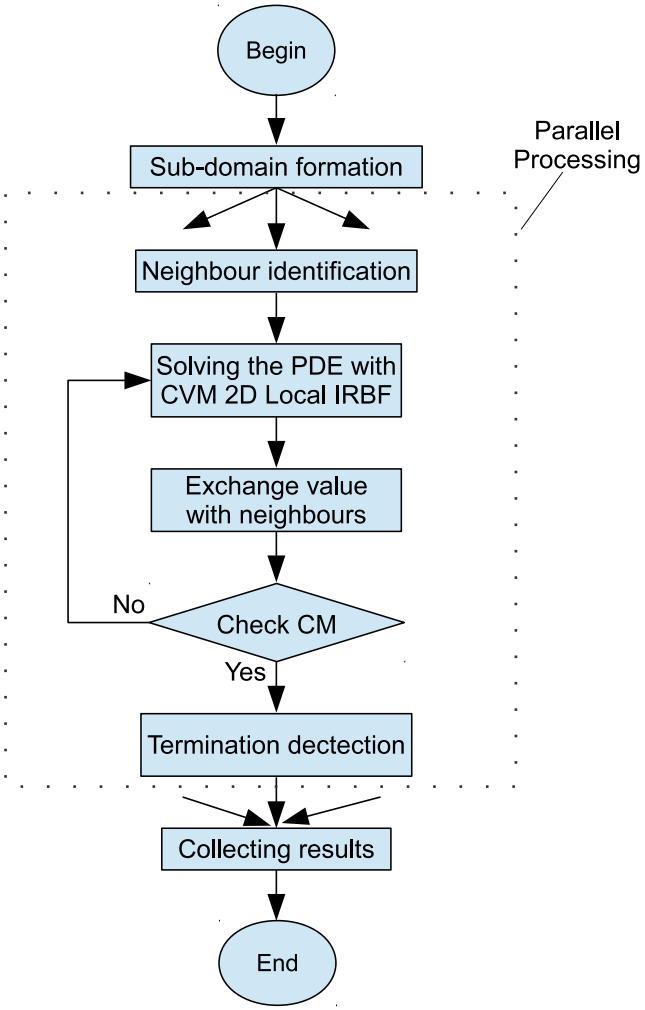


Figure 5.5 Algorithm of the parallel DD method using the local IRBF based CV approach.

The problem is solved using the local 9-point stencil 2D-IRBF scheme as presented in Section 5.2.2 with the time derivative being discretised using a first-order Euler scheme and the diffusive terms being treated implicitly. The boundary condition for ω is computed through Eq. (5.18) using 1D global IRBF as described in Section 5.2.1.

The general procedure for solving a LDC problem is given as follows.

1. Guess the initial values of ω;
2. Solve (5.18) for ψ;
3. Approximate the values of ω on boundaries and the convective terms;
4. Solve (5.17) for ω;
5. Check convergence measure for ω.

5.5.1 Square lid-driven cavity fluid flow problem

For the square LDC fluid flow problem, the geometry of the analysis domain with the chosen coordinate system is shown in Fig. 5.6. The boundary conditions are given in terms of the stream-function as.

$$\psi = 0, \quad \frac{\partial \psi}{\partial x} = 0 \quad \forall (x, y) \in \Gamma_2 \cup \Gamma_3; \tag{5.19}$$

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 0 \quad \forall (x, y) \in \Gamma_4; \tag{5.20}$$

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 1 \quad \forall (x, y) \in \Gamma_1. \tag{5.21}$$

The Dirichlet boundary condition on ψ is used to solve Eq. (5.18) in step 2, while the Neumann boundary condition on ψ is used to approximate the value of ω on boundaries. The values of ω on boundaries, in turn, are used as boundary conditions to solve Eq. (5.17) in step 4 above.

The iterative procedure for solving the square LDC fluid flow problem with parallel DD method is as follows.

1. Divide the analysis domain into a number of sub-domains. Guess initial boundary condition on ABs;
2. Solve the fluid flow problem in each and every sub-domain as described above;
3. Exchange the values of ψ and ω at interfaces with neighbours;
4. Calculate CM on all interfaces;

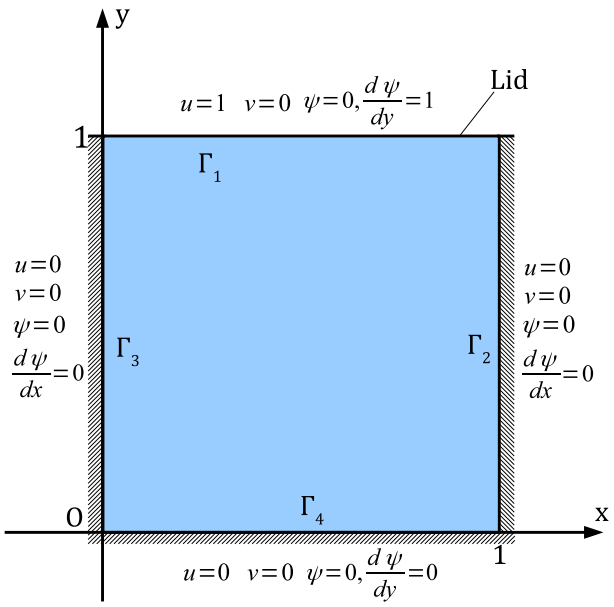
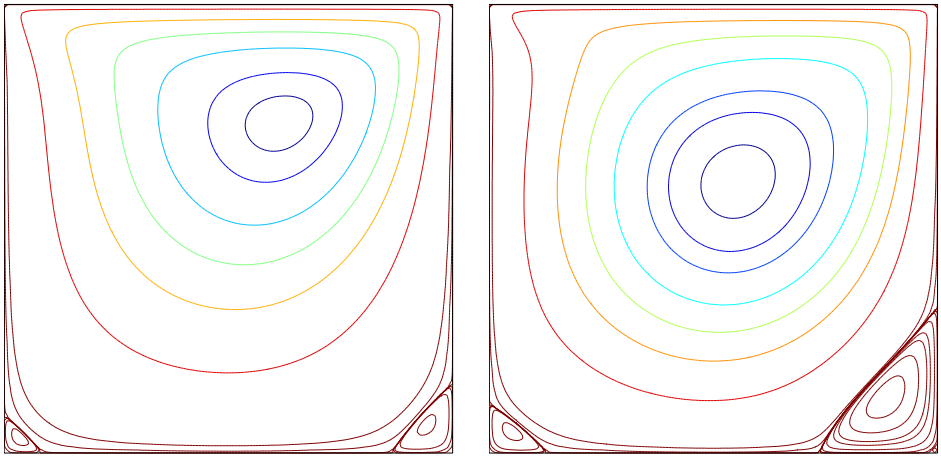


Figure 5.6 The square LDC fluid flow problem. Geometry and boundary conditions. No slip is assumed between the fluid and solid surfaces. The top lid is moving from left to right with a speed of 1.

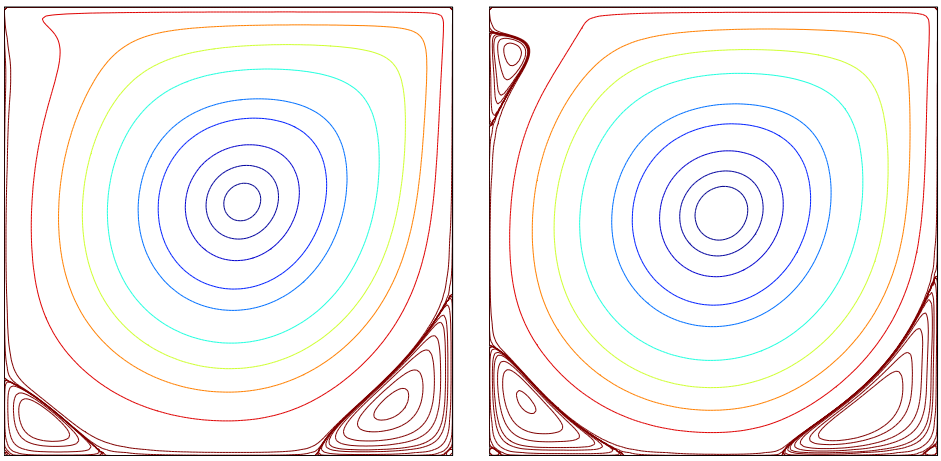
5. Check for termination condition.

In this chapter, the square LDC problem is simulated for a range of Reynolds numbers $Re = \{100, 400, 1000, 3200\}$. Figure 5.7 depicts streamlines of the flow obtained by the present parallel method using a grid of 151×151 collocation points, $\Delta t = 10^{-3}$, $ABCM_{tol} = 10^{-6}$, $CM_{tol} = 10^{-6}$ and $\beta = 2$, 4 sub-domains associated with 4 CPUs for $Re = 100, 400, 1000$ (Figs. 5.7(a) - 5.7(c)) and 2 sub-domains for $Re = 3200$ (Fig.5.7(d)). The results are in very good agreement with those presented in Ghia et al. (1982) as well as in Botella and Peyret (1998). Similar results can be found for vorticity contours in Fig. 5.8. Furthermore, Fig. 5.9 provides the profiles of the velocities along the vertical and horizontal centrelines by the present method along with the benchmark values from Ghia et al. (1982). As can be seen, the results match up very well with the benchmark solution.

The efficiency of the present parallel method is assessed using the following criteria: the number of iterations N_i, computation time T_p, speed-up S - ratio of the computation times using one processor and multi processors and efficiency E - the ratio of speed-up and the number of CPUs used. A fixed grid 151×151 is chosen to run the problem with various number of CPUs and the results are provided in Tables 5.2 - 5.5 for different Reynolds numbers. Results described in the left hand side of Tables 5.2 - 5.5 show that the computation time of the present parallel method (the P-CV method) for the time-dependent square LDC problem decrease quickly as the number of CPUs increases. An interpretation on the significant improvement of throughput can be found in Pham-Sy et al. (2013) for the parallel collocation method; a similar interpretation is applicable here. Again, there are always some thresholds, called CPU_{opt} over which the increase of number of CPUs influences insignificantly on the efficiency based on all criteria (T_p, S and E). For example, the improvement of efficiency of computation is not significant anymore as the number of CPUs is more than 49, 64, 30 and 49

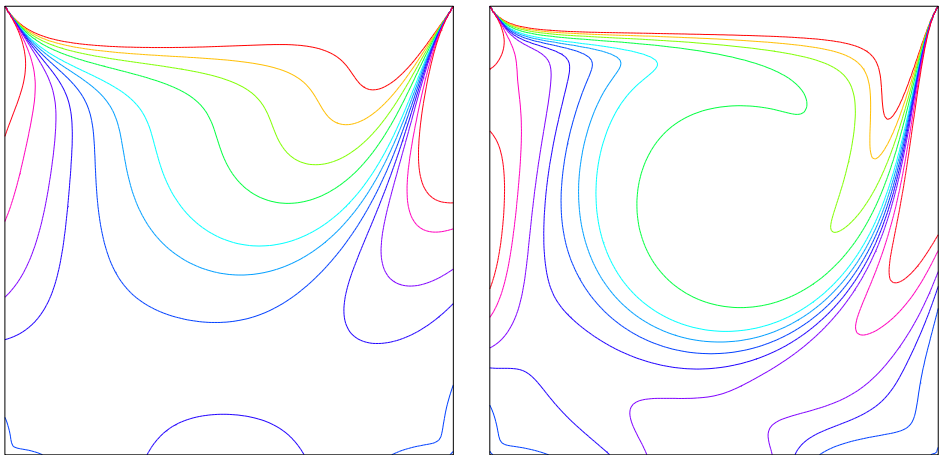


(a) $Re = 100$ (b) $Re = 400$

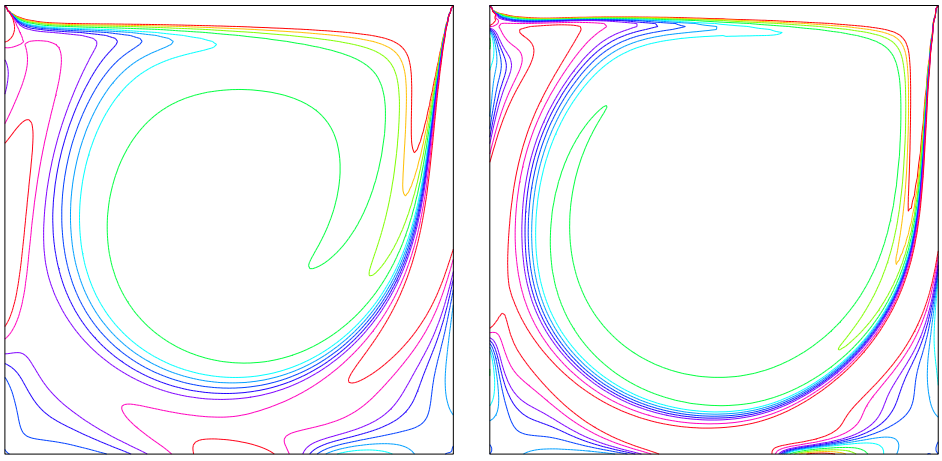


(c) $Re = 1000$ (d) $Re = 3200$

Figure 5.7 The square LDC fluid flow problem. Stream-function (ψ) contours of the flow for several Reynolds numbers $Re = \{100, 400, 1000, 3200\}$ by the present parallel method using 4 sub-domains for $Re = 100, 400, 1000$ and 2 sub-domains for $Re = 3200$ with the specifications: grid 151×151, $\Delta t = 10^{-3}$, $ABCM_{tol} = 10^{-6}$, $CM_{tol} = 10^{-6}$ and $\beta = 2$.

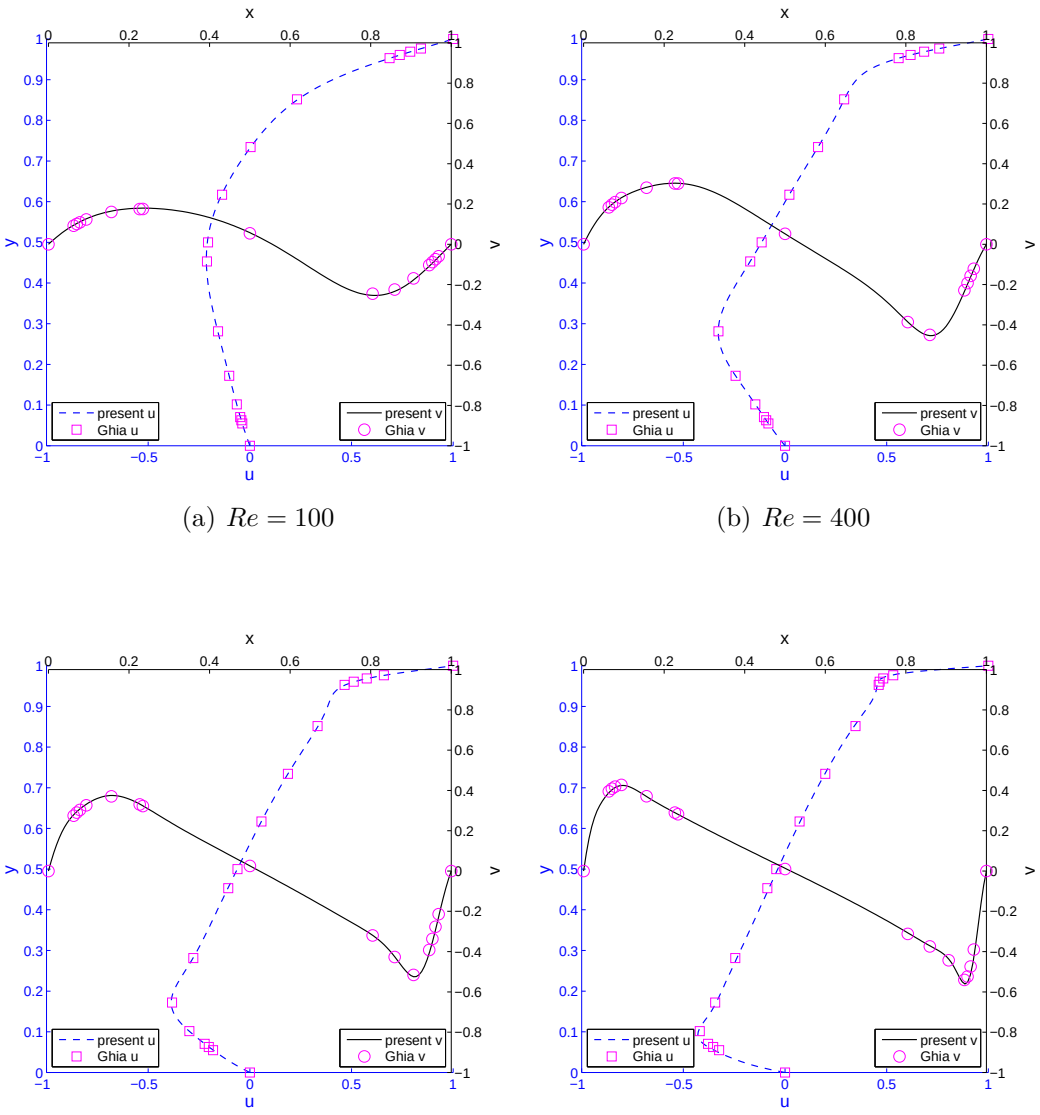


(a) $Re = 100$ (b) $Re = 400$



(c) $Re = 1000$ (d) $Re = 3200$

Figure 5.8 The square LDC fluid flow problem. Vorticity (ω) contours of the flow for several Reynolds numbers $Re = \{100, 400, 1000, 3200\}$ by the present parallel method using 4 sub-domains for $Re = \{100, 400, 1000\}$ and 2 sub-domains for $Re = 3200$. The other parameters are given in Fig. 5.7.



(a) $Re = 100$ (b) $Re = 400$

(c) $Re = 1000$ (d) $Re = 3200$

Figure 5.9 The square LDC fluid flow problem. Profiles of the u velocity along the vertical centreline and the v velocity along the horizontal centreline (solid lines) for several Reynolds numbers $Re = \{100, 400, 1000, 3200\}$ by the present parallel method in comparison with the corresponding Ghia's results (□ for u velocity and ○ for v velocity). The parameters of the present method are given in Fig. 5.7.

Table 5.2 The square LDC fluid flow problem. Comparison between parallel CV (P-CV) and parallel collocation (P-C) methods with $Re = 100$, grid 151×151, $dt = 10^{-3}$, $ABCM_{tol} = 10^{-6}, CM_{tol} = 10^{-6}, \beta = 2$. CPUs: number of CPUs; N_i: number of iterations; T_p: parallel computation time (minutes); S: speed-up; E: efficiency. The observed super-linear speed up can be explained in terms of reduced matrix condition numbers (see main text).

	P-CV method				P-C method			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
1	8814	132.25	1.00	100.00	8814	130.50	1.00	100.00
2	1578	127.34	1.04	51.93	1574	126.62	1.03	51.53
4	1687	63.88	2.07	51.76	1682	63.31	2.06	51.53
6	1703	42.83	3.09	51.47	1698	44.33	2.94	49.06
9	1727	27.74	4.77	52.98	1722	27.82	4.69	52.11
12	1719	17.11	7.73	64.42	1713	16.60	7.86	65.51
16	1710	10.36	12.77	79.82	1703	10.07	12.95	80.96
20	1699	7.81	16.93	84.64	1691	7.73	16.88	84.39
25	1683	6.13	21.59	86.34	1675	5.98	21.81	87.24
30	1629	4.34	30.47	101.57	1619	4.43	29.44	98.12
36	1544	3.32	39.87	110.75	1531	3.25	40.14	111.49
42	1494	2.82	46.89	111.64	1479	2.80	46.65	111.08
49	1456	2.33	56.65	115.61	1446	2.32	56.13	114.55
56	2150	2.78	47.49	84.81	2146	2.70	48.32	86.29
64	2687	2.73	48.48	75.75	2682	2.76	47.36	74.00
72	2816	2.68	49.39	68.60	2811	2.66	48.98	68.03
81	2937	2.49	53.11	65.57	2932	2.45	53.28	65.78
90	3038	2.33	56.83	63.15	3032	2.29	56.95	63.28
100	3144	2.11	62.75	62.75	3138	2.05	63.76	63.76
110	3916	2.53	52.36	47.60	3904	2.43	53.77	48.88
121	4111	2.57	51.47	42.54	4098	2.51	52.00	42.98
132	4306	2.34	56.45	42.76	4291	2.24	58.18	44.08
144	4459	2.42	54.65	37.95	4445	2.38	54.88	38.11

(Tables 5.2 - 5.5) for Re numbers 100, 400, 1000 and 3200, respectively using the grid of 151×151. Furthermore, the tendency of computational efficiency can be found similarly with the present parallel algorithm using the collocation method (named P-C method) (right hand side of the Tables 5.2 - 5.5).

The efficiency, speed-up and simulation time of the present parallel method can be seen visually in Figs. 5.10(a), 5.10(c) and 5.10(e). These figures also depict the influence of the Reynolds number on the mentioned criteria of the present parallel algorithm with respect to the number of CPUs. For example, the efficiency of the present parallel method is higher for the lower Re. While the throughput increases gradually with respect to the number of CPUs (Fig. 5.10(e)), the gradients of time curves decrease as the number of CPUs is more than around 20. This is also indicated by the efficiency curves given in Fig. 5.10(a). Similar trends of the efficiency, speed-up and throughput are also obtained by the present parallel algorithm using the collocation method and given in Figs. 5.10(b), 5.10(d) and 5.10(f).

It is observed that a super-linear speed-up is achieved using a range of numbers of CPUs 30, 36, 42 and 49 with corresponding efficiencies 101%, 110%, 111% and

Table 5.3 The square LDC fluid flow problem. Comparison between P-CV and P-C methods with $Re = 400$, grid 151×151. CPUs: number of CPUs; N_i: number of iterations; T_p: parallel computation time (minutes); S: speed-up; E: efficiency. Other parameters are given in Table 5.2.

	P-CV method				P-C method			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
1	22122	324.98	1.00	100.00	22107	338.55	1.00	100.00
2	3347	292.51	1.11	55.55	3340	240.24	1.41	70.46
4	3469	131.45	2.47	61.81	3459	130.46	2.60	64.88
6	3606	97.93	3.32	55.31	3593	90.65	3.73	62.25
9	3757	67.73	4.80	53.31	3742	58.94	5.74	63.83
12	3786	38.32	8.48	70.66	3768	34.32	9.86	82.21
16	3845	23.77	13.67	85.45	3826	23.05	14.69	91.80
20	3844	17.67	18.39	91.96	3824	17.85	18.96	94.81
25	3955	14.06	23.12	92.46	3932	13.90	24.35	97.41
30	4162	11.29	28.78	95.94	4134	10.69	31.68	105.60
36	4550	9.86	32.97	91.59	4520	9.11	37.16	103.23
42	4619	8.24	39.46	93.95	4589	8.02	42.24	100.56
49	4699	7.05	46.12	94.12	4671	7.14	47.42	96.78
56	4953	6.32	51.43	91.84	4917	6.07	55.79	99.62
64	5265	5.22	62.23	97.23	5218	5.09	66.47	103.86
72	5405	4.86	66.86	92.86	5345	4.80	70.58	98.03
81	5540	4.53	71.71	88.53	5471	4.49	75.36	93.04
90	5763	4.27	76.12	84.57	5694	4.14	81.73	90.81
100	5832	3.80	85.50	85.50	5758	3.74	90.51	90.51
110	5808	3.43	94.89	86.26	5753	3.46	97.90	89.00
121	5978	3.60	90.29	74.62	5919	3.55	95.48	78.91
132	6154	3.22	100.87	76.42	6053	3.22	105.05	79.58
144	6286	3.32	97.79	67.91	6222	3.30	102.73	71.34

Table 5.4 The square LDC fluid flow problem. Comparison between P-CV and P-C methods with $Re = 1000$, grid 151×151. CPUs: number of CPUs; N_i: number of iterations; T_p: parallel computation time (minutes); S: speed-up; E: efficiency. Other parameters are given in Table 5.2.

	P-CV method				P-C method			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
1	30536	453.65	1.00	100.00	30442	400.02	1.00	100.00
2	5016	429.95	1.06	52.76	5081	407.46	0.98	49.09
4	4763	178.57	2.54	63.51	4824	179.80	2.22	55.62
6	4661	121.38	3.74	62.29	4655	119.21	3.36	55.93
9	4684	72.85	6.23	69.19	4671	79.54	5.03	55.88
12	5214	51.39	8.83	73.56	5129	51.00	7.84	65.36
16	6211	37.53	12.09	75.55	6139	38.45	10.40	65.02
20	6498	29.17	15.55	77.75	6428	28.54	14.02	70.09
25	6765	22.87	19.84	79.36	6699	22.90	17.47	69.87
30	7452	18.38	24.69	82.29	7397	18.76	21.32	71.07
36	8580	17.78	25.52	70.89	8517	17.02	23.50	65.28
42	8709	15.09	30.07	71.60	8660	15.25	26.24	62.48
49	8593	13.35	33.98	69.34	8543	13.17	30.37	61.97
56	9299	11.47	39.57	70.65	9207	11.21	35.69	63.74
64	10750	10.61	42.76	66.81	10628	10.28	38.91	60.80
72	11542	10.21	44.45	61.73	11410	10.15	39.42	54.76
81	11922	9.39	48.32	59.66	11789	9.19	43.51	53.72
90	12122	8.42	53.85	59.83	11959	8.56	46.75	51.95
100	11637	7.32	62.01	62.01	11475	7.20	55.53	55.53
110	12025	7.01	64.68	58.80	11863	7.08	56.48	51.34
121	12315	7.23	62.77	51.88	12121	7.10	56.36	46.58
132	13058	6.69	67.76	51.33	12874	6.69	59.77	45.28
144	13359	6.76	67.12	46.61	13159	6.81	58.78	40.82

Table 5.5 The square LDC fluid flow problem. Comparison between P-CV and P-C methods with $Re = 3200$, grid 151×151. CPUs: number of CPUs; N_i: number of iterations; T_p: parallel computation time (minutes); S: speed-up; E: efficiency. Other parameters are given in Table 5.2.

	P-CV method				P-C method			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
1	69367	1003.93	1.00	100.00	69712	999.36	1.00	100.00
2	16057	1257.83	0.80	39.91	16590	1382.11	0.72	36.15
4	17139	622.24	1.61	40.34	16500	578.98	1.73	43.15
6	17033	439.62	2.28	38.06	16755	433.14	2.31	38.45
9	17023	289.41	3.47	38.54	17386	285.37	3.50	38.91
12	15755	154.36	6.50	54.20	15437	145.97	6.85	57.05
16	16630	101.29	9.91	61.95	16835	102.68	9.73	60.83
20	17404	78.81	12.74	63.69	17283	77.67	12.87	64.33
25	16376	57.55	17.45	69.78	15759	55.19	18.11	72.43
30	15305	39.14	25.65	85.50	14071	36.78	27.17	90.56
36	19120	39.43	25.46	70.72	17046	35.32	28.29	78.59
42	16089	27.98	35.88	85.43	16799	29.59	33.77	80.40
49	17182	25.98	38.64	78.86	17667	26.87	37.19	75.90
56	22360	27.24	36.86	65.82	22660	27.31	36.60	65.35
64	26652	25.62	39.19	61.23	26790	25.65	38.96	60.87
72	27080	24.21	41.47	57.60	27573	24.45	40.88	56.77
81	28656	22.26	45.09	55.67	29246	22.39	44.64	55.11
90	31817	22.13	45.37	50.41	31127	21.51	46.46	51.62
100	32548	19.76	50.80	50.80	31885	19.68	50.77	50.77
110	33739	19.18	52.36	47.60	32418	18.82	53.11	48.28
121	26585	16.46	61.01	50.42	33365	19.68	50.78	41.97
132	34222	16.60	60.49	45.83	33334	16.86	59.27	44.90
144	35285	17.35	57.87	40.19	34716	17.44	57.29	39.79

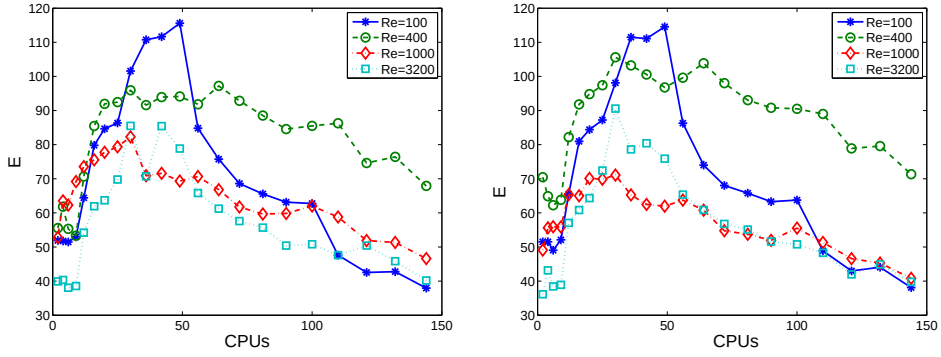
Table 5.6 The square LDC fluid flow problem. Condition numbers CN_ω and CN_ψ in single and parallel solutions with $Re = 100$ and grid = 151×151. CPUs: number of CPUs (sub-domains).

CPUs	CN_ω	CN_ψ	CPUs	CN_ω	CN_ψ
1	2.6341	1.29E+04	49	1.1581	3.02E+02
2	1.1799	5.33E+03	56	1.1581	2.44E+02
4	1.1581	3.49E+03	64	1.1581	2.06E+02
6	1.1581	2.18E+03	72	1.1581	1.83E+02
9	1.1581	1.61E+03	81	1.1581	1.64E+02
12	1.1581	1.12E+03	90	1.1581	1.44E+02
16	1.1581	8.71E+02	100	1.1567	1.28E+02
20	1.1581	6.99E+02	110	1.1581	1.19E+02
25	1.1581	5.87E+02	121	1.1581	1.12E+02
30	1.1581	4.65E+02	132	1.1567	1.03E+02
36	1.1581	3.87E+02	144	1.1567	9.56E+01
42	1.1581	3.39E+02			

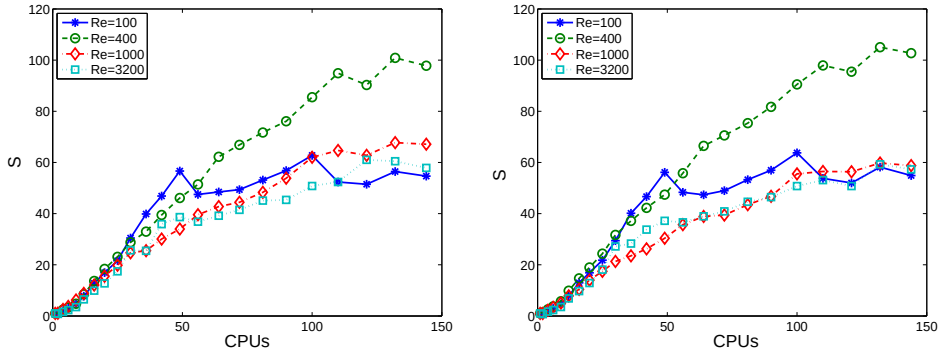
115% for $Re = 100$ (Table 5.2). This is an exclusive behaviour and sometimes controversial in classical parallel computing, when the speed-up is higher than the number of CPUs used in a parallel algorithm. For these cases, the super-linear speed-up is considered to be related to the decrease of condition number of each sub-domain which plays a crucial role for the stability of a numerical method. Indeed, by decomposing the domain, sub-problem in each sub-domain is not only smaller in terms of DoF but also has smaller condition number (see Table 5.6).

The efficiency of the algorithm in large scale problems is also investigated. For testing purposes, the fluid with $Re = 1000$ is simulated using very fine grids including grid-1 401×401 and grid-2 601×601 with the following parameters $ABCM_{tol} = 10^{-6}$, $CM_{tol} = 10^{-6}$, $\beta = 2$ and $\Delta t = 10^{-3}$ for grid-1 and 5×10^{-4} for grid-2.

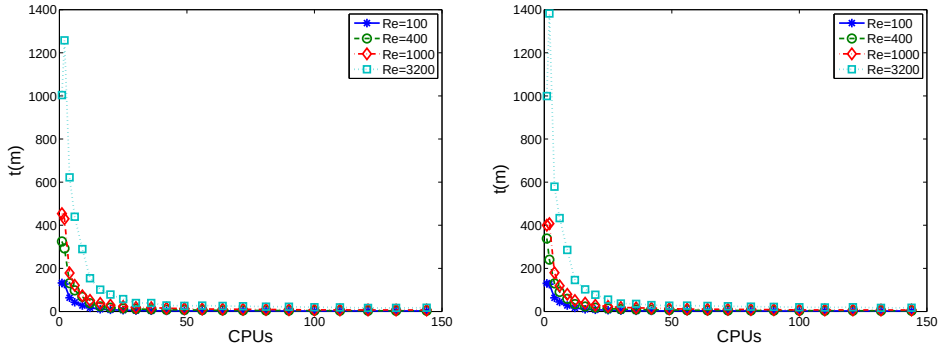
While Fig. 5.11(a) points out a gradual increase of throughput with respect to the number of CPUs for different scales by the present P-CV method, Fig. 5.11(b) depicts the influence of the grid density on the efficiency with respect to the number of CPUs. Indeed, the gradient of time curves of finer grid-size is steeper, which again indicates that the efficiency of the present parallel method will be higher for larger scale problems.



(a) P-CV method - efficiency (b) P-C method - efficiency

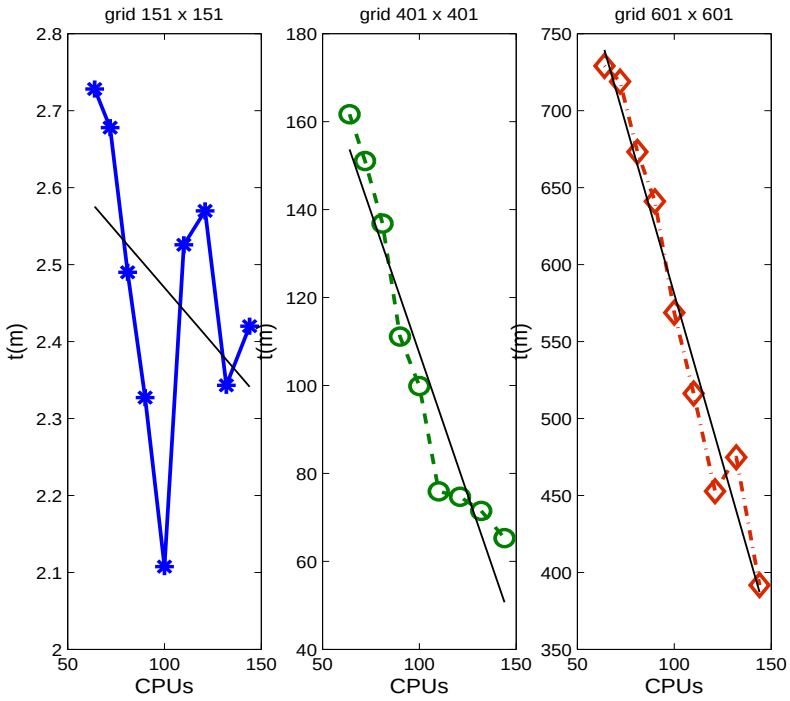


(c) P-CV method - speed-up (d) P-C method - speed-up

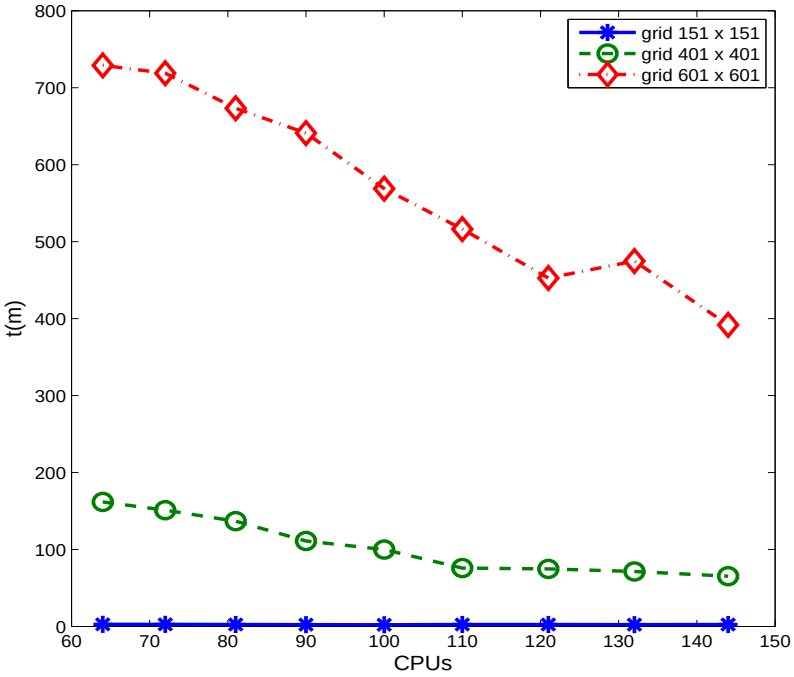


(e) P-CV method - simulation time (f) P-C method - simulation time

Figure 5.10 The square LDC fluid flow problem. Comparison between the parallel performance of the P-C and P-CV methods for several Reynolds numbers ($Re = 100, 400, 1000$ and 3200) with a grid of 151×151: the efficiency, speed-up and simulation time of the two methods as a function of the number of CPUs. Other parameters are given in Tables 5.2 - 5.5.



(a)



(b)

Figure 5.11 The square LDC fluid flow problem. Simulation time of the P-CV method with $Re = 1000$ using different grids: 151×151, 401×401 and 601×601 as a function of the number of CPUs.

5.5.2 Triangular lid-driven cavity fluid flow problem

The triangular LDC fluid flow problem has been proposed as a test case for the numerical algorithm in the case of non-rectangular domain. The domain is an equilateral triangle with the left and right sides being fixed and the top side or the lid, moving at a constant velocity from left to right. The problem's geometry and boundary condition can be seen visually in Fig. 5.12.

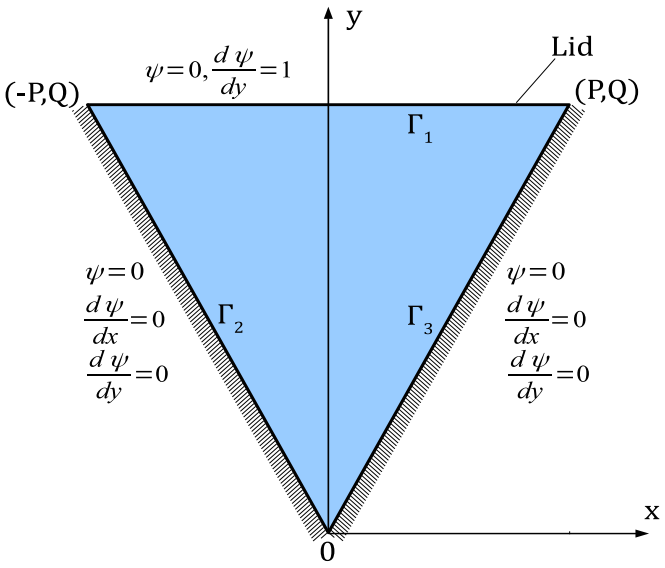


Figure 5.12 Triangular LDC flow problem. Geometry and boundary conditions. $P = \sqrt{3}, Q = 3$. No slip is assumed between the fluid and solid surfaces. The top lid is moving from left to right with a speed of 1.

It is noted that while implementing CVs with non-rectangular domains, one needs to take extra care regarding points closed to boundary to make sure that CVs do not intersect with each other nor with the boundary. Fig. 5.13 shows an example of CV formation for a triangular domain.

As stated in section 5.4.1, some difficulties may arise when the present algorithm is applied to problems with a non-rectangular boundary. In such cases it is difficult to maintain even loads between sub-domains. Indeed, in the case of non-rectangular domains, the domain decomposition is manually handled to avoid bad load imbalance. This process might take considerable time and effort. Fortunately, the decomposition needs to be carried out only once for all simulations. Nevertheless, no matter how well the manual formation of sub-domains is performed, it always exists a load imbalance which explains the poorer performance of the present algorithm in the case of non-rectangular domains.

The boundary conditions are given in terms of the stream-function as

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 1 \quad \forall (x, y) \in \Gamma_1; \tag{5.22}$$

$$\psi = 0, \quad \frac{\partial \psi}{\partial x} = 0, \quad \frac{\partial \psi}{\partial y} = 0 \quad \forall (x, y) \in \Gamma_2 \cup \Gamma_3, \tag{5.23}$$

where the variables are defined before. The solving procedure remains the same

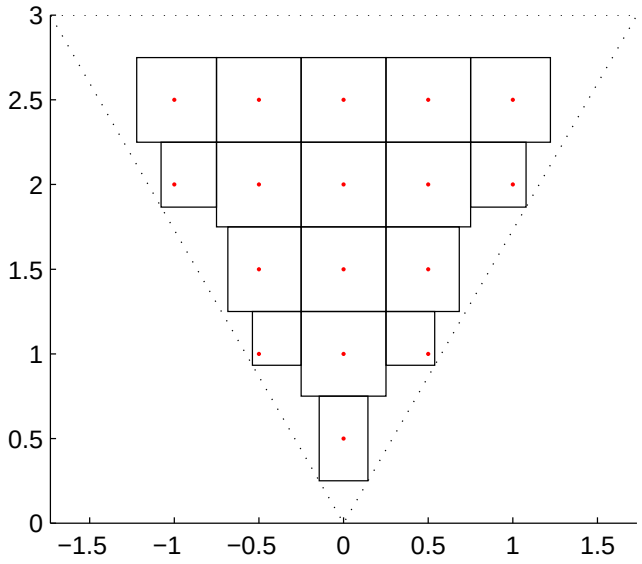


Figure 5.13 CV formation in 2D.

as for the square cavity problem. However, when approximate the boundary value for ω two following cases must be considered.

First, for boundary points that lie on both x grid-line and y grid-line the approximation can be carried out normally by using 1D-IRBF in two directions following Eq. (5.18).

Second, for boundary points, that lie only on x grid-line or y grid-line, its approximation, thus, is available only in one direction. In this case, equivalent formulas provided by Le-Cao et al. (2009) are used as follows.

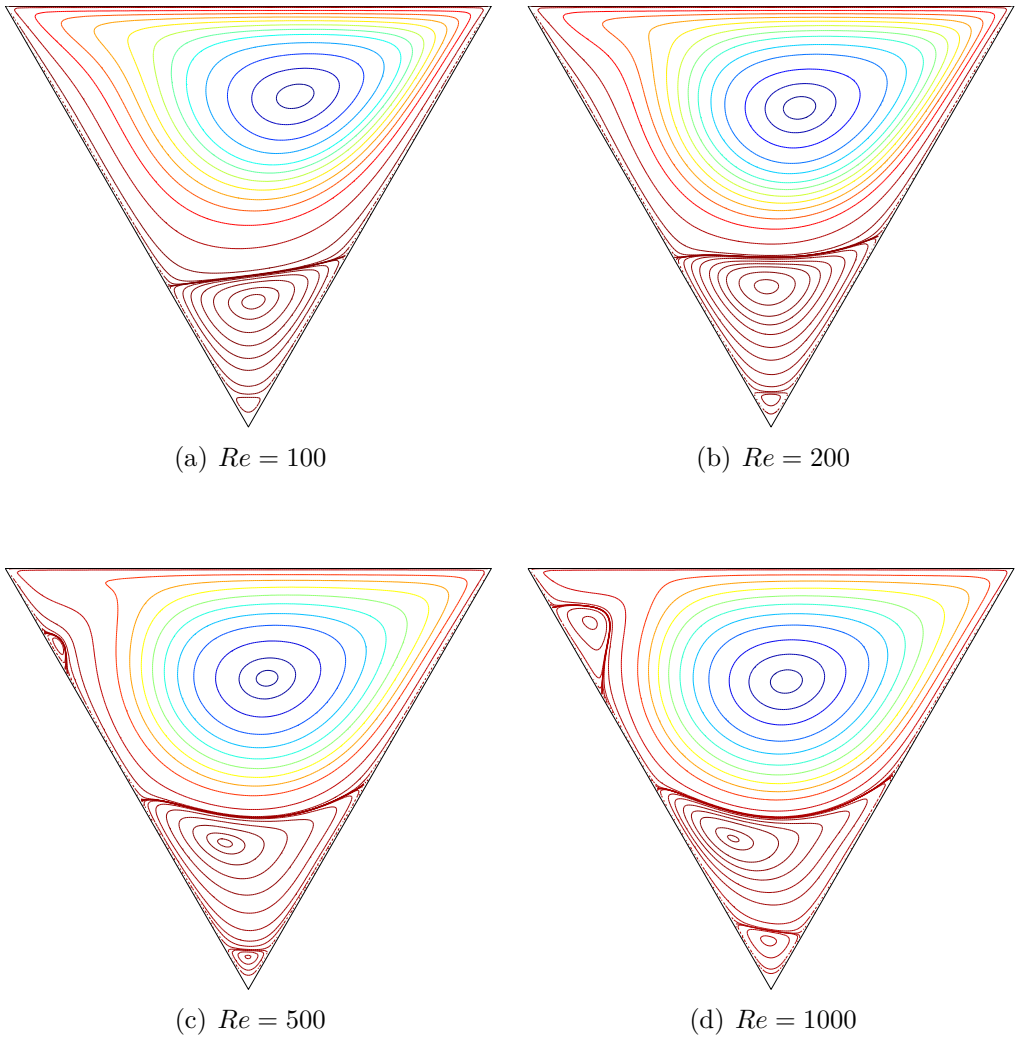
$$\omega_b = -\left[1 + \left(\frac{t_x}{t_y}\right)^2\right] \frac{\partial^2 \psi_b}{\partial x^2}, \tag{5.24}$$

for points on x-grid line and

$$\omega_b = -\left[1 + \left(\frac{t_y}{t_x}\right)^2\right] \frac{\partial^2 \psi_b}{\partial y^2}, \tag{5.25}$$

for points on y-grid line, where t_x and t_y are the x- and y-components of the unit vector tangential to the boundary.

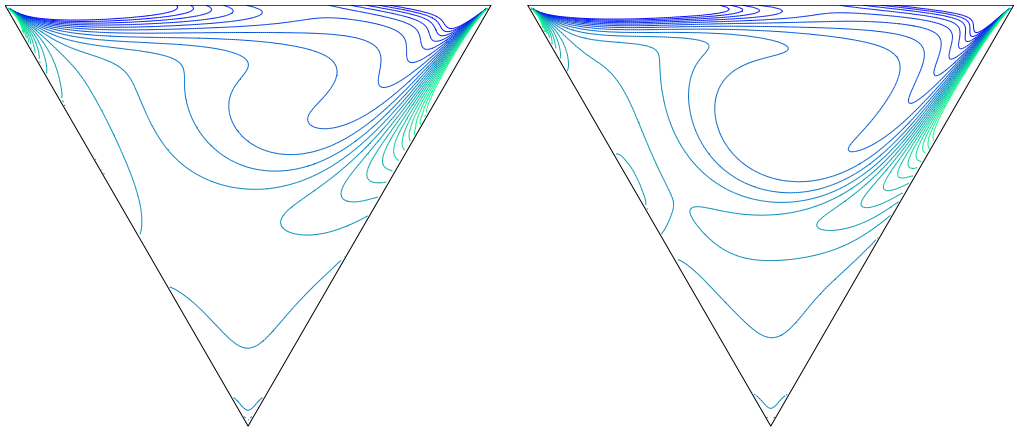
In this chapter, a range of Reynolds numbers $\{100, 200, 500, 1000\}$ is investigated. Again, the streamline (Fig. 5.14), vorticity contours (Fig. 5.15) and velocity profiles along central horizontal line $y = 2$ and vertical line $x = 0$ (Fig. 5.16) by the present parallel method with 4 sub-domains agree very well with ones by Kohno and Bathe (2006) using flow-conditioned-based finite element method.



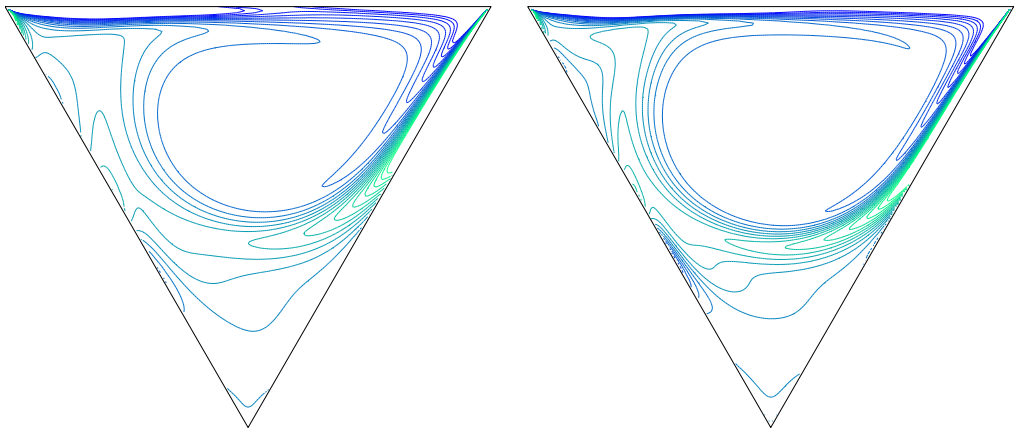
(a) $Re = 100$ (b) $Re = 200$

(c) $Re = 500$ (d) $Re = 1000$

Figure 5.14 The triangular LDC fluid flow problem. Stream-function (ψ) contours of the flow for several Reynolds numbers by the present parallel method using 4 sub-domains with grid of 24697 points, $\Delta t = 5 \times 10^{-4}$, $ABCM_{tol} = 10^{-6}$, $CM_{tol} = 10^{-6}$ and $\beta = 1$.



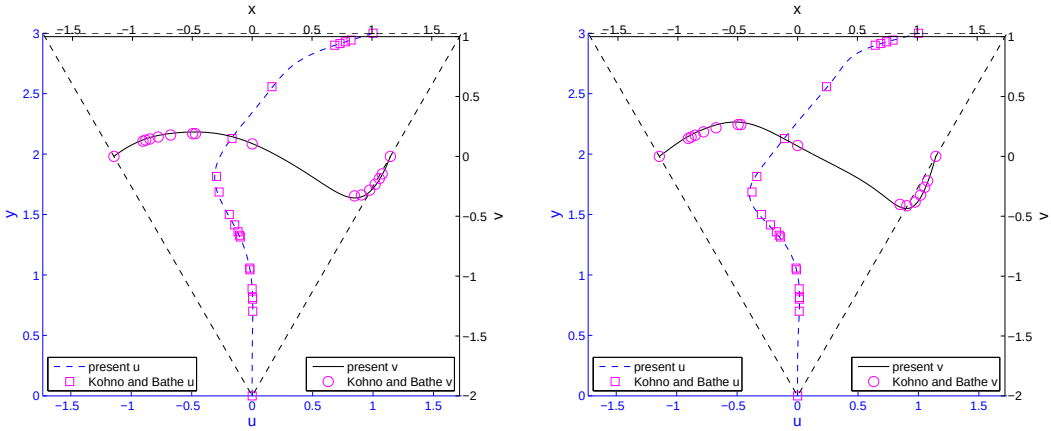
(a) $Re = 100$ (b) $Re = 200$



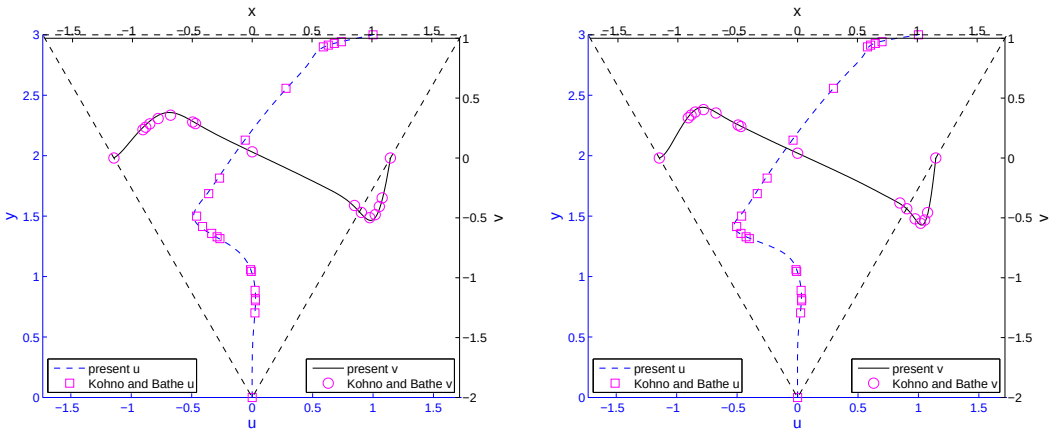
(c) $Re = 500$ (d) $Re = 1000$

Figure 5.15 The triangular LDC fluid flow problem. Vorticity (ω) contours of the flow for several Reynolds numbers by the present parallel method. Other parameters are given in Fig. 5.14.

.



(a) $Re = 100$ (b) $Re = 200$



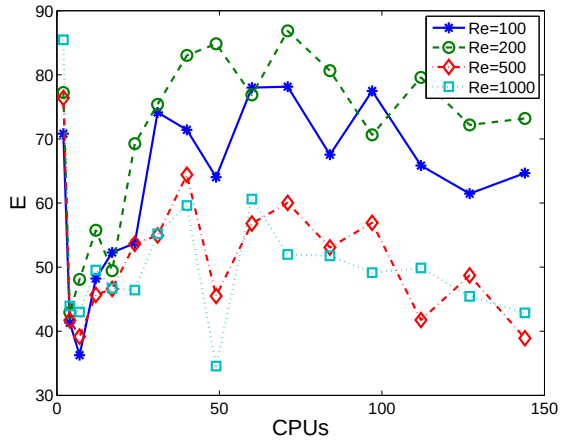
(c) $Re = 500$ (d) $Re = 1000$

Figure 5.16 The triangular LDC fluid flow problem. Vorticity profiles along vertical line ($x = 0$) and horizontal line ($y = 2$) for several Reynolds numbers by the present parallel method in comparison with the corresponding Kohno and Bathe's results (□ for u velocity and ○ for v velocity). Other parameters of the present method are given in Fig. 5.14.

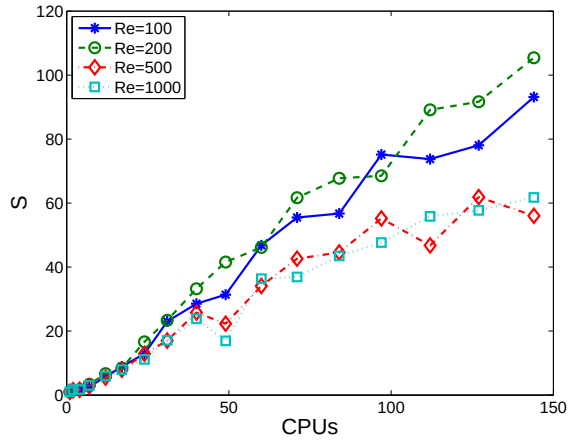
Table 5.7 The triangular LDC fluid flow problem. Results by the present P-CV method with grid of 24697 points, $\Delta t = 5.E-04$, $ABCM_{tol} = 10^{-6}$, $CM_{tol} = 10^{-6}$, $\beta = 1$. CPUs: number of CPUs; N_i: number of iterations; T_p: parallel computation time (minutes); S: speed-up; E: efficiency.

	$Re = 100$				$Re = 200$				$Re = 500$				$Re = 1000$			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E	N_i	T_p	S	E	N_i	T_p	S	E
1	47800	848.12	1.00	100	86584	1493.69	1.00	100	72673	1313.30	1.00	100	83900	1460.14	1.00	100
2	10352	598.72	1.42	70.83	15758	966.92	1.54	77.24	14362	859.59	1.53	76.39	14173	854.31	1.71	85.46
4	11643	513.13	1.65	41.32	18926	869.66	1.72	42.94	17434	786.89	1.67	41.72	18679	830.83	1.76	43.94
7	13247	334.09	2.54	36.27	18286	443.81	3.37	48.08	18898	478.99	2.74	39.17	19413	485.07	3.01	43.00
12	12594	146.53	5.79	48.23	18908	223.26	6.69	55.75	20373	239.64	5.48	45.67	20370	245.55	5.95	49.55
17	12511	95.40	8.89	52.29	23245	177.67	8.41	49.45	22050	165.74	7.92	46.61	24900	183.87	7.94	46.71
24	13729	65.77	12.90	53.73	19644	89.86	16.62	69.26	21977	102.03	12.87	53.63	28436	131.05	11.14	46.42
31	12912	36.90	22.98	74.13	21156	63.92	23.37	75.38	26608	77.11	17.03	54.94	28122	85.33	17.11	55.20
40	12695	29.69	28.57	71.42	19745	44.97	33.21	83.03	22067	50.98	25.76	64.41	26965	61.23	23.85	59.62
49	15267	27.03	31.38	64.04	20648	35.93	41.57	84.84	34223	58.90	22.30	45.51	49217	86.22	16.93	34.56
60	13267	18.12	46.80	78.01	24210	32.38	46.13	76.88	28281	38.53	34.08	56.81	29830	40.14	36.38	60.63
71	12884	15.29	55.48	78.14	20574	24.22	61.67	86.86	26316	30.81	42.62	60.03	34076	39.57	36.90	51.98
84	15687	14.95	56.73	67.54	22934	22.05	67.75	80.65	30586	29.46	44.57	53.06	35805	33.58	43.48	51.76
97	13683	11.28	75.16	77.49	25704	21.80	68.52	70.64	28596	23.79	55.21	56.91	36386	30.63	47.67	49.14
112	14760	11.50	73.75	65.85	21280	16.76	89.14	79.59	36899	28.09	46.76	41.75	33732	26.14	55.86	49.87
127	16187	10.86	78.06	61.47	24681	16.29	91.68	72.19	32695	21.23	61.87	48.71	38666	25.30	57.71	45.44
144	14810	9.11	93.13	64.68	22833	14.17	105.41	73.20	37874	23.44	56.03	38.91	38753	23.65	61.75	42.88

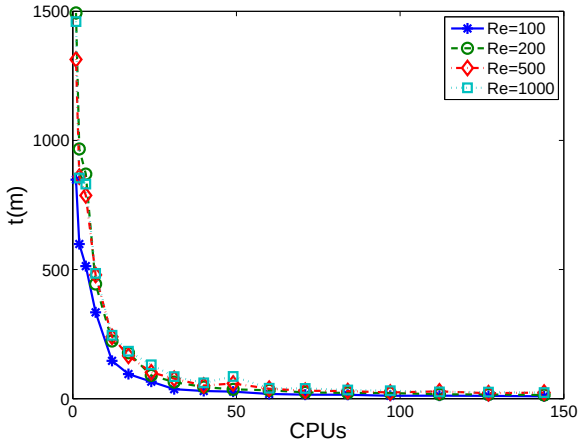
In terms of parallel efficiency, Table 5.7 gives detailed results of the parallel algorithm for several Reynolds numbers and with a grid of 205×205 (or 24697 grid points). Visual forms can be found in Fig. 5.17. For each Reynolds number, although results showed that the simulation time decreases gradually as the number of sub-domains (CPUs) increases (Fig. 5.17(c)), the optimum number of CPUs (CPU_{opt}) of the parallel method described by the efficiency for each case is not clear (Fig. 5.17(a)). This can be explained as the influence of the DD for a non-rectangular domain problem where the numbers of collocation points/CVs in sub-domains are not equal, resulting in significant variation of the amount of work to be completed from sub-domain to sub-domain. Thus, the results show that the sub-domain formation plays an important role in parallel computation of complex domains.



(a) Efficiency



(b) Speed-up



(c) Simulation time

Figure 5.17 The triangular LDC fluid flow problem. Parallel performance of the P-CV methods for several Re using a grid of 24607 points: the efficiency, speed-up and simulation time as a function of the number of CPUs. Other parameters are given in Table 5.7.

5.6 Conclusion

In this chapter, we proposed a DD parallel distributed method coupled with a local IRBF-CV approach. The proposed method is successfully implemented to simulate the LDC flow in rectangular and non-rectangular domains. It has been shown that results produced by the method are in excellent agreement with the spectral benchmark solutions by Botella and Peyret (1998) and Ghia et al. (1982) for the square domain and by Kohno and Bathe (2006) for the triangular domain. A very important achievement of this chapter is the high time-efficiency of the parallel algorithm including the speed-up. It is shown that the speed-up grows steadily with the increase of the number of CPUs. This indicates excellent scalability of the method. Moreover, a super-linear efficiency has been observed for several cases; this phenomenon is best explained by the decrease of condition numbers of sub-domain matrices. The parallel algorithm performs well in both collocation and CV methods. Indeed, the trend in efficiency with increasing number of CPUs for several Reynolds numbers is consistent with results achieved by the collocation method reported in Chapter 4.

Chapter 6

Compact local integrated RBF based parallel non-overlapping DD approach

In previous chapters, the overlapping Domain Decomposition (DD) method has been investigated and parallel algorithms based on this scheme have been successfully implemented. Although overlapping DD method has the benefit of simple implementation and intuitive presentation, it is not very suitable for problem with complex boundaries. As presented in Chapter 5, for the triangular lid-driven cavity (LDC) fluid flow problem the efficiency sometime compares less favourably with those of the square LDC flow problem. This was probably due to the difficulty associated with load balancing in non-rectangular domains. Furthermore, a drawback of overlapping DD method is a significant additional degrees of freedom (DoF) produced in each and every sub-domain due to the overlapping volumes. This chapter develops a parallel algorithm based on the non-overlapping DD technique and the compact local Integrated Radial Basis Function (CLIRBF) method for solving Navier-Stokes equations (Pham-Sy et al., 2015b). The classical non-overlapping Dirichlet-Neumann DD is used to divide the main problem into sub-domains. In each sub-domain, a compact local IRBF scheme is used to solve the incompressible Navier-Stokes equation. A relaxation factor is applied for the transmission conditions at the interfaces between sub-domains to ensure the convergence of the DD method while a termination detection algorithm is employed to achieve the global termination. The parallel algorithm is verified through two common problems: the LDC fluid flow and the natural convection (NC) in concentric annuli. The results confirm the efficiency of the present method in comparison with a sequential algorithm. Super-linear speed-up is found for a range of numbers of CPUs. In terms of accuracy, the obtained results are in a good agreement with benchmark results.

6.1 Introduction

Domain decomposition method was first proposed by Schwarz (1869) as a technique to solve classical boundary value problems involving harmonic functions.

This method is called Schwarz alternating DD method. The idea is to split a problem domain into smaller sub-domains and solve the problem in each sub-domain separately. Boundary conditions (BCs) on the artificial boundaries (ABs) are of Dirichlet type and are updated by the obtained values from neighbouring sub-domains (NSs). This Schwarz DD method belongs to a group of overlapping DD methods (Smith et al., 1996) which are characterised by the fact that adjacent sub-domains must overlap with each other. This fact causes difficulties in solving problems with irregular domains and adds the overhead of DoF to each sub-domain.

Non-overlapping DD methods belong to another group which can be categorised into two sub-classes, namely the Schur complement method and the Steklov-Poincare method (Quarteroni and Valli, 1999). The idea of Schur complement method, which was proposed by Przemieniecki in 1963 and demonstrated by Haynsworth (1968), is to divide a structure into substructures with the assumption that all ABs are fixed (Gander and Tu, 2014). The actual value of field variables on ABs is determined from the equilibrium equations of forces. The problem is then solved separately in substructures. Although the Schur method allows high degree of parallelisation, the cost of obtaining solutions on ABs is huge in comparison with the benefit by the parallel calculation on substructures. This is the biggest drawback of the method in solving large-scale problems. On the other hand, the Steklov-Poincare method focuses on constructing an equivalent problem by introducing transmission conditions. The newly formed problem is then defined and solved separately in sub-domains. Obtained results are finally gathered back into the original domain. This method is more economical than the overlapping DD one because the DoF overhead is minimised (Tran et al., 2009). It also has high potential for parallelisation where all sub-domains can run concurrently from the beginning without causing the bottle-neck situation as in the Schur complement method. Although the method does not originally include parallelisation, only some small modifications are needed to make it parallel. More details on this method will be presented in Section 6.3 as a part of the present work.

Integrated radial basis function has been a well known approximation method owing to its high convergence rate (Mai-Duy and Tran-Cong, 2001). However, its stability degrades relatively fast with respect to grid density because of the ill-condition of the corresponding system matrix. This undesired property of IRBF is mitigated by introducing compact local schemes. It has been shown in Hoang-Trieu et al. (2012) that by appropriately choosing the shape parameter β, compact scheme can achieve a very high convergence rate while maintain an acceptable condition number of the system matrix.

In this chapter, the Steklov-Poincare non-overlapping DD approach and the CLIRBF method are combined into one parallel algorithm. The obtained algorithm is used to solve two fluid flow problems, namely the lid-driven cavity fluid flow and natural convection in concentric annuli. The accuracy and efficiency of the algorithm will be investigated and compared with those of the non-parallel method as well as with benchmark results.

6.2 Review of compact local 2D-IRBF method

6.2.1 2D-IRBF method

Consider a 2D elliptic problem

$$\begin{cases} \mathcal{L}u(\mathbf{x}) &= f, \quad \mathbf{x} \in \Omega \\ \mathcal{B}u(\mathbf{x}) &= g, \quad \mathbf{x} \in \partial\Omega \end{cases} \tag{6.1}$$

where $u(\mathbf{x})$ is an unknown function; $\mathbf{x}$ the position vector; $\mathcal{L}$ the second order differential operator; $\mathcal{B}$ the BCs; f and g known functions of $\mathbf{x}$; $\partial\Omega$ the boundary; and Ω the considered domain.

By the 2D-IRBF discretisation approach (Mai-Duy and Tran-Cong, 2010), the second derivatives of function u are expressed as a combination of RBFs

$$\frac{\partial^2 u(\mathbf{x})}{\partial x_j^2} = \sum_{i=1}^{n} w_i g_i(\mathbf{x}) = \sum_{i=1}^{n} w_i G_i^{[2]}(\mathbf{x}), \tag{6.2}$$

where x_j is the j-component of $\mathbf{x}$ ($j = 1, 2$); $\{w_i\}_{i=1}^n$ the set of weights; and $\{g_i(\mathbf{x})\}_{i=1}^n$ the set of RBFs associated with n centres. In our work, the centres are chosen to be the grid points. The Multiquadric RBF (MQ-RBF) function is used in this work and given by

$$G_i^{[2]}(\mathbf{x}) = \sqrt{(\mathbf{x} - \mathbf{c}_i)^2 + a_i^2},$$

where $\{\mathbf{c}_i\}_{i=1}^n$ and $\{a_i\}_{i=1}^n$ are MQ-RBF centres and widths, respectively. It is noted that superscript $^{[.]}$ is used to denote the associated derivative order.

To obtain expressions for the first-order derivative and the function value, expression (6.2) is integrated successively with respect to x_j as follows.

$$\frac{\partial u}{\partial x_j} = \sum_{i=1}^{n} w_i G_i^{[1]}(\mathbf{x}) + C_1, \tag{6.3}$$

$$u = \sum_{i=1}^{n} w_i G_i^{[0]}(\mathbf{x}) + C_1 x_j + C_2, \tag{6.4}$$

where $G_i^{[1]}(\mathbf{x}) = \int G_i^{[2]}(\mathbf{x}) dx_j$, $G_i^{[0]}(\mathbf{x}) = \int G_i^{[1]}(\mathbf{x}) dx_j$ and C_1, and C_2 are constants of integration in the sense that $C_i = C_i(x_k)$, $k \neq j$.

Collocating equations (6.2) - (6.4) at the grid points $\{\mathbf{x}_i\}_{i=1}^n$ yields

$$\frac{\partial^2 \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}_{x_j}^{[2]} \widetilde{\mathbf{w}}_{x_j}, \tag{6.5}$$

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial x_j} = \mathcal{G}_{x_j}^{[1]} \widetilde{\mathbf{w}}_{x_j}, \tag{6.6}$$

$$\widetilde{\mathbf{u}} = \mathcal{G}_{x_j}^{[0]} \widetilde{\mathbf{w}}_{x_j}, \tag{6.7}$$

with

$$\begin{aligned}\widetilde{\mathbf{w}}_{x_j} &= (w_1, w_2, \cdots, w_n, C_1, C_2)^T_{x_j}, \\ \widetilde{\mathbf{u}} &= (u_1, u_2, \cdots, u_n)^T, \\ \frac{\partial^k \widetilde{\mathbf{u}}}{\partial x_j^k} &= \left(\frac{\partial^k u_1}{\partial x_j^k}, \frac{\partial^k u_2}{\partial x_j^k}, \cdots, \frac{\partial^k u_n}{\partial x_j^k}\right)^T,\end{aligned}$$

where $u_i = u(\mathbf{x}_i)$ $(i = 1, 2, \cdots, n)$; $\mathcal{G}^{[2]}_{x_j}, \mathcal{G}^{[1]}_{x_j}$ and $\mathcal{G}^{[0]}_{x_j}$ are known matrices. In the above equations, the subscript x_j denotes the quantity associated with the integration process in the x_j dimension.

It is worth noting that, in a 2D problem Eq. (6.4) produces two approximations for the function u and these two approximations will be naturally forced to be identical. In the following sections, x and y notations will be used to represent dimensions instead of x_1 and x_2 and $\mathbf{x}_k$ to represent a grid point k.

6.2.2 Compact local IRBF scheme

In local schemes, a local set of points called stencil is used for discretisation. For 2D-IRBF, a well-known 9-point stencil is shown in Fig. 6.1.

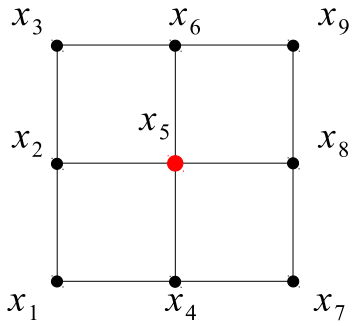


Figure 6.1 2D 9-point stencil with $\mathbf{x}_5$ is the point under consideration.

For this stencil, Eq. (6.5) - (6.7) is rewritten explicitly for the x-dimension as

$$\frac{\partial^2 \widetilde{\mathbf{u}}}{\partial x} = \mathcal{G}^{[2]}_x \widetilde{\mathbf{w}}_x, \tag{6.8}$$

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial x} = \mathcal{G}^{[1]}_x \widetilde{\mathbf{w}}_x, \tag{6.9}$$

$$\widetilde{\mathbf{u}} = \mathcal{G}^{[0]}_x \widetilde{\mathbf{w}}_x, \tag{6.10}$$

where

$$\mathcal{G}_x^{[2]} = \begin{bmatrix} G_{x1}^{[2]}(\mathbf{x}_1), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_1), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_2), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_2), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_3), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_3), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_4), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_4), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_5), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_5), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_6), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_6), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_7), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_7), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_8), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_8), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[2]}(\mathbf{x}_9), & \cdots, & G_{x9}^{[2]}(\mathbf{x}_9), & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}, \tag{6.11}$$

$$\mathcal{G}_x^{[1]} = \begin{bmatrix} G_{x1}^{[1]}(\mathbf{x}_1), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_1), & 1, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_2), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_2), & 0, & 1, & 0, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_3), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_3), & 0, & 0, & 1, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_4), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_4), & 1, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_5), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_5), & 0, & 1, & 0, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_6), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_6), & 0, & 0, & 1, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_7), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_7), & 1, & 0, & 0, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_8), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_8), & 0, & 1, & 0, & 0, & 0, & 0 \\ G_{x1}^{[1]}(\mathbf{x}_9), & \cdots, & G_{x9}^{[1]}(\mathbf{x}_9), & 0, & 0, & 1, & 0, & 0, & 0 \end{bmatrix}, \tag{6.12}$$

$$\mathcal{G}_x^{[0]} = \begin{bmatrix} G_{x1}^{[0]}(\mathbf{x}_1), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_1), & x_1, & 0, & 0, & 1, & 0, & 0 \\ G_{x1}^{[0]}(\mathbf{x}_2), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_2), & 0, & x_2, & 0, & 0, & 1, & 0 \\ G_{x1}^{[0]}(\mathbf{x}_3), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_3), & 0, & 0, & x_3, & 0, & 0, & 1 \\ G_{x1}^{[0]}(\mathbf{x}_4), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_4), & x_1, & 0, & 0, & 1, & 0, & 0 \\ G_{x1}^{[0]}(\mathbf{x}_5), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_5), & 0, & x_2, & 0, & 0, & 1, & 0 \\ G_{x1}^{[0]}(\mathbf{x}_6), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_6), & 0, & 0, & x_3, & 0, & 0, & 1 \\ G_{x1}^{[0]}(\mathbf{x}_7), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_7), & x_1, & 0, & 0, & 1, & 0, & 0 \\ G_{x1}^{[0]}(\mathbf{x}_8), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_8), & 0, & x_2, & 0, & 0, & 1, & 0 \\ G_{x1}^{[0]}(\mathbf{x}_9), & \cdots, & G_{x9}^{[0]}(\mathbf{x}_9), & 0, & 0, & x_3, & 0, & 0, & 1 \end{bmatrix}, \tag{6.13}$$

and the y-dimension as

$$\frac{\partial^2 \widetilde{\mathbf{u}}}{\partial y} = \mathcal{G}_y^{[2]} \widetilde{\mathbf{w}}_y, \tag{6.14}$$

$$\frac{\partial \widetilde{\mathbf{u}}}{\partial y} = \mathcal{G}_y^{[1]} \widetilde{\mathbf{w}}_y, \tag{6.15}$$

$$\widetilde{\mathbf{u}} = \mathcal{G}_y^{[0]} \widetilde{\mathbf{w}}_y, \tag{6.16}$$

where

$$\mathcal{G}_y^{[2]} = \begin{bmatrix} G_{y1}^{[2]}(\mathbf{x}_1), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_1), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_2), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_2), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_3), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_3), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_4), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_4), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_5), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_5), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_6), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_6), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_7), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_7), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_8), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_8), & 0, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[2]}(\mathbf{x}_9), & \cdots, & G_{y9}^{[2]}(\mathbf{x}_9), & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}, \tag{6.17}$$

$$\mathcal{G}_y^{[1]} = \begin{bmatrix} G_{y1}^{[1]}(\mathbf{x}_1), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_1), & 1, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_2), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_2), & 1, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_3), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_3), & 1, & 0, & 0, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_4), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_4), & 0, & 1, & 0, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_5), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_5), & 0, & 1, & 0, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_6), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_6), & 0, & 1, & 0, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_7), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_7), & 0, & 0, & 1, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_8), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_8), & 0, & 0, & 1, & 0, & 0, & 0 \\ G_{y1}^{[1]}(\mathbf{x}_9), & \cdots, & G_{y9}^{[1]}(\mathbf{x}_9), & 0, & 0, & 1, & 0, & 0, & 0 \end{bmatrix}, \tag{6.18}$$

$$\mathcal{G}_y^{[0]} = \begin{bmatrix} G_{y1}^{[0]}(\mathbf{x}_1), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_1), & y_1, & 0, & 0, & 1, & 0, & 0 \\ G_{y1}^{[0]}(\mathbf{x}_2), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_2), & y_2, & 0, & 0, & 1, & 0, & 0 \\ G_{y1}^{[0]}(\mathbf{x}_3), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_3), & y_3, & 0, & 0, & 1, & 0, & 0 \\ G_{y1}^{[0]}(\mathbf{x}_4), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_4), & 0, & y_1, & 0, & 0, & 1, & 0 \\ G_{y1}^{[0]}(\mathbf{x}_5), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_5), & 0, & y_2, & 0, & 0, & 1, & 0 \\ G_{y1}^{[0]}(\mathbf{x}_6), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_6), & 0, & y_3, & 0, & 0, & 1, & 0 \\ G_{y1}^{[0]}(\mathbf{x}_7), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_7), & 0, & 0, & y_1, & 0, & 0, & 1 \\ G_{y1}^{[0]}(\mathbf{x}_8), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_8), & 0, & 0, & y_2, & 0, & 0, & 1 \\ G_{y1}^{[0]}(\mathbf{x}_9), & \cdots, & G_{y9}^{[0]}(\mathbf{x}_9), & 0, & 0, & y_3, & 0, & 0, & 1 \end{bmatrix}. \tag{6.19}$$

The conversion matrix is constructed as

$$\begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \\ \widetilde{\mathbf{k}} \end{pmatrix} = \underbrace{\begin{bmatrix} \mathcal{G}_x^{[0]}, & 0 \\ \mathcal{G}_x^{[0]}, & -\mathcal{G}_y^{[0]} \\ \mathcal{K}_x, & \mathcal{K}_y \end{bmatrix}}_{\mathcal{C}} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}, \tag{6.20}$$

where the first sub-matrix $\widetilde{\mathbf{u}} = \mathcal{G}_x^{[0]} \widetilde{\mathbf{w}}_x$ is employed to collocate the function u over the stencil; the second sub-matrix $\mathcal{G}_x^{[0]} \widetilde{\mathbf{w}}_x - \mathcal{G}_y^{[0]} \widetilde{\mathbf{w}}_y = \widetilde{\mathbf{0}}$ is employed to enforce nodal values of u obtained from the integration with respect to x and y to be identical; the third sub-matrix $\mathcal{K}_x \widetilde{\mathbf{w}}_x + \mathcal{K}_y \widetilde{\mathbf{w}}_y = \widetilde{\mathbf{k}}$ is employed to represent values of the PDE (6.1) at selected points, which are $(\mathbf{x}_2, \mathbf{x}_4, \mathbf{x}_6, \mathbf{x}_8)$ in this study.

The conversion of the network-weight space into the physical space is achieved by inverting Eq. (6.20) as

$$\begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C}^{-1} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \\ \widetilde{\mathbf{k}} \end{pmatrix}, \tag{6.21}$$

or

$$\widetilde{\mathbf{w}}_x = \mathcal{C}_{\mathrm{x}} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \\ \widetilde{\mathbf{k}} \end{pmatrix}, \tag{6.22}$$

$$\widetilde{\mathbf{w}}_y = \mathcal{C}_{\mathrm{y}} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \\ \widetilde{\mathbf{k}} \end{pmatrix}, \tag{6.23}$$

where $[\mathcal{C}_x, \mathcal{C}_y]^T = \mathcal{C}^{-1}$.

By substituting Eqs. (6.22), (6.23) into Eqs. (6.8) - (6.10) and (6.14) - (6.16), the derivatives of function u with respect to x and y directions over a local stencil are now expressed in physical space.

Boundary conditions used in CLIRBF schemes

With Dirichlet BC, the governing equation yields at each grid point an algebraic equation whose variables are the function values of points in the associated stencil. Applying this procedure for all interior points and their associated stencils yields a system of algebraic equations. Since the variables of the equation system at boundary points are known as Dirichlet BC, they are subtracted from the system and moved to the right hand side. A new system of equations can now be solved by an iterative method.

In IRBF approach, there are two ways to impose the Neumann BC on the final system of algebraic equations. The first one is to add expressions of Neumann BC to the final equation system. This approach is effective with coarse grids. However, with dense grids the condition number of system matrices is usually high, which may cause numerical instability. The other way is to add expressions of Neumann BCs into conversion matrices of associated stencils. Governing equations at the boundary points are then also derived and put into the final algebraic system of equations.

Take a 9-point stencil as shown in Fig. 6.1 with Neumann BC $\dfrac{\partial u}{\partial x} = e(\mathbf{x})$ applied at the boundary nodal points $\{\mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9\}$. The conversion matrix is constructed as

$$\begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \\ \widetilde{\mathbf{k}} \\ \widetilde{\mathbf{e}} \end{pmatrix} = \underbrace{\begin{bmatrix} \mathcal{G}_x^{[0]}, & \mathcal{O} \\ \mathcal{G}_x^{[0]}, & -\mathcal{G}_y^{[0]} \\ \mathcal{K}_x, & \mathcal{K}_y \\ \mathcal{G}_x^{[1]}, & \mathcal{O} \end{bmatrix}}_{\mathcal{C}} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}, \tag{6.24}$$

where $e(\mathbf{x})$ is a given function, and $\widetilde{\mathbf{e}} = [e(\mathbf{x}_7), e(\mathbf{x}_8), e(\mathbf{x}_9)]^T$.

Non-rectangular domains

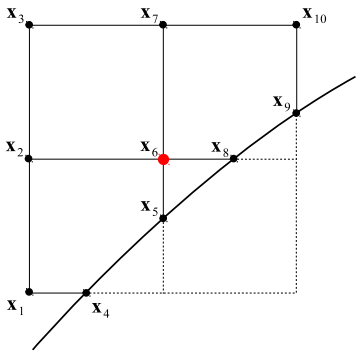


Figure 6.2 Non-rectangular stencil

In non-rectangular domains, some points are positioned near the non-rectangular boundary and the stencils associated with these points can be arbitrary as shown in Fig. 6.2. The IRBF approximation for these stencils is carried out in a dual-stencil associated with x-gridlines and y-gridlines, namely the x and y-stencils.

Take the non-rectangular stencil in Fig. 6.2 as an example, the x-stencil is the union of grid points and boundary points generated by the intersection of x-gridlines and the boundary

$$X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_{10}\}.$$

The y-stencil is the union of grid points and boundary points generated by the intersection of y-gridlines and the boundary

$$Y = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_9, \mathbf{x}_{10}\}.$$

The x-stencil is used to approximate values of unknowns and their derivatives with respect to x, i.e., $\dfrac{\partial^2 u}{\partial x^2}, \dfrac{\partial u}{\partial x}$ and u_x, while the y-stencil is used to approximate those with respect to y, i.e., $\dfrac{\partial^2 u}{\partial y^2}, \dfrac{\partial u}{\partial y}$ and u_y. Note that u_x is obtained in the manner of Eq.(6.4) with $x_j \equiv x$ and u_y with $x_j \equiv y$.

To construct the conversion matrix, some sets of points are explicitly defined as follows.

$$\begin{cases} S_1 = X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_{10}\} \\ S_2 = Y \backslash X = \{\mathbf{x}_5, \mathbf{x}_9\} \\ S_3 = X \cap Y = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_{10}\} \\ S_4 = \{\mathbf{x}_2, \mathbf{x}_7\} \\ S_5 = \{\mathbf{x}_2\} \end{cases}$$

where S_1 is the set of all points in x-stencil; S_2 the set of boundary points generated by y-gridlines crossing the non-rectangular boundary; S_3 the set of common points between x-stencil and y-stencil and S_4 the set of compact points, which are grid points lying on the cross whose centre is the reference point $\mathbf{x}_5$.

Suppose that a Neumann BC $\dfrac{\partial u}{\partial x} = e(\mathbf{x})$ is imposed at $\mathbf{x}_2$ then $S_5 = \{\mathbf{x}_2\}$ is

defined as the set of points with Neumann BC in x dimension. The conversion matrix associated with this stencil is determined as follows.

$$\begin{pmatrix} \widetilde{\mathbf{u}}_{S_1} \\ \widetilde{\mathbf{u}}_{S_2} \\ \widetilde{\mathbf{0}}_{S_3} \\ \widetilde{\mathbf{k}}_{S_4} \\ \widetilde{\mathbf{e}}_{S_5} \end{pmatrix} = \underbrace{\begin{bmatrix} \mathcal{G}_x^{[0]}, & 0 \\ \mathcal{O}, & \mathcal{G}_y^{[0]} \\ \mathcal{G}_x^{[0]}, & -\mathcal{G}_y^{[0]} \\ \mathcal{K}_x, & \mathcal{K}_y \\ \mathcal{G}_x^{[1]}, & 0 \end{bmatrix}}_{\mathcal{C}} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}, \tag{6.25}$$

where the sub-matrix $\mathcal{G}_x^{[0]}\widetilde{\mathbf{w}}_x = \widetilde{\mathbf{u}}_{S_1}$ captures the collocation of u_x at S_1; $\mathcal{G}_y^{[0]}\widetilde{\mathbf{w}}_y = \widetilde{\mathbf{u}}_{S_2}$ the collocation of u_y at S_2; $\mathcal{G}_x^{[0]}\widetilde{\mathbf{w}}_x - \mathcal{G}_y^{[0]}\widetilde{\mathbf{w}}_y = \widetilde{\mathbf{0}}_{S_3}$ the enforcement $u_x = u_y$ at S_3; $\mathcal{K}_x\widetilde{\mathbf{w}}_x + \mathcal{K}_y\widetilde{\mathbf{w}}_y = \widetilde{\mathbf{k}}_{S_4}$ the compact information, which represent the governing equation at S_4; and the last sub-matrix $\mathcal{G}_x^{[1]}\widetilde{\mathbf{w}}_x = \widetilde{\mathbf{e}}_{S_5}$ the Neumann BC at S_5.

6.3 Parallel non-overlapping DD method

The Steklov-Poincare non-overlapping DD method is used in this work and an example with two sub-domains and one AB is shown in Fig. 6.3, in which $\partial\Omega = \Gamma_1 \cup \Gamma_2$.

6.3.1 Non-overlapping Dirichlet-Neumann DD method

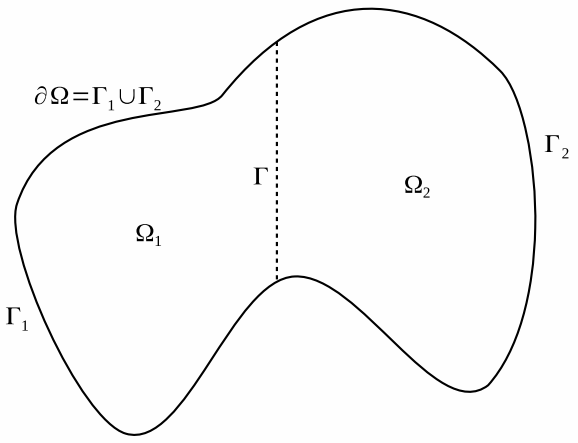


Figure 6.3 Non-overlapping DD method in 2D

The 2D problem (6.1) is rewritten with the introduction of AB Γ

$$\begin{cases} \mathcal{L}u &= f, \quad \mathbf{x} \in \Omega, \\ \mathcal{B}u &= g, \quad \mathbf{x} \in \Gamma_1 \cup \Gamma_2. \end{cases} \tag{6.26}$$

The problem (6.26) is reformulated in an equivalent form with two sub-domains

Ω_1 with $\partial\Omega_1 = \Gamma_1 \cup \Gamma$ and Ω_2 with $\partial\Omega_2 = \Gamma_2 \cup \Gamma$ as follows.

$$\begin{cases} \mathcal{L}u_1 = f, & \mathbf{x} \in \Omega_1 \\ \mathcal{B}u_1 = g, & \mathbf{x} \in \Gamma_1 \\ u_1 = u_2, & \mathbf{x} \in \Gamma \\ \mathcal{L}u_2 = f, & \mathbf{x} \in \Omega_2 \\ \mathcal{B}u_2 = g, & \mathbf{x} \in \Gamma_2 \\ \dfrac{\partial u_2}{\partial n} = \dfrac{\partial u_1}{\partial n}, & \mathbf{x} \in \Gamma \end{cases} \tag{6.27}$$

where subscripts 1 and 2 denote sub-domains. The two constraints $u_1 = u_2$ and $\dfrac{\partial u_2}{\partial n} = \dfrac{\partial u_1}{\partial n}$ are transmission conditions for u across Γ. It is proved that the solution of the equivalent problem (6.27) converges to the solution of the original problem (6.26) (Quarteroni and Valli, 1999).

The non-overlapping Dirichlet-Neumann DD method can be written for step k as

$$\begin{cases} \mathcal{L}u_1^k = f, & \mathbf{x} \in \Omega_1 \\ \mathcal{B}u_1^k = g, & \mathbf{x} \in \Gamma_1 \\ u_1^k = u_2^{k-1}, & \mathbf{x} \in \Gamma \end{cases} \tag{6.28}$$

and

$$\begin{cases} \mathcal{L}u_2^k = f, & \mathbf{x} \in \Omega_2 \\ \mathcal{B}u_2^k = g, & \mathbf{x} \in \Gamma_2 \\ \dfrac{\partial u_2^k}{\partial n} = \dfrac{\partial u_1^k}{\partial n}, & \mathbf{x} \in \Gamma \end{cases} \tag{6.29}$$

The convergence of non-overlapping Dirichlet-Neumann DD method is guaranteed by introducing a relaxation factor θ into the transmission condition in (6.28) so that

$$u_1^k = \theta u_1^{k-1} + (1-\theta)u_2^{k-1}, \quad \mathbf{x} \in \Gamma$$

where $\theta \in [0, 1]$.

6.3.2 Parallel version of non-overlapping Dirichlet-Neumann DD method

It is clear that the DD method presented above is not suitable for parallel computing because the calculation in sub-domain 2 via by Eqs. (6.29) requires values on the AB from sub-domain 1 obtained by Eqs. (6.28) in the same iteration step. Fortunately, this data dependence can be handled so that parallel calculation is applicable. There are two common approaches of parallelisation of non-overlapping DD method.

The first approach is to use the black and white colouring technique. Within this technique, sub-domains that have no-common AB, i.e. not adjacent, are marked with black colour. The remaining sub-domains, which are also not adjacent, are marked with white colour. In the first iteration, only black sub-domains are computed in parallel, and in the next interaction only white sub-domains are

computed in parallel and so on. Although the approach maintains the form of the equivalent problem (6.27), it only allows at most $N/2$ sub-domains to run in parallel, where N is the number of sub-domains.

For the second approach, a small modification to Eqs. (6.29) is made as follows.

$$\begin{cases} \mathcal{L}u_2^k &= f, & \mathbf{x} \in \Omega_2 \\ \mathcal{B}u_2^k &= g, & \mathbf{x} \in \Gamma_2 \\ \dfrac{\partial u_2^k}{\partial n} &= \dfrac{\partial u_1^{k-1}}{\partial n}, & \mathbf{x} \in \Gamma \end{cases} \tag{6.30}$$

where the values of first order derivative on the AB of sub-domain 2 are obtained from sub-domain 1 in previous time step. This modification allows all sub-domains to run concurrently. This approach is employed in this work.

6.3.3 Parallel algorithm based on non-overlapping DD method coupled with IRBF

The original domain of problem is divided into non-overlapping sub-domains. In each sub-domain, a sub-problem with either Dirichlet or Dirichlet and Neumann BC will be solved using the compact local IRBF approximation scheme as presented in Section 6.2. The solution is considered converged once the convergence measures (CM) in each sub-domain as well as on all ABs reach some predefined tolerances.

Communication and Synchronisation

The algorithm is implemented in MATLAB environment. The parallel communication is achieved by using standard send and receive functions (MATLAB, 2012). Since these operations are synchronous, the synchronisation and delivery of messages are guaranteed.

Termination

As presented before, the algorithm stops when CMs reach given tolerances. CM is the norm-2 of a field variable and is defined as follows.

$$CM = \frac{\sqrt{\sum_{i=1}^{n}(u_i^{k+1} - u_i^k)^2}}{\sqrt{\sum_{i=1}^{n}(u_i^k)^2}}, \tag{6.31}$$

where k is the iteration step and n the number collocation points.

In this method, three CMs are used to verify the convergence of the numerical solution. $CM^{[u]}$ is the convergence measure of the field variable obtained by two

successive time steps within a sub-domain. $ABCM^{[u]}$ is the convergence measure of the field variable on the AB between two adjacent sub-domains. $ABCM^{[\frac{\partial u}{\partial n}]}$ is the convergence measure of the first derivative of field variable on the AB between two adjacent sub-domains. The solution is considered converged if all three following conditions are met

$$\begin{cases} CM^{[u]} & < & CM_{tol}, & \text{in sub-domain} \\ ABCM^{[u]} & < & ABCM_{tol}^{[u]}, & \text{on AB} \\ ABCM^{[\frac{\partial u}{\partial n}]} & < & ABCM_{tol}^{[\frac{\partial u}{\partial n}]}, & \text{on AB} \end{cases} \tag{6.32}$$

In parallel computation, each sub-domain is a part of a distributed system. A sub-domain can terminate if there is no other adjacent sub-domain communicating with it. In this work, the Bitmap distributed termination detection (DTD) algorithm presented in Chapter 3 is used to achieve an efficient termination. The Bitmap algorithm allows any sub-domain to detect the termination. Furthermore, the detection delay and the data overhead of Bitmap DTD are low and hence they affect insignificantly on the overall performance of the parallel computation.

The algorithm

The parallel algorithm is shown in Fig. 6.4 and consists of two parts, namely a sequential part and a parallel part.

The sequential part includes two blocks, the DD at the beginning and the collection of results at the end. The DD block is responsible for creating sub-domains and their relative coordinates in the whole domain. The result collection block collects data from all sub-domains and then merges them into one global solution.

In the parallel part, each sub-domain firstly needs to identify its neighbours. Then the Navier-Stokes equations are solved using 2D Compact Local IRBF scheme. The obtained solution is used to approximate the first order normal derivative on ABs. Both the field variable and its normal derivative on ABs are sent to NSs. These data are used to determine CMs and to check the convergence condition (6.32). The last step, termination detection, is to find whether the calculation in a current sub-domain is finished and to terminate accordingly.

6.4 Numerical examples

The efficiency of the present method is demonstrated by solving two benchmark problems: the lid-driven cavity fluid flow and the natural convection in concentric annuli.

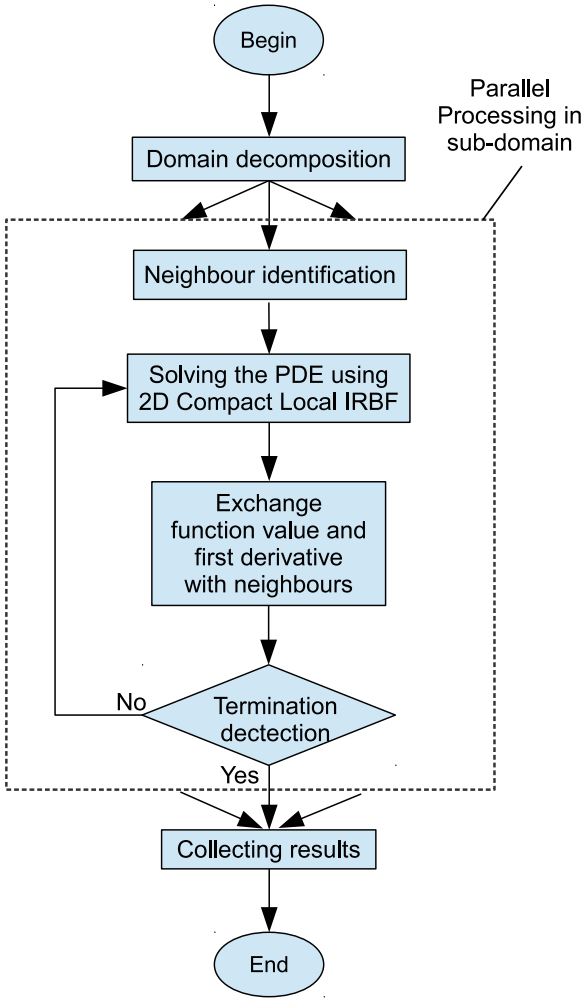


Figure 6.4 Flowchart of parallel algorithm based on non-overlapping Dirichlet-Neumann DD method

6.4.1 Lid-driven cavity fluid flow problem

In dimensionless form, the LDC fluid flow problem is described in Fig. 6.5 where the square cavity has three fixed walls and a sliding top lid. The cavity is filled with a viscous fluid which is driven by the lid with a velocity of 1 from left to right.

The governing equation of the problem is given in stream-function and vorticity formulation as follows.

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = -\omega, \tag{6.33}$$

$$\frac{1}{Re}\left(\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2}\right) = \frac{\partial\omega}{\partial t} + \left(\frac{\partial\psi}{\partial y}\frac{\partial\omega}{\partial x} - \frac{\partial\psi}{\partial x}\frac{\partial\omega}{\partial y}\right), \tag{6.34}$$

where ψ is the stream-function variable, ω the vorticity variable, Re the Reynolds number defined as

$$Re = \frac{UL}{\nu},$$

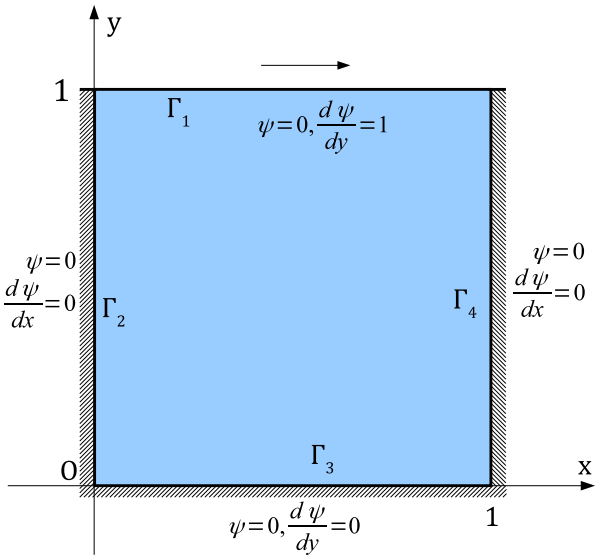


Figure 6.5 The LDC fluid flow problem. Geometry and BCs. For parallel computing, the domain is simply divided into a number of equal rectangular sub-domains

where U is the speed of the lid; L the side of the cavity and ν the kinematic viscosity of the fluid.

The relationship of stream-function and vorticity with velocity components u and v is given as follows. $u = \dfrac{\partial \psi}{\partial y}$; $v = -\dfrac{\partial \psi}{\partial x}$ and $\omega = \dfrac{\partial v}{\partial x} - \dfrac{\partial u}{\partial y}$.

The BCs of the LDC problem in Fig. 6.5 are reproduced as follows.

$$\psi = 0, \quad \frac{\partial \psi}{\partial y} = 1 \quad \forall (x, y) \in \Gamma_1; \tag{6.35}$$

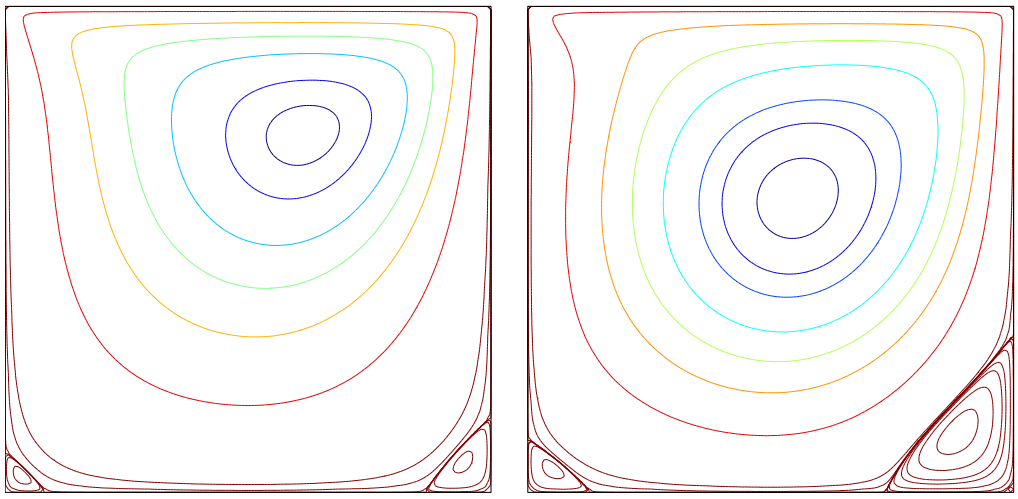
$$\psi = 0, \quad \frac{\partial \psi}{\partial x} = 0 \quad \forall (x, y) \in \Gamma_2 \cup \Gamma_3 \cup \Gamma_4. \tag{6.36}$$

For parallel computing, the original domain (Fig. 6.5) is simply divided to $\{1, 2, 4, 9, \cdots, 81\}$ equal rectangular sub-domains. Since the Dirichlet-Neumann non-overlapping DD method is used, there is an alternation from the Dirichlet BCs on one side of ABs to the Neumann BCs on the other side. For load balancing purpose, the Dirichlet BCs are set on the top and right ABs and the Neumann BCs on the bottom and left ABs in each sub-domain. In each sub-domain, the problem is solved through the following procedure.

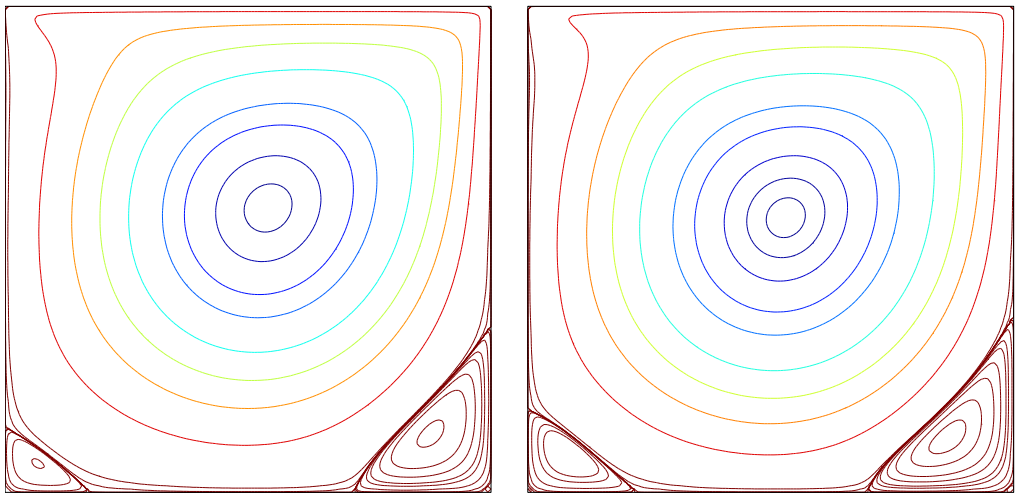
1. Set the rest state of unknowns ψ and ω. Guess the initial BCs on the ABs.
2. Solve Eq. (6.33) for ψ.
3. Approximate ω on the ABs and solve Eq. (6.34) for ω.
4. Exchange the Dirichlet and/or Neumann BCs over the ABs.
5. Check the termination condition (6.32). If satisfied then terminate. Otherwise, update BCs on the ABs and go to step 2.

Since the BCs of ω are unknown, they need to be determined in order to solve Eq. (6.34). In this implementation, the BCs of ω are approximated from Eq. (6.33) using the global 1D IRBF method (Mai-Duy and Tran-Cong (2001)).

The LDC fluid flow problem is investigated with several Reynolds numbers {100, 400, 600, 1000}. The contours of stream-function ψ and vorticity ω are presented in Figs. 6.6 and 6.7 while the velocity profiles are shown in Fig. 6.8.



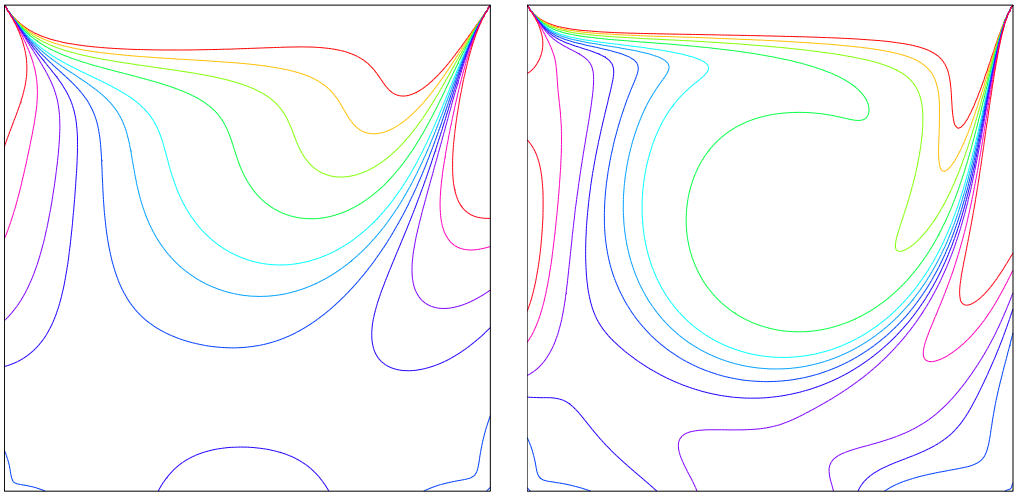
(a) $Re = 100$ (b) $Re = 400$



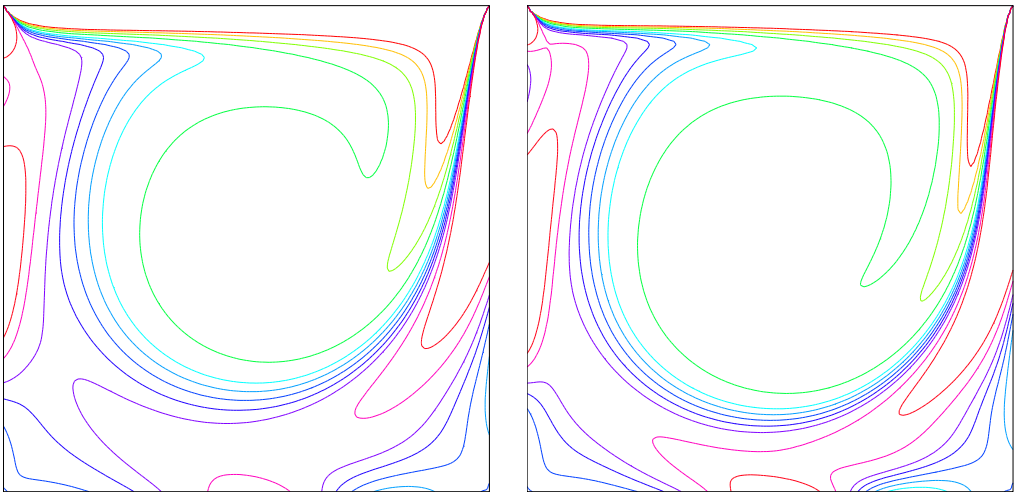
(c) $Re = 600$ (d) $Re = 1000$

Figure 6.6 The LDC fluid flow problem. Stream-function (ψ) contours of the flow for several Reynolds numbers ($Re = 100$, 400, 600 and 1000) by the present parallel method using 16 CPUs with the specifications: grid 151×151, $\Delta t = 10^{-3}$, $CM_{\text{tol}}^{[u]} = 10^{-6}$, $ABCM_{tol}^{[u]} = 10^{-6}$, $ABCM_{tol}^{[\frac{\partial u}{\partial n}]} = 10^{-6}$, $\theta = 0.45$ and $\beta = 2$.

The contours of stream-function and vorticity in Figs. 6.6 and 6.7 show a great agreement with benchmark results by Ghia et al. (1982) and by Botella and Peyret (1998). While the velocity profiles u along the vertical centreline and v along the horizontal centreline in Fig. 6.8 also go through corresponding values given by Ghia et al. (1982).

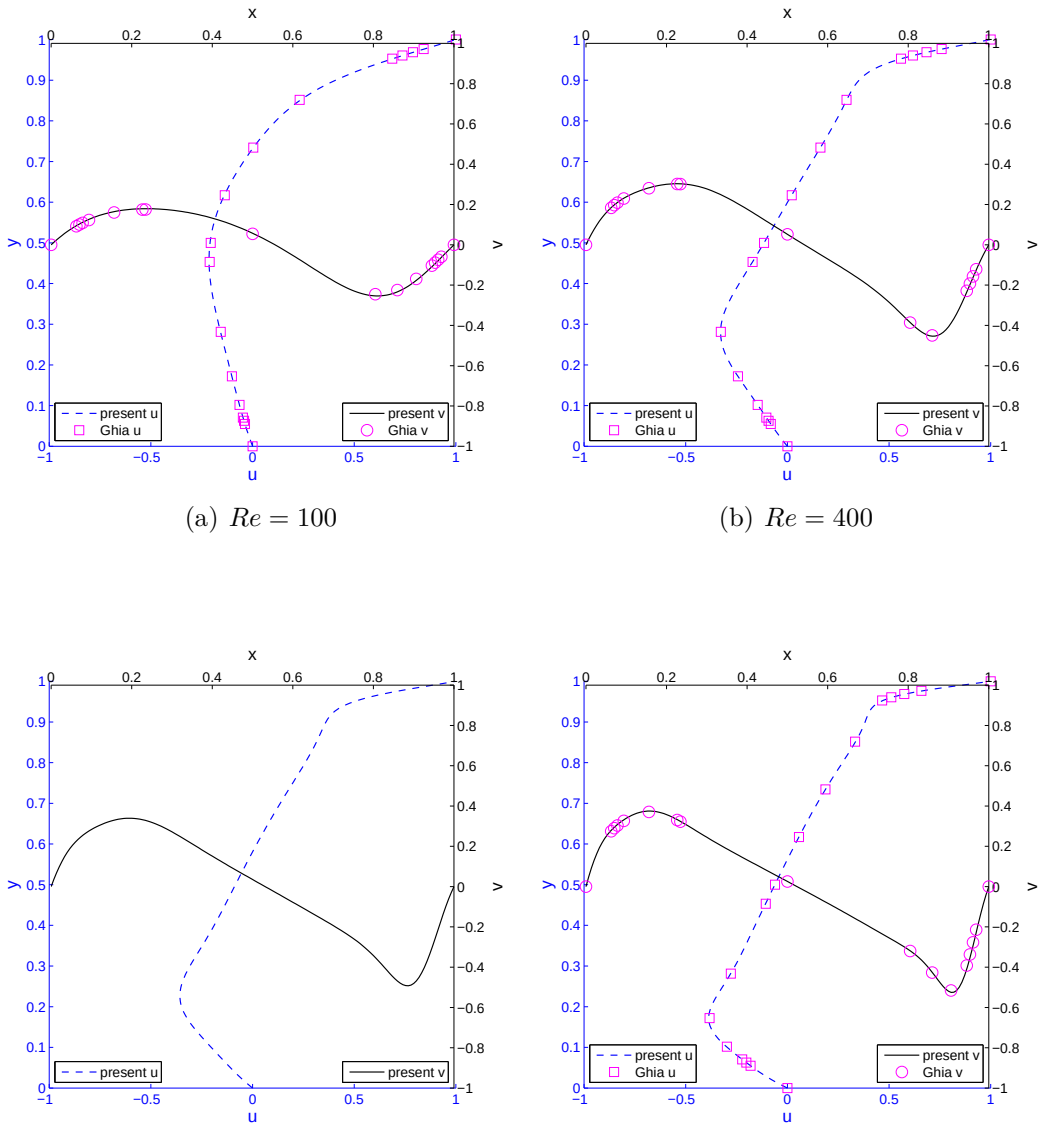


(a) $Re = 100$ (b) $Re = 400$



(c) $Re = 600$ (d) $Re = 1000$

Figure 6.7 The LDC fluid flow problem. Vorticity (ω) contours of the flow for several Reynolds numbers ($Re = 100$, 400, 600 and 1000) by the present parallel method using 16 CPUs. Other parameters are given in Fig. 6.6.



(a) $Re = 100$ (b) $Re = 400$

(c) $Re = 600$ (d) $Re = 1000$

Figure 6.8 The LDC fluid flow problem. Profiles of the u velocity along the vertical centreline (dashed lines) and the v velocity along the horizontal centreline (solid lines) for several Reynolds numbers ($Re = 100, 400, 600$ and 1000) by the present parallel method in comparison with the corresponding Ghia's results (□ for u velocity and ○ for v velocity). Parameters of the problem are given in Fig. 6.6.

Table 6.1 The LDC fluid flow problem. Parallel performance with grid 151×151, $\Delta t = 10^{-3}$, $CM_{\text{tol}}^{[u]} = 10^{-6}$, $ABCM_{tol}^{[u]} = 10^{-6}$, $ABCM_{tol}^{[\frac{\partial u}{\partial n}]} = 10^{-6}$, $\theta = 0.45$ and $\beta = 2$. CPUs: number of CPUs (sub-domains); N_i: number of iterations; T_p - parallel computation time (minutes) on parallel CPUs, $i = 1$ for single CPU; S - speed-up; E - efficiency.

	$Re = 100$				$Re = 400$			
CPUs	N_i	T_p	S	E	i	T_p	S	E
1	8814	132.25	1.00	100.00	22107	324.98	1.00	100.00
2	13486	91.41	1.45	72.34	28594	193.52	1.68	83.97
4	13793	42.37	3.12	78.04	30770	96.15	3.38	84.50
9	14330	18.07	7.32	81.30	31931	40.45	8.03	89.27
16	16332	8.80	15.03	93.96	33649	18.61	17.46	109.14
25	15096	5.12	25.84	103.34	37848	12.29	26.45	105.81
36	17665	4.08	32.44	90.10	34088	7.81	41.62	115.60
49	16212	2.90	45.59	93.04	37932	6.33	51.35	104.79
64	16257	2.31	57.22	89.41	33964	4.48	72.60	113.44
81	15989	1.97	67.05	82.78	38383	4.31	75.40	93.08

The efficiency of parallel method is often evaluated by the speed-up S and the efficiency E.

$$\begin{aligned} S &= \frac{T_s}{T_p}, \\ E &= \frac{S}{p}, \end{aligned} \tag{6.37}$$

where T_s is the computation time on a single CPU, T_p the computation time on parallel CPUs, p the number of parallel CPUs.

The efficiency of the present parallel method is demonstrated using a grid of 151×151 with a range of CPU numbers from 1 to 81 as described in Fig. 6.9. As can be seen in Fig. 6.9 (bottom figure), the throughput of the present parallel method is very high. Furthermore, the speed-up grows steadily with the increase of number of CPUs (top - left figure), this shows a good scalability of the present parallel algorithm. It is noted that the speed-up deteriorates as the number of CPUs exceeds a certain threshold. For example, the threshold is 64 CPUs for $Re = 400$ and $Re = 600$; and 49 CPUs for $Re = 1000$. For a parallel program, the threshold sets an upper-bound, over which the efficiency of a parallel computation starts to decrease. In this example, it appears that the CPU threshold decreases with increasing Re number.

The results show that the present method achieves a very high efficiency. Furthermore, a super-linear speed-up is achieved with 25 CPUs for $Re = 100$, with $\{16, 25, 36, 49, 64\}$ CPUs for $Re = 400$, with $\{25, 36, 49, 64\}$ CPUs for $Re = 600$, and with $\{16, 25, 36, 49, 64\}$ CPUs for $Re = 1000$ (Fig. 6.9, top - right). A reason of the high efficiency is the insignificant increase of number of iterations in the present parallel computation in comparison with the sequential one. Detailed efficiency of the present parallel method is presented in Tables 6.1 and 6.2 for $Re = \{100, 400\}$ and $Re = \{600, 1000\}$, respectively.

To investigate the ratio of the communication time to the total execution time on each CPU, the time profile of the case of 25 CPUs is recorded and provided in

Table 6.2 The LDC fluid flow problem. Parallel performance with grid 151×151, $\Delta t = 10^{-3}$, $CM_{\text{tol}}^{[u]} = 10^{-6}$, $ABCM_{tol}^{[u]} = 10^{-6}$, $ABCM_{tol}^{[\frac{\partial u}{\partial n}]} = 10^{-6}$, $\theta = 0.45$ and $\beta = 2$. CPUs - number of CPUs (sub-domains); N_i - number of iterations; T_p - parallel computation time (minutes) on parallel CPUs except for the case with 1 CPU which is non-parallel; S - speed-up; E - efficiency.

	$Re = 600$				$Re = 1000$			
CPUs	N_i	T_p	S	E	i	T_p	S	E
1	25824	356.29	1.00	100.00	30442	453.65	1.00	100.00
2	33026	242.43	1.47	73.48	38044	261.37	1.74	86.78
4	35653	109.17	3.26	81.59	43619	130.61	3.47	86.83
9	35595	47.00	7.58	84.22	40853	50.51	8.98	99.80
16	38769	22.97	15.51	96.96	48079	27.34	16.60	103.72
25	36903	13.65	26.11	104.43	45085	14.49	31.30	125.20
36	38148	8.79	40.54	112.61	53924	11.35	39.98	111.06
49	40018	6.81	52.34	106.81	45231	7.76	58.44	119.27
64	40340	5.42	65.78	102.78	54754	6.93	65.49	102.33
81	46869	5.29	67.41	83.22	57140	6.24	72.73	89.79

Table 6.3. The obtained results show that the percentage of the communication time in the total time ranges from 12.38% to 24.29%.

In order to have an objective comparison on the efficiency of present parallel computation over the sequential one, a normalised computation time is defined as the time required for each CPU to complete a computation task in an ideal parallel program and given by

$$T_n = \frac{T_s}{p}. \tag{6.38}$$

The efficiency E described in Eq. (6.37) can now be determined by following formula

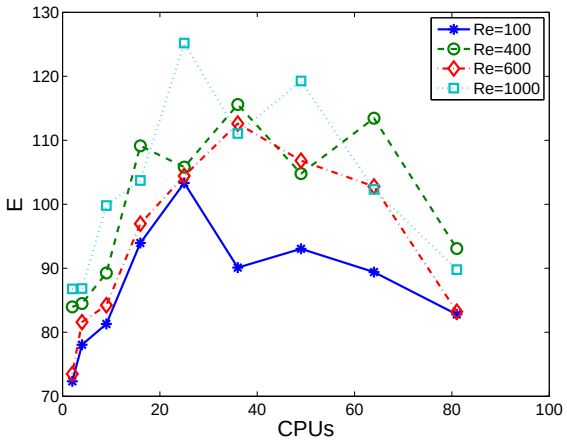
$$E = \frac{T_n}{T_p}. \tag{6.39}$$

The efficiency by the present parallel method is shown in Table 6.3.

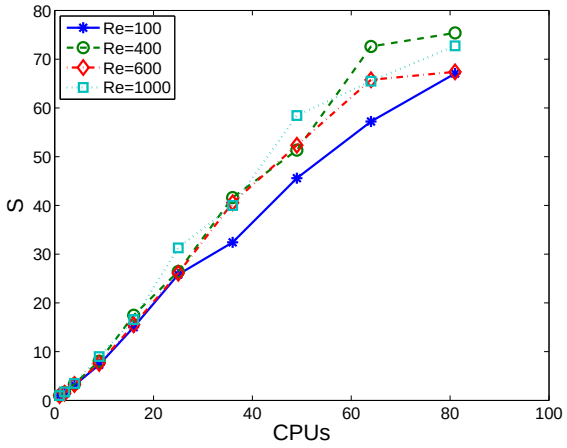
A bar graph of the efficiency of the parallel and the sequential program is also provided in Fig. 6.10. The graph presents a quantitative comparison between the communication time and computation time in a parallel computation as well as between the present parallel computation using 25 CPUs and the associated sequential one, in which super-linear efficiency is achieved.

Table 6.3 The LDC fluid flow problem. Comparison between computation time and communication time in parallel program and total time between parallel program and sequential program. Parallel program runs on 25 CPUs. Re is Reynolds number, T_{cmm}: communication time (minute), T_{cmp}: computation time (minute), T_p: parallel computation time (minutes), % T_{cmm}: percentage of communication time in total time, T_s: sequential computation time (minutes), T_n: normalised time, E: efficiency

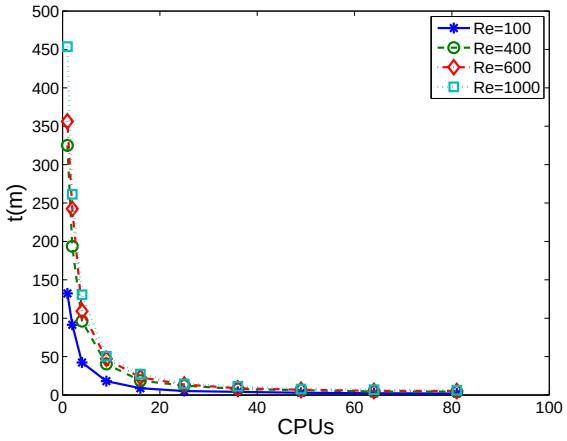
	Parallel				Sequential		
Re	T_{cmm}	T_{cmp}	T_p	$\%T_{cmm}$	T_s	T_n	E
100	0.63	4.49	5.12	12.38	132.25	5.29	103.34
400	2.78	9.50	12.29	22.66	324.98	13.00	105.81
600	3.31	10.33	13.65	24.29	356.29	14.25	104.43
1000	2.92	11.58	14.49	20.13	453.65	18.15	125.20



(a) Efficiency



(b) Speed-up



(c) Simulation time

Figure 6.9 The LDC fluid flow problem. Performance of the present parallel algorithm for several Reynolds numbers (100, 400, 600 and 1000) with a grid of 151×151: the efficiency, speed-up and simulation time as a function of the number of CPUs. Other parameters are given in Tables 6.1, 6.2.

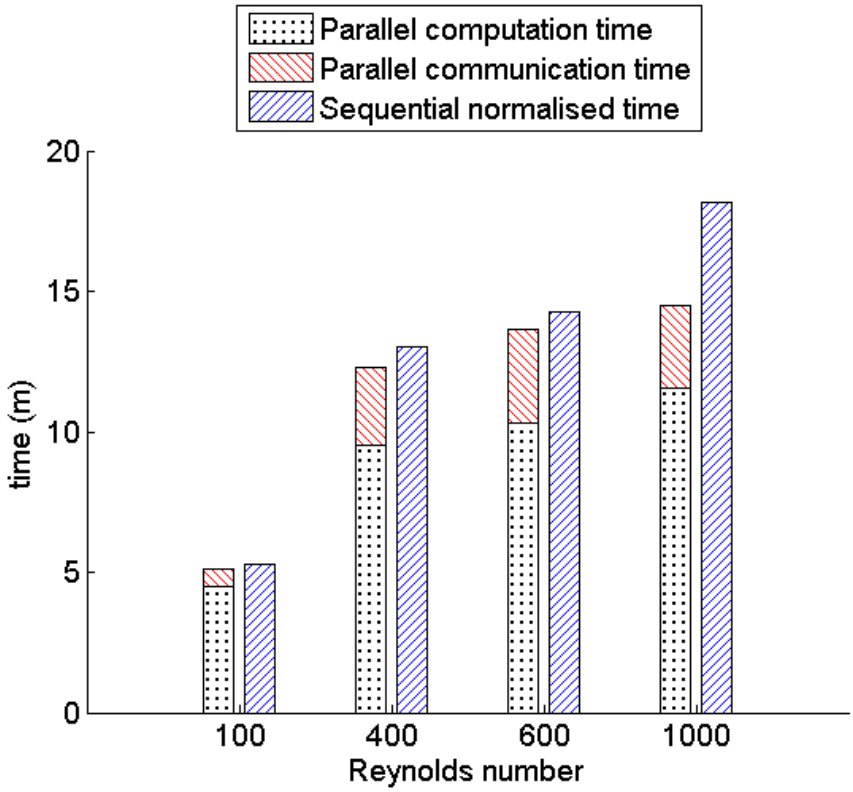


Figure 6.10 The LDC fluid flow problem. Comparison of execution times between the sequential computation and parallel computation using 25 CPUs

6.4.2 Natural convection in concentric annuli

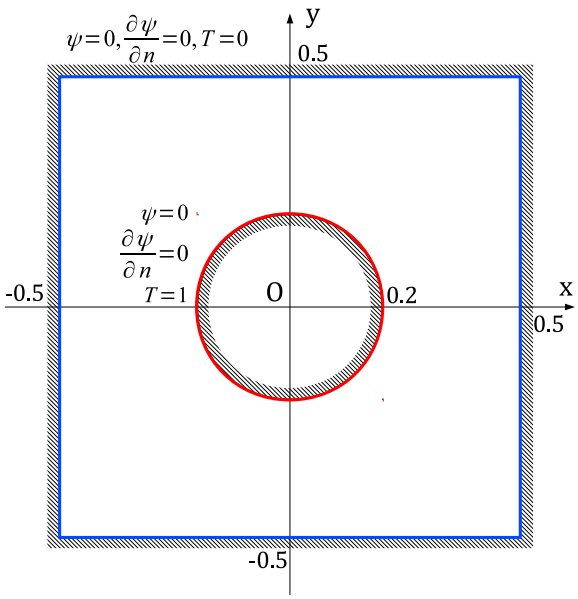


Figure 6.11 The NC problem. Geometry and BCs.

The problem of NC in concentric annuli is of significant interest to the engineering community because of its high applicability to engineering problems such as heat distribution, reactor design, engine design etc. A viscous fluid is enclosed in an annulus defined by an outer square cylinder and a concentric inner circular cylinder (Fig. 6.11). The outer cool wall is kept at a constant temperature $T = 0$ and the inner hot wall at $T = 1$. Under Boussinesq approximation, the temperature differential between the walls generates natural convection. The problem was recently solved by several researchers using various non-parallel methods (Moukalled and Acharya, 1996; Kim et al., 2008; Le-Cao et al., 2009; Hussain and Hussein, 2010; Ngo-Cong et al., 2012).

Since the inner boundary is non-rectangular, for some numerical methods the physical domain needs to be transformed into a rectangular computational domain. However, with IRBF approximation, the transformation is not necessary and a Cartesian grid with added irregular boundary points is used as described in Fig. 6.12.

The dimensionless governing equations for the NC problem are given by

$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2} = -\omega, \tag{6.40}$$

$$\sqrt{\frac{Pr}{Ra}}\left(\frac{\partial^2\omega}{\partial x^2}+\frac{\partial^2\omega}{\partial y^2}\right) = \frac{\partial\omega}{\partial t}+\left(u\frac{\partial\omega}{\partial x}+v\frac{\partial\omega}{\partial y}-\frac{\partial T}{\partial x}\right), \tag{6.41}$$

$$\frac{1}{\sqrt{RaPr}}\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right) = \frac{\partial T}{\partial t}+\left(u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}\right), \tag{6.42}$$

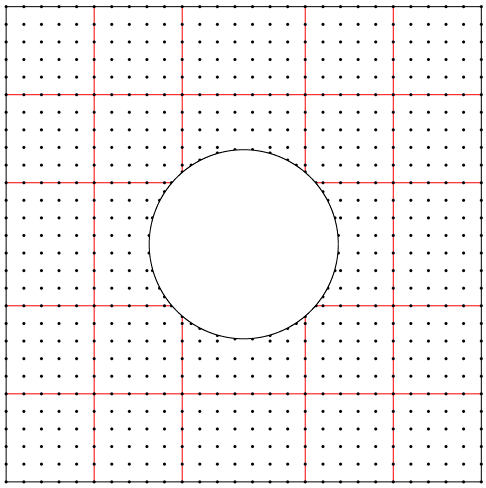


Figure 6.12 The NC problem. A sample grid of 28×28 points with 24 sub-domains.

where Ra is the Rayleigh number, and Pr the Prandtl number defined as

$$\begin{aligned} Pr &= \frac{\nu}{\alpha}, \\ Ra &= \frac{g\beta\Delta T L^3}{\nu\alpha}, \end{aligned}$$

where ν is the kinematic viscosity, α the thermal diffusivity, g the acceleration due to gravity, β the thermal expansion coefficient, L the characteristic length and ΔT and the difference between wall temperature and quiescent temperature.

The problem is investigated with the following parameters

$$\begin{aligned} \frac{L}{2R} &= 2.5; \\ Pr &= 0.71; \\ Ra &= \{10^4, 10^5, 10^6\}; \end{aligned}$$

where L is the length of sides of the outer square profile; R the radius of the inner circular profile.

BCs of the NC problem presented in Fig. 6.11 are reproduced as follows.

$$\begin{aligned} \psi = 0, \quad \frac{\partial\psi}{\partial n} = 0, \quad T = 0 \quad &\text{on outer square,} \\ \psi = 0, \quad \frac{\partial\psi}{\partial n} = 0, \quad T = 1 \quad &\text{on inner circle.} \end{aligned} \tag{6.43}$$

Fig. 6.13 depicts one way to partition the considered domain into 24 non-overlapping sub-domains. Each sub-domain is assigned a number from 1 to 24 called process-index. The process-index is used to position sub-domains in the parallel system and to communicate with others.

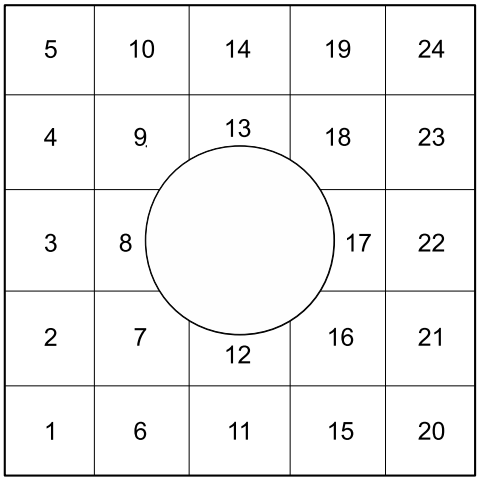


Figure 6.13 The NC problem. Sub-domain formation and enumeration.

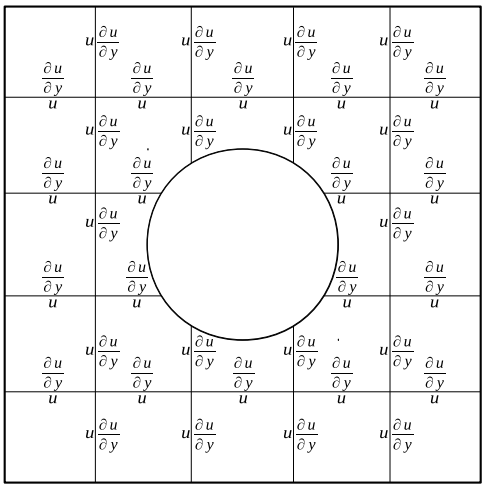


Figure 6.14 The NC problem. BCs on ABs of sub-domains.

For each sub-domain, the top and left ABs are of Dirichlet type and the bottom and right ABs are of Neumann type. A detailed example is shown in Fig. 6.14.

In each sub-domain, the solving procedure for the natural convection problem using the present method can now be described as follows.

1. Set the initial values for ψ, ω, T. Guess the initial values of BCs on ABs.
2. Solve Eq. (6.40) for ψ.
3. Approximate u, v and solve Eq. (6.42) for T.
4. Approximate ω on boundary and solve Eq. (6.41) for ω.
5. Exchange the values of Dirichlet and/or Neumann BCs of ψ, ω and T over ABs.

6. Check the termination condition (6.32). If satisfied then terminate. Otherwise, update BCs on ABs and go to step 2.

As presented in step 4 of the solving procedure, the value of ω on boundary needs to be approximated. As can be seen in Fig. 6.12, since some boundary points on the inner cylinder do not coincide with grid points, equivalent formulas instead of Eq. (6.40) are used to approximate ω on ABs as follows (Le-Cao et al., 2009).

$$\omega_b = -\left[1 + \left(\frac{t_x}{t_y}\right)^2\right] \frac{\partial^2 \psi_b}{\partial x^2}, \tag{6.44}$$

for boundary points on x-grid lines and

$$\omega_b = -\left[1 + \left(\frac{t_y}{t_x}\right)^2\right] \frac{\partial^2 \psi_b}{\partial y^2}, \tag{6.45}$$

for boundary points on y-grid lines, where t_x and t_y are the x- and y-components of the unit vector tangential to the boundary, respectively. Thus, the value of ω on the inner cylinder can be approximated in only one dimension x or y.

Figs. 6.15 - 6.16 show the stream-function (ψ), vorticity (ω) and temperature (T) contours and normal derivative of temperature of the NC problem for several Rayleigh numbers ($Ra = 10^4$, 10^5 and 10^6) using 24 sub-domains. The results by the present parallel method are in very good agreement with those by other works cited above, using the same rheological parameters.

In order to verify the accuracy of the method, the average Nusselt numbers associated with the outer and the inner walls are calculated to compare with those by other works.

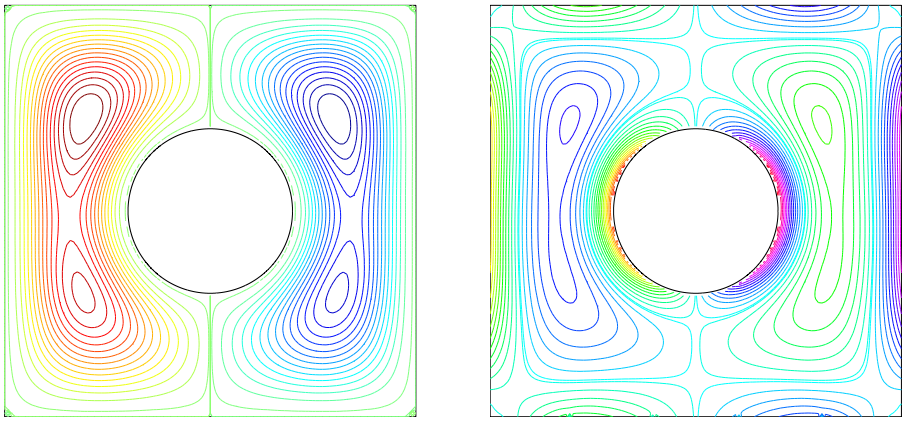
$$\overline{Nu} = \oint \frac{\partial T}{\partial n} ds. \tag{6.46}$$

As presented in Table 6.4, the average Nusselt number on the hot wall by the present method is in great agreement with other results obtained by different non-parallel numerical methods with single domain. It is noted that the average Nusselt number in some of the works cited are calculated for half of the domain and thus in this work, the actual average Nusselt number needs to be divided by 2 for comparison purpose.

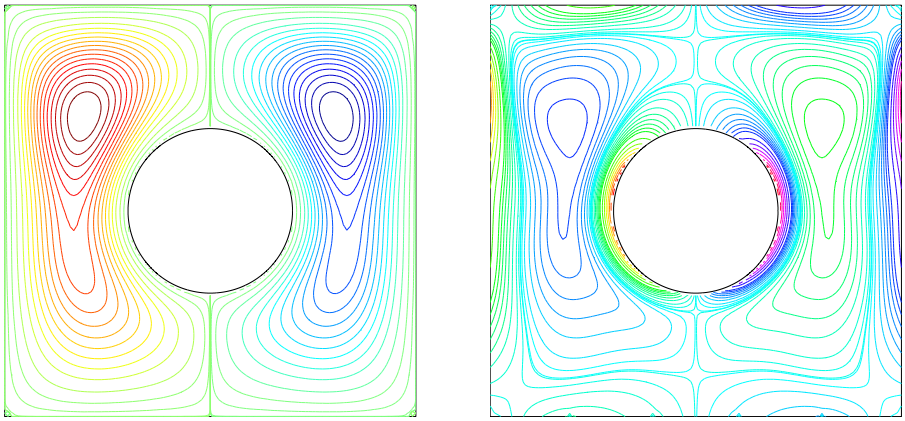
Table 6.4 The NC problem. Comparison of average Nusselt numbers. Present results are obtained by a parallel algorithm with 24 CPUs

Ra	10^4	10^5	10^6
Present	3.23	4.91	8.90
Ngo-Cong et al. (2012) [a]	3.23	4.92	8.90
Le-Cao et al. (2009) [b]	3.21	4.89	8.85
Hussain and Hussein (2010) [c]	3.40	5.13	9.39
Kim et al. (2008) [d]	3.41	5.14	9.39
Shu and Zhu (2002) [e]	3.24	4.86	8.90
Moukalled and Acharya (1996) [f]	3.33	5.08	9.37

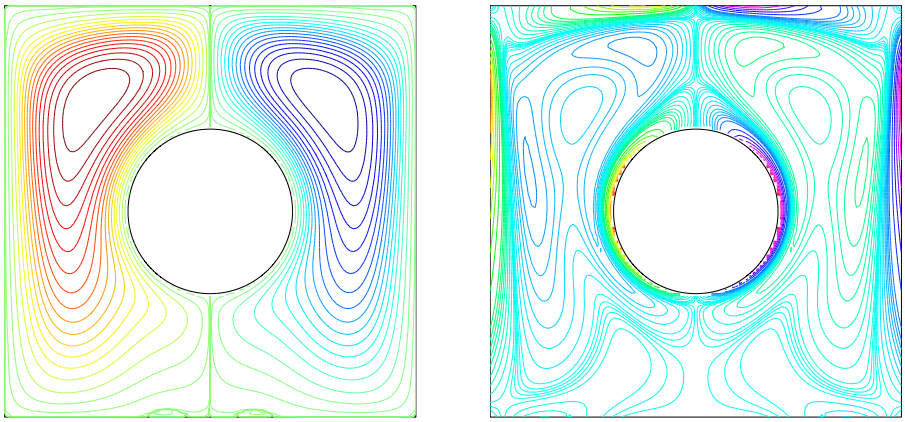
[a] Local moving least square 1D-IRBFN
[b] 1D-IRBFN
[c] Finite volume method
[d] Immersed boundary method
[e] Differential quadrature method
[f] Finite volume method



$Ra = 10^4$

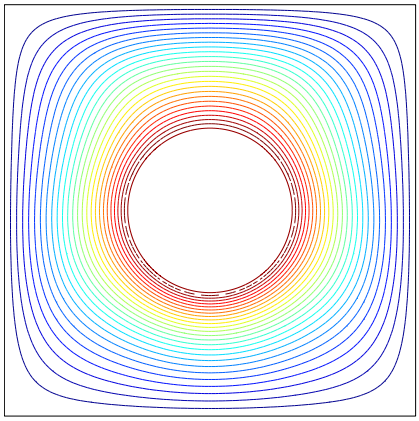


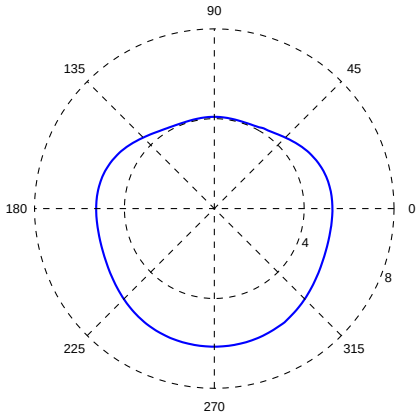
$Ra = 10^5$



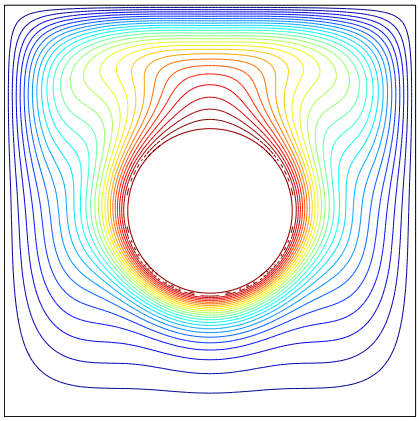
$Ra = 10^6$

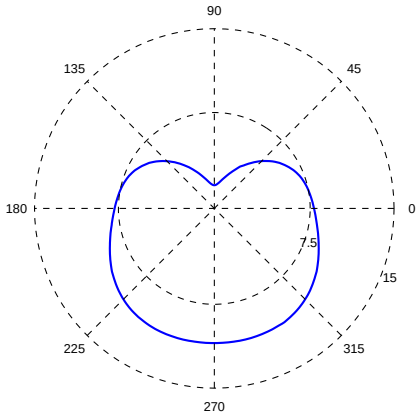
Figure 6.15 The NC problem. Stream-function (ψ) contours (on the left) and vorticity (ω) contours (on the right) of the flow for several Rayleigh numbers using 24 sub-domains, grid 120×120, $\Delta t = 10^{-4}$, $CM_{\text{tol}}^{[u]} = 10^{-9}$, $ABCM_{tol}^{[u]} = 10^{-8}$, $ABCM_{tol}^{[\frac{\partial u}{\partial n}]} = 10^{-8}$, $\theta = 0.25$ and $\beta = 2$.



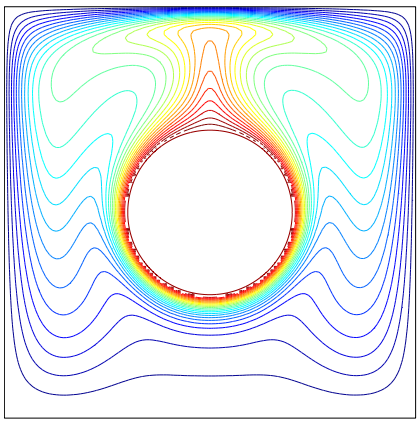


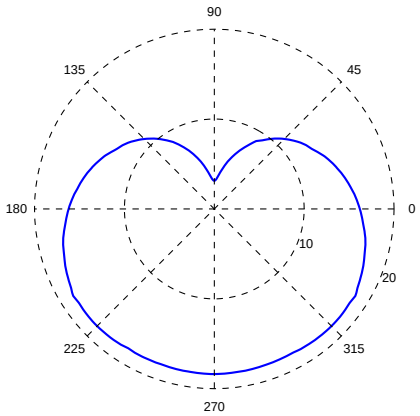
$Ra = 10^4$





$Ra = 10^5$





$Ra = 10^6$

Figure 6.16 The NC problem. Temperature (T) contours (on the left) and normal derivative of temperature $\frac{\partial T}{\partial n}$ along the inner circle (on the right) of the flow for several Rayleigh numbers, using 24 sub-domains, other parameters are given in Fig. 6.15.

Table 6.5 The NC problem. Parallel performance with $Ra = 10^4$, grid 120×120, $\Delta t = 10^{-4}$, $CM_{\text{tol}}^{[u]} = 10^{-9}$, $ABCM_{tol}^{[u]} = 10^{-8}$, $ABCM_{tol}^{[\frac{\partial u}{\partial n}]} = 10^{-8}$, $\theta = 0.25$ and $\beta = 2$. CPUs: number of CPUs (sub-domains); N_i: number of iterations; $\overline{Nu}_o$ average Nusselt number on the outer square (cold wall); $\overline{Nu}_i$ average Nusselt number on the inner circle (hot wall); t(m): elapsed time (minutes); S: speed-up; E: efficiency.

CPUs	N_i	$\overline{Nu}_o$	$\overline{Nu}_i$	t(m)	S	E
1	118288	3.2253	3.2245	1635.58	1.00	100.00
2	120297	3.2253	3.2249	446.17	3.67	183.29
4	120275	3.2253	3.2249	232.23	7.04	176.08
8	125224	3.2253	3.2247	124.47	13.14	164.26
16	124170	3.2252	3.2251	51.05	32.04	200.25
24	132465	3.2253	3.2253	33.48	48.85	203.56
32	131055	3.2253	3.2247	27.62	59.21	185.02
48	130681	3.2253	3.2262	18.37	89.05	185.52
60	135502	3.2253	3.2251	15.35	106.55	177.58
72	136952	3.2254	3.2244	13.35	122.48	170.11

Table 6.6 The NC problem. Parallel performance with $Ra = 10^5$, grid 120×120, other parameter can be seen in Table 6.5.

CPUs	N_i	$\overline{Nu}_o$	$\overline{Nu}_i$	t(m)	S	E
1	386141	4.9053	4.9065	4542.72	1.00	100.00
2	392906	4.9053	4.9072	1509.73	3.01	150.45
4	401923	4.9053	4.9073	764.73	5.94	148.51
8	420629	4.9047	4.9063	420.99	10.79	134.88
16	422929	4.9054	4.9069	190.56	23.84	148.99
24	448339	4.9045	4.9072	115.11	39.46	164.43
32	443774	4.9040	4.9059	90.69	50.09	156.54
48	446269	4.9047	4.9100	62.49	72.69	151.44
60	446010	4.9030	4.9060	52.31	86.84	144.73
72	445250	4.9035	4.9055	45.80	99.19	137.76

Tables 6.5 - 6.7 present numerical results of the NC problem in a concentric annulus using the present parallel computation method with a range of number of CPUs for three Rayleigh numbers $Ra = \{10^4, 10^5, 10^6\}$

It can be seen that the oscillation of the average Nusselt numbers on the outer square and the inner circle with respect to various numbers of CPUs is insignificant. For example, the maximum differences of Nusselt numbers are around 0.08%, 0.11% and 0.13% for $Ra = 10^4$, $Ra = 10^5$ and $Ra = 10^6$, respectively (Tables 6.5 - 6.7). This confirms the stability of the present parallel computation with different numbers of CPUs.

In this numerical example, the super-linear efficiency is observed with a whole range of numbers of CPUs from 2 to 112 for all three considered Rayleigh numbers (see Tables 6.5 - 6.7 column E). One reason of the super-linearity can be explained by the reduction of condition number with the increase of number of CPUs as seen in Tables 6.8 for both the LDC and NC problems. For example, the condition number for the NC problem reduces from from 4563.29 down to 106.50 when increasing the number of CPUs from 1 to 72 sub-domains with a 120×120 grid.

Table 6.7 The NC problem. Parallel performance with $Ra = 10^6$, grid 120×120, other parameter can be seen in Table 6.5.

CPUs	N_i	$\overline{Nu}_o$	$\overline{Nu}_i$	t(m)	S	E
1	759280	8.8726	8.9015	9093.03	1.00	100.00
2	759000	8.8673	8.9034	2882.77	3.15	157.71
4	785182	8.8676	8.9034	1483.05	6.13	153.28
8	846039	8.8652	8.8990	833.02	10.92	136.45
16	895329	8.8686	8.9021	404.71	22.47	140.42
24	902945	8.8658	8.9043	245.91	36.98	154.07
32	902989	8.8655	8.8991	181.62	50.07	156.45
48	897475	8.8717	8.9075	126.41	71.93	149.85
60	900293	8.8668	8.9019	100.34	90.62	151.04
72	904160	8.8624	8.8956	88.91	102.28	142.05

Table 6.8 Condition number of system matrix in LDC and NC problems with respect to number of CPUs (sub-domains)

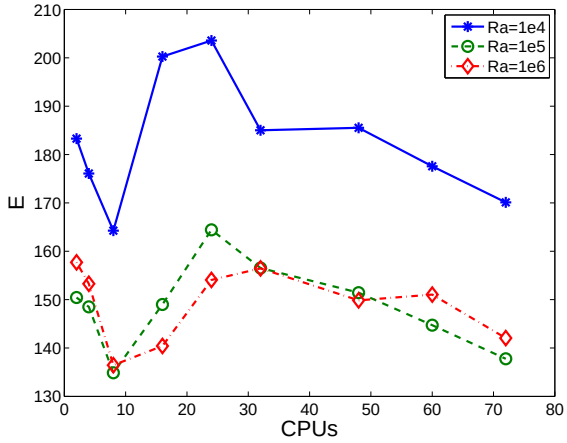
LDC grid 151×151		NC grid 120×120	
CPUs	CN_ψ	CPUs	CN_ψ
1	10724.71	1	4563.29
2	4143.51	2	4316.68
4	2680.41	4	4036.09
9	1191.68	8	883.07
16	688.34	16	346.73
25	429.04	24	251.40
36	297.15	32	230.75
49	230.75	48	122.08
64	171.32	60	93.48
81	137.00	72	106.50

The speed-up for all three Rayleigh numbers presented in Fig. 6.17 (top - right) linearly increases with respect to the number of CPUs. As in the LDC problem, this confirms the high scalability of the present parallel method.

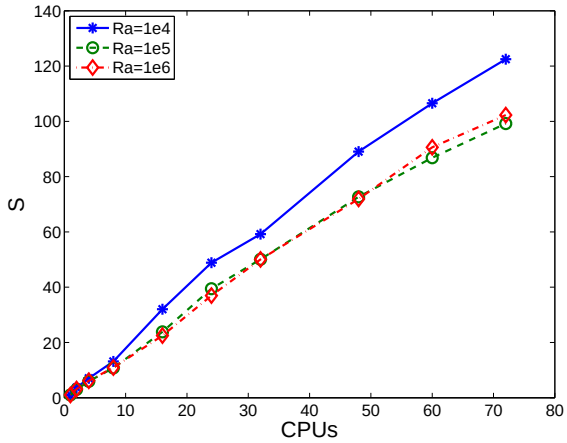
The comparison of the communication time and the computation time in parallel computing as well as the total time in the parallel program with 24 CPUs and sequential one is given in Table 6.9. A bar graph is also provided in Fig. 6.18 for better visualisation of these comparison.

Table 6.9 The NC flow problem. Comparison between the computation time and communication time in parallel program, and the total time between parallel computation and sequential computation. Parallel program using 24 CPUs. Ra: Rayleigh number, T_{cmm}: communication time in minutes, T_{cmp}: computation time in minutes, T_p: computation time (minutes) on parallel CPUs, $\%T_{cmm}$: percentage of communication time in total time, T_s: computation time (minutes) on single CPU, T_n: normalised time (minutes), E: efficiency

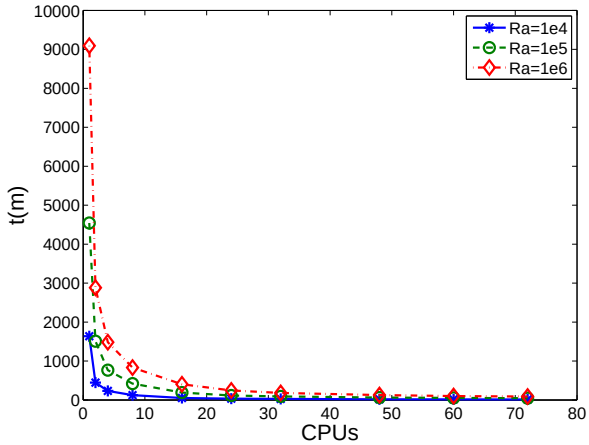
	Parallel				Sequential		
Ra	T_{cmm}	T_{cmp}	T_p	$\%T_{cmm}$	T_s	T_n	E
10^4	10.07	23.41	33.48	30.07	1635.58	68.15	203.56
10^5	42.03	73.09	115.11	36.51	4542.72	189.28	164.43
10^6	64.95	180.96	245.91	26.41	9093.03	378.88	154.07



(a) Efficiency



(b) Speed-up



(c) Simulation time

Figure 6.17 The NC flow problem. Performance of the present parallel algorithm for several Rayleigh numbers: the efficiency, speed-up and simulation time with respect to the number of CPUs. Other parameters are given in Table 6.5.

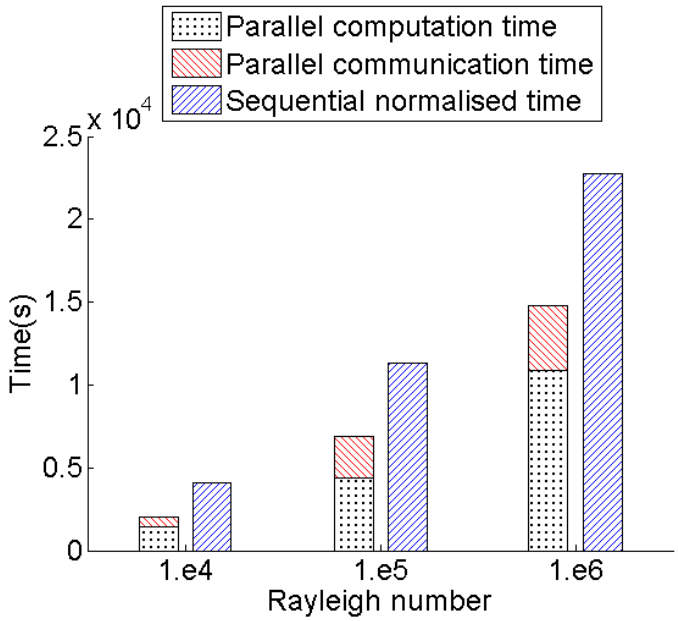


Figure 6.18 The NC flow problem. Comparison of execution times between the sequential computation and parallel computation using 24 CPUs

6.5 Conclusion and remarks

In this chapter, the non-overlapping Dirichlet-Neumann DD method coupled with a 2D CLIRBF scheme to solve the incompressible Navier-Stokes equations has been successfully developed. The achieved results for the LDC fluid flow and the NC in concentric annuli are in good agreement with benchmark results. The time efficiency is very high and super-linear efficiency is achieved with a wide range of numbers of CPUs. This super-linearity can be partly explained by the decrease of the condition number of the system matrix of sub-problems and by the insignificant increase of number of iterations required for the solution to converge. The oscillation of solutions with varying number of CPUs is small. Furthermore, the method also shows good scalability as the speed-up grows consistently as a function of number of CPUs with given grids.

Chapter 7

Parallel IRBF method for the numerical simulation of viscoelastic fluid flows

In this chapter, the parallel algorithm based on the high order IRBF methods will be further demonstrated by simulating the flow of Newtonian and viscoelastic fluids in a 4:1 planar contraction geometry. The 4:1 planar contraction flow of Oldroyd-B fluids at several Weissenberg numbers will be analysed. The performance of parallel computation is evaluated with a range of numbers of CPUs. The overall speed-up and efficiency of the method is high. The results obtained from the simulation are comparable with other results in the literature.

7.1 The 4:1 planar contraction flows

In rheological computation, the simulation of flows of a viscoelastic fluid through an abrupt contraction has always been considered as a benchmark problem for numerical methods. The difficulty of this problem arises from the singularity of stress at the re-entrant corners. At these sharp corners, the profiles of stresses are discontinued, and that poses a great challenge for approximation methods. The problem has attracted many researchers. For example, Marchal and Crochet (1986, 1987) used FEM with Hermitian elements; Yoo and Na (1991) used finite volume SIMPLER algorithm with non-uniform staggered grid to simulate the flow with Weissenberg number We up to 1.04; Carew et al. (1993) implemented a method with Taylor-Petrov-Galerkin algorithm to simulate the flow at several Reynolds numbers Re; Sato and Richardson (1994) used a mixed finite element/finite volume method with a pressure-correction scheme to simulate flow at $Re = \{0.01, 0.1, 1\}$ and We up to 2; Matallah et al. (1998) proposed the recovery and stress-splitting scheme within a finite-element formulation to investigate viscoelastic flows; Phillips and Williams (1999) applied a semi-Lagrangian finite volume method for analysis of creeping and inertial flow. Recently, Fattal and Kupferman (2004) proposed log-conformation tensor approach which was a good treatment for the exponential behaviour of stresses near the re-entrant corners. In this chapter the present parallel IRBF method based on overlapping Domain Decomposition method is demonstrated in simulating the flows of Newtonian and

non-Newtonian fluids through a 4:1 planar contraction.

The domain of a planar contraction problem consists of an upstream channel and a downstream channel. The two channels are connected by a sharp re-entrant. The contraction ratio, which is the ratio between the widths of the upstream and downstream channels respectively, is chosen to be 4:1. The flow of fluid through this contraction is driven by a given inlet velocity profile u. The geometry of the problem can be seen in Fig. 7.1.

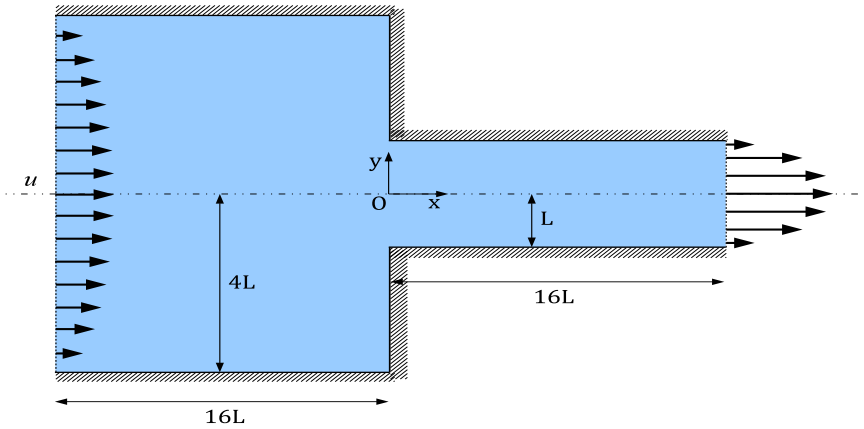


Figure 7.1 The 4:1 planar contraction geometry.

7.2 Governing equations and boundary conditions

7.2.1 Governing equations

The dimensionless equations of mass and momentum conservation are given by, respectively,

$$\nabla \cdot \mathbf{v} = 0, \tag{7.1}$$

$$\frac{\partial \mathbf{v}}{\partial t} + Re\mathbf{v} \cdot \nabla \mathbf{v} = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \alpha \nabla^2 \mathbf{v}, \tag{7.2}$$

and the constitutive equation for the elastic part of the stress tensor $\boldsymbol{\tau}$ for Oldroyd-B fluid is expressed by

$$\boldsymbol{\tau} + We\left(\frac{\partial \boldsymbol{\tau}}{\partial t} + \mathbf{v} \cdot \nabla \boldsymbol{\tau} - \boldsymbol{\tau} \cdot \nabla \mathbf{v} - \nabla \mathbf{v}^T \cdot \boldsymbol{\tau}\right) = 2(1-\alpha)(\nabla \mathbf{v} + \nabla \mathbf{v}^T), \tag{7.3}$$

where α is the ratio of the retardation to relaxation times of the fluid $\left(\alpha = \dfrac{\lambda_2}{\lambda_1}\right)$, and its value is taken to be 1/9. Re and We are the Reynolds number and Weissenberg number, respectively, and given by $Re = \dfrac{\rho U L}{\eta}$, $We = \dfrac{\lambda_1 U}{L}$, with L is taken as half of the width of the downstream channel and U is the mean velocity of the downstream channel.

In a 2D Cartesian coordinate system, Eqs. (7.2)- (7.3) are presented as follows.

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0, \quad (7.4)$$

$$\frac{\partial u}{\partial t}+Re\left(u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\right)=-\frac{\partial p}{\partial x}+\alpha\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\frac{\partial \tau_{xx}}{\partial x}+\frac{\partial \tau_{xy}}{\partial y}, \quad (7.5)$$

$$\frac{\partial v}{\partial t}+Re\left(u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}\right)=-\frac{\partial p}{\partial y}+\alpha\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+\frac{\partial \tau_{xy}}{\partial x}+\frac{\partial \tau_{yy}}{\partial y}, \quad (7.6)$$

$$\tau_{xx}+We\left(\frac{\partial \tau_{xx}}{\partial t}+u\frac{\partial \tau_{xx}}{\partial x}+v\frac{\partial \tau_{xx}}{\partial y}-2\frac{\partial u}{\partial x}\tau_{xx}-2\frac{\partial u}{\partial y}\tau_{xy}\right)=2(1-\alpha)\frac{\partial u}{\partial x}, \quad (7.7)$$

$$\tau_{xy}+We\left(\frac{\partial \tau_{xy}}{\partial t}+u\frac{\partial \tau_{xy}}{\partial x}+v\frac{\partial \tau_{xy}}{\partial y}-\frac{\partial v}{\partial x}\tau_{xx}-\frac{\partial u}{\partial y}\tau_{yy}\right)=(1-\alpha)\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right), \quad (7.8)$$

$$\tau_{yy}+We\left(\frac{\partial \tau_{yy}}{\partial t}+u\frac{\partial \tau_{yy}}{\partial x}+v\frac{\partial \tau_{yy}}{\partial y}-2\frac{\partial v}{\partial x}\tau_{xy}-2\frac{\partial v}{\partial y}\tau_{yy}\right)=2(1-\alpha)\frac{\partial v}{\partial y}. \quad (7.9)$$

7.2.2 Boundary conditions

The boundary condition of the 4:1 contraction flow are described in Phillips and Williams (1999) as follows.

At the inlet, the fully developed velocity profile is employed and the stresses are inferred from the constitutive equation

$$\begin{aligned} u &= \frac{3}{128}(16-y^2), \\ v &= 0, \\ \tau_{xx} &= 2We(1-\alpha)\left(\frac{\partial u}{\partial y}\right)^2, \\ \tau_{yy} &= 0, \\ \tau_{xy} &= (1-\alpha)\frac{\partial u}{\partial y}. \end{aligned} \quad (7.10)$$

The non-slip boundary condition is imposed on the solid walls, and thus the equations for stresses are simplified.

On the solid walls parallel to the x-axis

$$\begin{aligned} u &= v=\frac{\partial v}{\partial x}=\frac{\partial v}{\partial y}=\frac{\partial u}{\partial x}=0, \\ \tau_{xx} &= 2We(1-\alpha)\left(\frac{\partial u}{\partial y}\right)^2, \\ \tau_{xy} &= (1-\alpha)\frac{\partial u}{\partial y}, \\ \tau_{yy} &= 0. \end{aligned} \quad (7.11)$$

On solid walls parallel to the y-axis

$$\begin{aligned} u &= v = \frac{\partial u}{\partial y} = \frac{\partial v}{\partial y} = \frac{\partial u}{\partial x} = 0, \\ \tau_{xx} &= 0, \\ \tau_{xy} &= (1-\alpha)\frac{\partial v}{\partial x}, \\ \tau_{yy} &= 2We(1-\alpha)\left(\frac{\partial v}{\partial x}\right)^2. \end{aligned} \tag{7.12}$$

At the outlet, the value of pressure is fixed at a value of zero, and the fully developed profile for velocity and the first derivatives with respect to the x-direction of extra stresses are given

$$\frac{\partial \tau_{xx}}{\partial x} = \frac{\partial \tau_{xy}}{\partial x} = \frac{\partial \tau_{yy}}{\partial x} = 0. \tag{7.13}$$

In some papers (Marchal and Crochet, 1986; Matallah et al., 1998) the Deborah number is used instead of Weissenberg number. For comparison purpose, a conversion between these two characteristic numbers is provided as follows.

$$De = 3We = \lambda_1 \dot{\gamma}, \tag{7.14}$$

where $\dot{\gamma}$ is the fully developed downstream wall shear rate and $\dot{\gamma} = 3$ for a unit flow rate through the downstream channel.

7.2.3 The re-entrant corners

As can be figured out from the boundary conditions Eq. (7.11)-(7.13), the extra stresses are discontinued at the re-entrant corners. There are several ways to treat the values of stresses at these positions. Yoo and Na (1991) adopted following expressions

$$\begin{aligned} u_y &= (0 - u_c)/\Delta y, \\ v_x &= (0 - v_c)/\Delta x, \end{aligned} \tag{7.15}$$

where u_c and v_c are the velocity components at the closest nodes to the re-entrant corner for u and v respectively, Δx and Δy being corresponding distances as shown in Fig. 7.2.

Phillips and Williams (1999) used a similar approach which is to average the shear stresses at two extra points taken inside the domain at some distance from the re-entrant corners.

In this work, the stresses at the re-entrant corners are simply ignored. Because stresses are discontinued, there will be no correct value of stresses at these corners. It is noted that in Eqs. (7.5) and (7.6) the unknowns u, v are located inside the domain, therefore all discrete equations for u, v will be constructed inside domain, where the value of extra stresses is available.

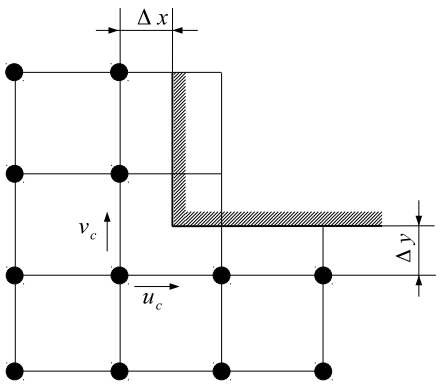


Figure 7.2 Staggered grid near the re-entrant corner

7.2.4 Projection method

In this work, the projection method is applied in the solution algorithm as follows.

Splitting Eq. (7.2) into two equations and discretising the derivative of $\mathbf{v}$ with respect to time by forward difference results in

$$\frac{\mathbf{v}^* - \mathbf{v}^k}{\Delta t} + Re\mathbf{v}^k \cdot \nabla \mathbf{v}^k = \nabla \cdot \tau + \alpha \nabla^2 \mathbf{v}^*, \tag{7.16}$$

$$\frac{\mathbf{v}^{k+1} - \mathbf{v}^*}{\Delta t} = -\nabla \widetilde{p}^{k+1}. \tag{7.17}$$

The velocity at time step $k+1$ is obtained from Eq. (7.17) as follows

$$\mathbf{v}^{k+1} = -\Delta t \nabla \widetilde{p}^{k+1} + \mathbf{v}^*. \tag{7.18}$$

It is noted that $\widetilde{p}$ now is a pseudo pressure, which is not the actual pressure but a correction to the divergence-free condition.

By taking the divergence of Eq. (7.18) and enforcing $\nabla \mathbf{v}^{k+1} = 0$, the pressure Poisson's equation (PPE) is obtained

$$\nabla^2 \widetilde{p}^{k+1} = \frac{\nabla \mathbf{v}^*}{\Delta t}. \tag{7.19}$$

7.2.5 Boundary condition for the PPE

It is clearly seen from the PPE (7.19) that the boundary condition for pressure can be derived from the boundary condition for velocity. First of all, we have

$$\mathbf{v}^* = \mathbf{v}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \Gamma$$

So Eq. (7.17) will become

$$\frac{\mathbf{v}^{k+1}}{\Delta t} = -\nabla \widetilde{p}, \quad \mathbf{x} \in \Gamma \tag{7.20}$$

Then projecting (7.20) on the normal vector yields

$$\frac{\hat{\mathbf{n}} \cdot \mathbf{v}^{k+1}}{\Delta t} = \hat{\mathbf{n}} \cdot \nabla \widetilde{p}^{k+1}, \quad \mathbf{x} \in \Gamma$$

Taking into account the divergence-free condition for velocity, one gets

$$\hat{\mathbf{n}} \cdot \nabla \widetilde{p}^{k+1} = 0, \quad \mathbf{x} \in \Gamma$$

or

$$\frac{\partial \widetilde{p}^{k+1}}{\partial n} = 0, \quad \mathbf{x} \in \Gamma \tag{7.21}$$

This is the homogeneous Neumann boundary condition for pressure.

7.3 Review of parallel domain decomposition local IRBF approach

To simulate the planar contraction flow problem, local IRBF schemes as presented in previous chapters are employed. The 2D local IRBF approximation scheme with a 9-point stencil is used to approximate the field variable and its derivatives while the boundary conditions are directly imposed using the 1D-IRBF approximation.

Consider an arbitrary interior node $\mathbf{x}_a$ in the domain, and its associated stencil locally numbered as

$$\begin{bmatrix} \mathbf{x}_3 & \mathbf{x}_6 & \mathbf{x}_9 \\ \mathbf{x}_2 & \mathbf{x}_5 & \mathbf{x}_8 \\ \mathbf{x}_1 & \mathbf{x}_4 & \mathbf{x}_7 \end{bmatrix},$$

where $\mathbf{x}_a \equiv \mathbf{x}_5$.

The procedure of collocating u on the stencil leads to an algebraic equation system as follows.

$$\begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \end{pmatrix} = \underbrace{\begin{bmatrix} \mathcal{G}_x^{[0]}, & \mathcal{O} \\ \mathcal{G}_x^{[0]}, & -\mathcal{G}_y^{[0]} \end{bmatrix}}_{\mathcal{C}} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C} \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}, \tag{7.22}$$

where $\mathcal{C}$ is the conversion matrix, $\widetilde{\mathbf{w}}_x$ and $\widetilde{\mathbf{w}}_y$ the RBF weight vectors of length 15; $\widetilde{\mathbf{u}}$ the vector of length 9, and $\widetilde{\mathbf{0}}$ the zeros vector of length 9; $\mathcal{O}$ the zeros matrix of dimension 9×15, and $\mathcal{G}_x^{[0]}$ and $\mathcal{G}_y^{[0]}$ the known matrices of dimensions 9×15. Furthermore, $\widetilde{\mathbf{u}}$, $\widetilde{\mathbf{w}}_x$ and $\widetilde{\mathbf{w}}_y$ are given by

$$\widetilde{\mathbf{u}} = (u_1, \ldots, u_9)^T, \tag{7.23}$$

$$\widetilde{\mathbf{w}}_x = (w_{x_1}, \ldots, w_{x_9}, C_1^x(y_1), C_1^x(y_2), C_1^x(y_3), C_2^x(y_1), C_2^x(y_2), C_2^x(y_3))^T, \tag{7.24}$$

$$\widetilde{\mathbf{w}}_y = (w_{y_1}, \ldots, w_{y_9}, C_1^y(x_1), C_1^y(x_2), C_1^y(x_3), C_2^y(x_1), C_2^y(x_2), C_2^y(x_3))^T, \tag{7.25}$$

$$\mathcal{G}_x^{[0]} = \begin{bmatrix} G_{1,x}^{[0]}(\mathbf{x}_1) & \cdots & G_{9,x}^{[0]}(\mathbf{x}_1) & x_1 & 0 & 0 & 1 & 0 & 0 \\ G_{1,x}^{[0]}(\mathbf{x}_2) & \cdots & G_{9,x}^{[0]}(\mathbf{x}_2) & 0 & x_2 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & x_3 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & x_4 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & x_5 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & x_6 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & x_7 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & x_8 & 0 & 0 & 1 & 0 \\ G_{1,x}^{[0]}(\mathbf{x}_9) & \cdots & G_{9,x}^{[0]}(\mathbf{x}_9) & 0 & 0 & x_9 & 0 & 0 & 1 \end{bmatrix}, \tag{7.26}$$

$$\mathcal{G}_y^{[0]} = \begin{bmatrix} G_{1,y}^{[0]}(\mathbf{x}_1) & \cdots & G_{9,y}^{[0]}(\mathbf{x}_1) & y_1 & 0 & 0 & 1 & 0 & 0 \\ G_{1,y}^{[0]}(\mathbf{x}_2) & \cdots & G_{9,y}^{[0]}(\mathbf{x}_2) & y_2 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & y_3 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & y_4 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & y_5 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & y_6 & 0 & 0 & 1 & 0 \\ \vdots & \ddots & \vdots & 0 & 0 & y_7 & 0 & 0 & 1 \\ \vdots & \ddots & \vdots & 0 & 0 & y_8 & 0 & 0 & 1 \\ G_{1,y}^{[0]}(\mathbf{x}_9) & \cdots & G_{9,y}^{[0]}(\mathbf{x}_9) & 0 & 0 & y_9 & 0 & 0 & 1 \end{bmatrix}, \tag{7.27}$$

where $G_{i,x}^{[0]}$ and $G_{i,y}^{[0]}$ $(i = 1..9)$ were defined previously in the x and y-directions and x_i, y_i two components of $\mathbf{x}_i$. It is noted that in Eq. (7.22),

$$\widetilde{\mathbf{u}} = [\mathcal{G}_x^{[0]}, \mathcal{O}] \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}$$

is obtained by collocating the field variable over a local stencil, and

$$\widetilde{\mathbf{0}} = [\mathcal{G}_x^{[0]}, -\mathcal{G}_y^{[0]}] \begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix}$$

is derived from the consistency condition $\int\int \frac{\partial^2 u}{\partial x^2} dx \Big|_{\mathbf{x}_i} = \int\int \frac{\partial^2 u}{\partial y^2} dy \Big|_{\mathbf{x}_i}$.

The conversion of the network-weight space into the physical space is achieved by inverting Eq. (7.22)

$$\begin{pmatrix} \widetilde{\mathbf{w}}_x \\ \widetilde{\mathbf{w}}_y \end{pmatrix} = \mathcal{C}^{-1} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\mathbf{0}} \end{pmatrix}. \tag{7.28}$$

Eq. (7.28) is used to approximate the first order derivatives of u with respect to x and y and the function itself over a local stencil. More details can be found in previous chapters.

7.4 Parallel implementation

In solving the contraction flow problem, parallelisation is based on the Schwarz overlapping DD method with the first order continuity condition on artificial boundaries (ABs). An example of sub-domain formation of the problem is presented in Fig. 7.3. In this figure the domain is divided in to 20 equal sub-domains. These sub-domains are assigned a unique index number, called process-index, from 1 to 20. They also overlap with their neighbouring sub-domains (NSs).

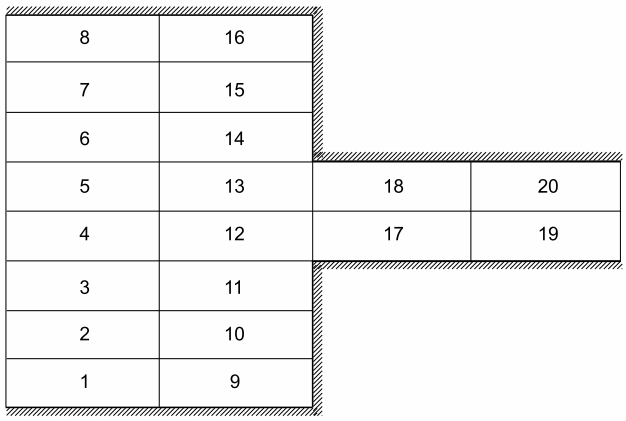


Figure 7.3 The 4:1 planar contraction flow problem. Sub-domains are overlapping even though the overlapped zones are not shown.

All variables including the velocity, pressure and extra stresses are the object of data transmission. After each iteration, each process needs to exchange the updated value of the above mentioned variables with its neighbouring process. Any process has from one to four neighbours depends on its position in the domain. Processes use process-index to a process to determine its neighbours and communicate with them.

For a sequential algorithm, the convergence measurement is defined as follows

$$CM = \frac{\sqrt{\sum_{i=1}^{n}(u_i^{k+1} - u_i^k)^2}}{\sqrt{\sum_{i=1}^{n}(u_i^k)^2}} \tag{7.29}$$

where k is the iteration step. When the CM of an iteration step reaches a predefined tolerance CM_{tol}, the algorithm is stopped.

With parallel algorithm, aside from the CM inside each sub-domain, the convergence measurement on the artificial boundary ($ABCM$) also needs to be verified. Hence, the algorithm is considered converged if the following conditions are satisfied

$$\begin{cases} CM & < & CM_{tol}, & \text{in sub-domain} \\ ABCM & < & ABCM_{tol}, & \text{on AB} \end{cases} \tag{7.30}$$

where $ABCM_{tol}$ is a predefined tolerance of CM on AB.

Since parallel processes form a distributed system, the termination of this system requires a dedicated distributed termination detection (DTD) algorithm. In this simulation, the Bitmap DTD (Chapter 3) is used thanks to its low message complexity and data overhead.

In a parallel algorithm, in each sub-domain a boundary value problem is defined. The algorithm in each sub-domain can be described as follows.

1. Guess initial values on ABs.
2. Solve the sub-domain flow problems using local IRBF method.
   (a) Calculate the extra-stress in Eq. (7.7) - (7.9) using forward first order Euler difference scheme.
   (b) Solve Eq. (7.16) for intermediate value $\mathbf{v}^*$ with vanishing pressure derivative. It is worth noting that time derivative is discretised using the forward difference scheme, the convection term and diffusion term are treated explicitly and implicitly, respectively.
   (c) Solve the PPE (7.19).
   (d) Correct $\mathbf{v}^{k+1}$ using obtained value of pressure by Eq. (7.18)
3. Exchange the values of u, v, p and extra stresses with neighbouring processes.
4. Check the convergence condition (7.30). If the procedure is not yet converged, go to step 2. Otherwise, stop the procedure.

7.5 Numerical results

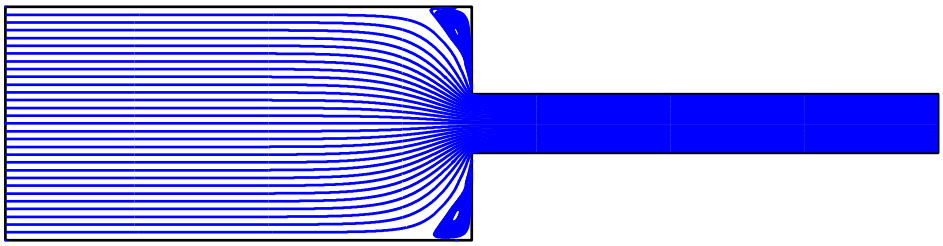
The efficiency of the parallel method is demonstrated by simulating the 4:1 contraction flow of both Newtonian and non-Newtonian fluid using several grids (Table 7.1).

Table 7.1 Grid characteristics

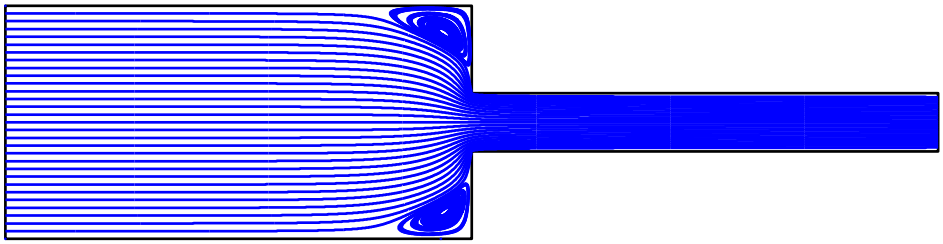
Grid	DoF	Δx	Δy
1	12603	0.20	0.20
2	39960	0.11	0.11
3	435603	0.02	0.02

7.5.1 The 4:1 contraction flow of Newtonian fluids

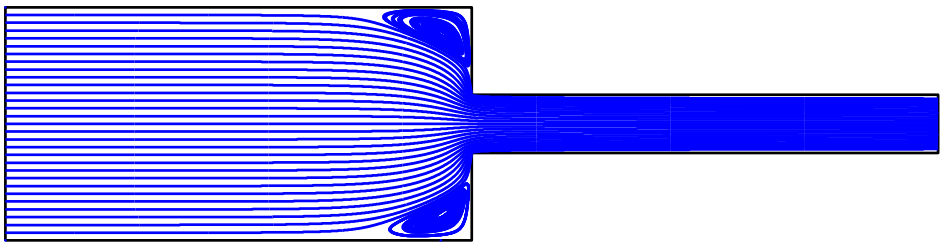
In the case of Newtonian fluid, a range of Reynolds numbers is considered $\{1, 100, 200, 300, 400, 500\}$. The streamlines for different Re numbers are depicted in Figs. (7.4) and (7.5). It is clear that the size and intensity of the corner vortices grow as the Reynolds number increases.



(a) $Re = 1$

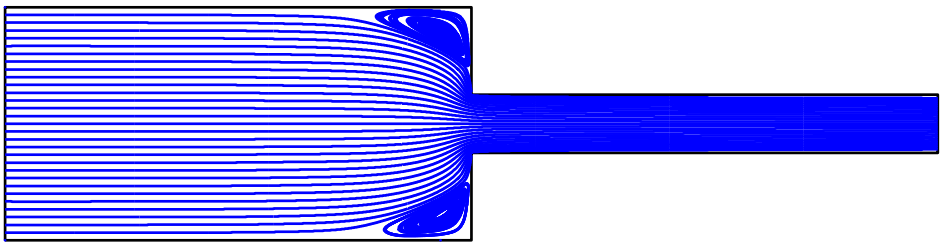


(b) $Re = 100$

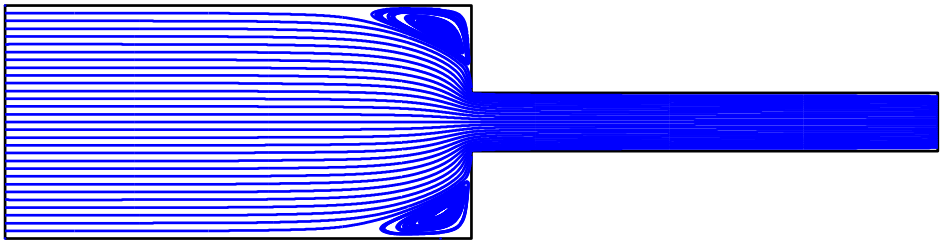


(c) $Re = 200$

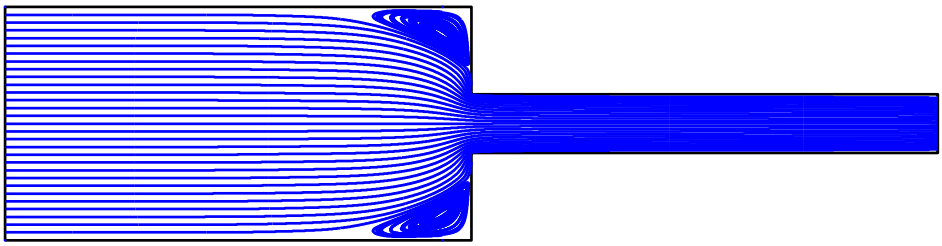
Figure 7.4 The 4:1 contraction flow of Newtonian fluids. Stream-function (ψ) contours of the flow for several Reynolds numbers ($Re = \{1, 100, 200\}$) by the present parallel method using 20 sub-domains, grid 3, $CM_{tol} = 10^{-8}$, $CM_{tol}^{AB} = 10^{-6}$, $\beta = 2$ and $\Delta t = 5 \times 10^{-3}$ for $Re = \{1, 100\}$ and 10^{-3} for $Re = 200$.



(a) $Re = 300$



(b) $Re = 400$



(c) $Re = 500$

Figure 7.5 The 4:1 contraction flow of Newtonian fluids. Stream-function (ψ) contours of the flow for several Reynolds numbers ($Re = \{300, 400, 500\}$) by the present parallel method using 20 sub-domains, grid 3 with the specifications $CM_{tol} = 10^{-8}$, $CM_{tol}^{AB} = 10^{-6}$, $\beta = 2$ and $\Delta t = 10^{-3}$.

To investigate the performance of the parallel method, a range of CPUs {20, 30, 40, 50, 60} is used to simulate the flow. The detailed results are provided in Tables (7.2) - (7.4). As the simulation is performed on a dense grid (large scale), the result by single CPU is not available for comparison. Instead, the result with lowest number of CPUs, which is 20 CPUs in this case, is used as a reference. Normally, speed-up and efficiency are defined as follows.

$$\begin{aligned} S &= \frac{T_s}{T_p}, \\ E &= \frac{S}{p}, \end{aligned} \tag{7.31}$$

where T_s is computation time on single CPU, T_p computation time on parallel CPUs, p is number of parallel CPUs. However, in the case where T_s is not available, it is symbolically calculated as

$$T_s = T_p^{[ref]} \times p^{[ref]} \tag{7.32}$$

where $T_p^{[ref]}$ and $p^{[ref]}$ are computation time and number of CPUs of a reference result.

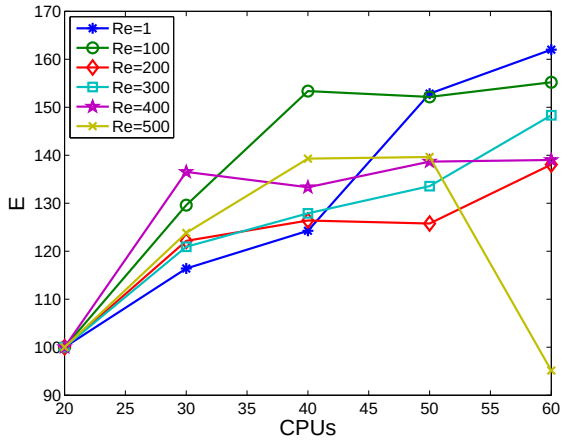
Table 7.2 The 4:1 contraction flow of Newtonian fluids. Parallel performance for $Re = \{1, 100\}$ with grid 3, $\Delta t = 5 \times 10^{-3}$, $CM_{tol} = 10^{-8}$, $ABCM_{tol} = 10^{-6}$, and $\beta = 2$. p: number of CPUs; N_i: number of iterations; T_p: parallel computation time (minutes); S: speed-up; E: efficiency. T_p with 20 CPUs is used for reference.

	$Re = 1$				$Re = 100$			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
20	5169	97.57	20.00	100.00	26908	506.36	20.00	100.00
30	5685	53.33	36.59	121.96	27554	260.45	38.88	129.61
40	5964	33.49	58.26	145.66	27044	165.08	61.35	153.37
50	6262	36.28	53.78	107.56	27812	133.12	76.08	152.16
60	6518	25.60	76.21	127.02	28543	108.75	93.12	155.20

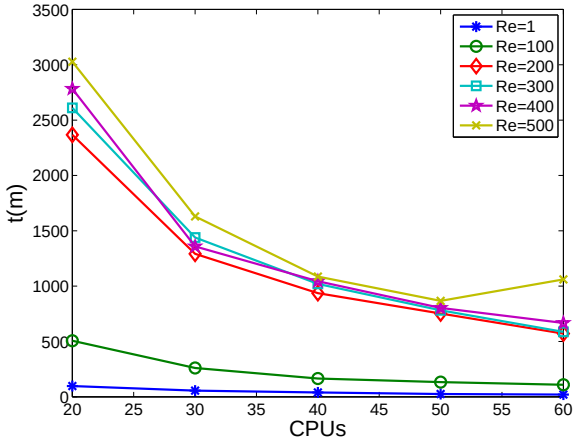
Table 7.3 The 4:1 contraction flow of Newtonian fluids. Parallel performance for $Re = \{200, 300\}$ with grid 3, $\Delta t = 10^{-3}$, other parameters are given in Table 7.2. T_p with 20 CPUs is used for reference.

	$Re = 200$				$Re = 300$			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
20	131605	2367.23	20.00	100.00	144017	2610.55	20.00	100.00
30	132370	1291.95	36.65	122.15	144480	1439.14	36.28	120.93
40	133272	936.33	50.56	126.41	145348	1020.57	51.16	127.90
50	133945	990.10	47.82	95.64	146077	781.88	66.78	133.55
60	134046	571.49	82.84	138.07	145276	586.68	88.99	148.32

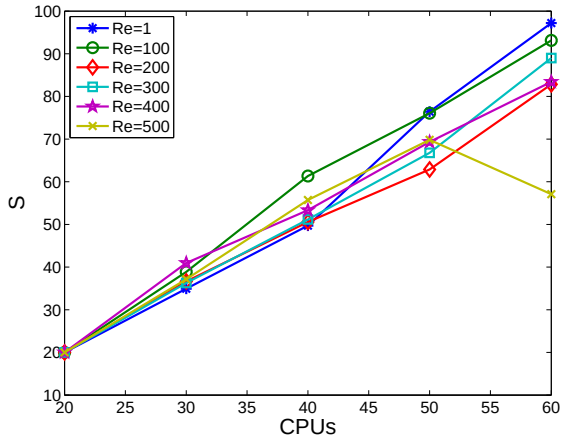
The visual presentation of efficiency, simulation time and speed-up of the parallel method is given in Fig. (7.6). It is clear that the speed-up grows linearly with the CPUs. However, as the Reynolds number gets larger this tendency begins to deteriorate. For example, with $Re = 500$ when the number of CPUs reaches 60 the speed-up drops to the same value as with 40 CPUs. This behaviour suggests that, high Reynolds numbers negatively affect not only the convergence of the sequential program but also the efficiency of parallel program.



(a) Efficiency



(b) Simulation time



(c) speed-up

Figure 7.6 The 4:1 contraction flow of Newtonian fluids. The efficiency (a), simulation time (b) and speed-up (c) of the present parallel method. Parameters are in given in Tables 7.2- 7.4

Table 7.4 The 4:1 contraction flow of Newtonian fluids. Parallel performance for $Re = \{400, 500\}$ with grid 3, $\Delta t = 10^{-3}$, other parameters are given in Table 7.2. T_p with 20 CPUs is used for reference.

	$Re = 400$				$Re = 500$			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
20	153723	2783.26	20.00	100.00	165370	3027.22	20.00	100.00
30	154001	2323.50	23.96	79.86	168315	1629.79	37.15	123.83
40	154732	1043.80	53.33	133.32	161257	1086.56	55.72	139.30
50	155386	802.78	69.34	138.68	183722	867.19	69.82	139.63
60	169742	667.34	83.41	139.02	161525	1060.51	57.09	95.15

7.5.2 The 4:1 contraction flow of an Oldroyd-B fluid

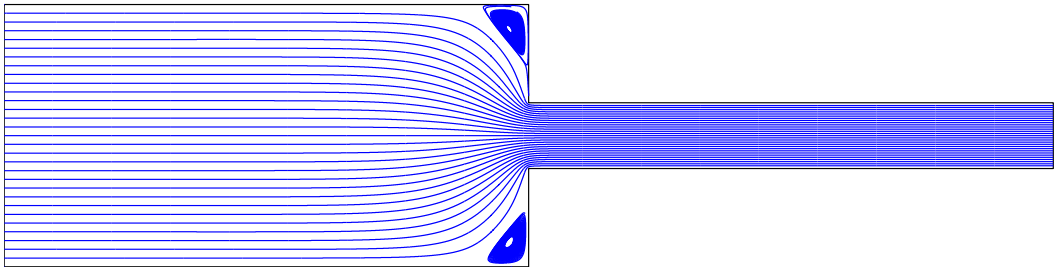
The contraction flow of an Oldroyd-B fluid is simulated at several Weissengberg numbers ranging from 0.1 to 1 using several grids as presented in Table 7.1. For example, grid 3 is used for $We = \{0.1, 0.2, 0.3\}$, grid 2 for $We = \{0.4, 0.5\}$ and grid 1 for $We = \{0.8, 0.9, 1\}$. The contours of stream-function and stresses of the flow are shown in Figs. (7.7) - (7.18).

The contours of stream-function and extra stresses are in good agreement with other results including those by Phillips and Williams (1999); Alves et al. (2003); Kim et al. (2005a).

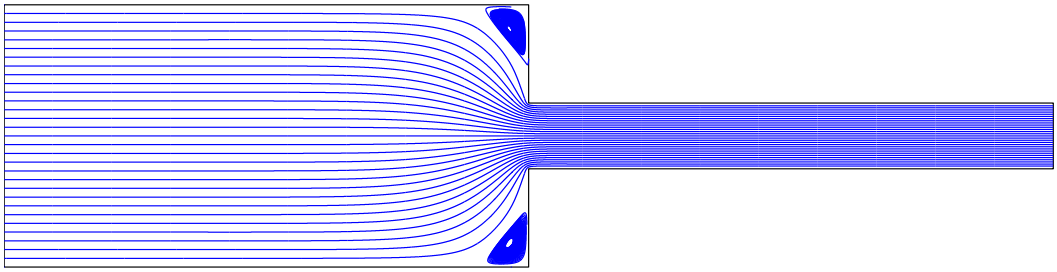
For the contraction flow of Oldroyd-B fluids, the simulation is performed using only grids 2 and 3. The simulation time, speed-up and efficiency of the method are provided in detail in Tables (7.5) - (7.7). Their visual presentation are also given in Figs. (7.19). As in the case of Newtonian fluids, a good scalability of the algorithm is also achieved as the speed-up grows linearly with the number of CPUs. However, the quality of speed-up deteriorates somewhat as We increases.

Table 7.5 The 4:1 contraction flow of Oldroyd-B fluid. Parallel performance with grid 3, $\Delta t = 10^{-4}$, $CM_{tol} = 10^{-8}$, $ABCM_{tol} = 10^{-6}$, and $\beta = 2$. N_i - number of iterations; T_p - parallel computation time (minutes); S - speed-up; E - efficiency. T_p with 20 CPUs is used as reference.

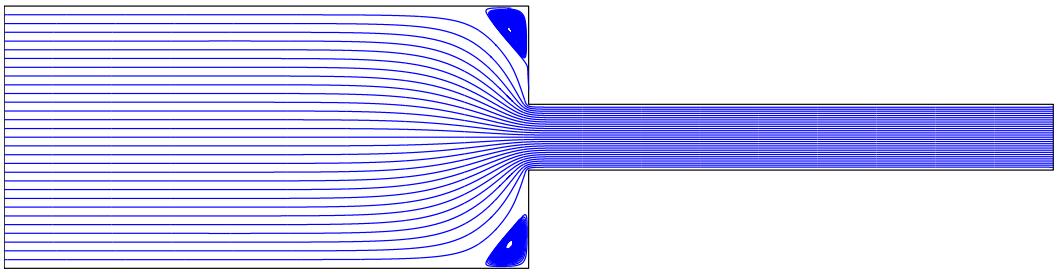
	$We = 0.1$				$We = 0.2$			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
20	62888	2980.02	20.00	100.00	56448	2830.67	20.00	100.00
30	63310	1549.78	38.46	128.19	59316	1527.25	37.07	123.56
40	62807	1035.25	57.57	143.93	60720	1034.28	54.74	136.84
50	63443	889.35	67.02	134.03	62222	737.07	76.81	153.62
60	63310	608.83	97.89	163.15	63693	526.35	107.56	179.26



(a) $We = 0.1$



(b) $We = 0.2$

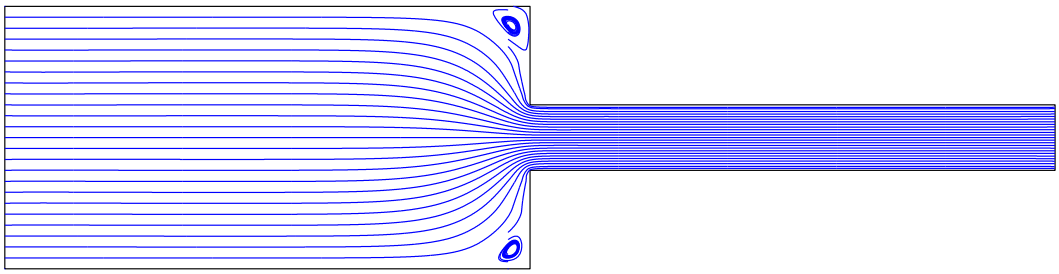


(c) $We = 0.3$

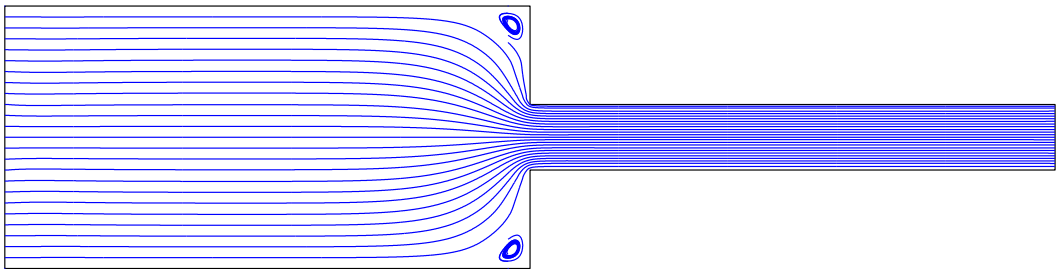
Figure 7.7 The 4:1 contraction flow of Oldroyd-B fluid. Stream-function (ψ) contours of the flow for $We = \{0.1, 0.2, 0.3\}$ by the present parallel method using 20 CPUs, grid 3, $\Delta t = 10^{-3}$, $CM_{tol} = 10^{-8}$, $ABCM_{tol} = 10^{-6}$ and $\beta = 2$.

Table 7.6 The 4:1 contraction flow of Oldroyd-B fluid. Parallel performance with grid 3, $\Delta t = 10^{-4}$, $CM_{tol} = 10^{-8}$, $ABCM_{tol} = 10^{-6}$, and $\beta = 2$. N_i - number of iterations; T_p - parallel computation time (minutes); S - speed-up; E - efficiency. T_p with 20 CPUs is used as reference.

	$We = 0.3$			
CPUs	N_i	T_p	S	E
20	75341	3481.00	20.00	100.00
30	80137	2057.33	33.84	112.80
40	83356	1586.35	43.89	109.72
50	89155	1207.91	57.64	115.27
60	91904	939.96	74.07	123.45



(a) $We = 0.4$

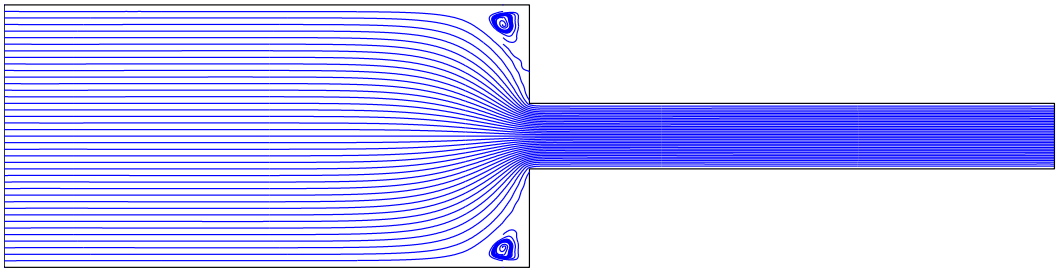


(b) $We = 0.5$

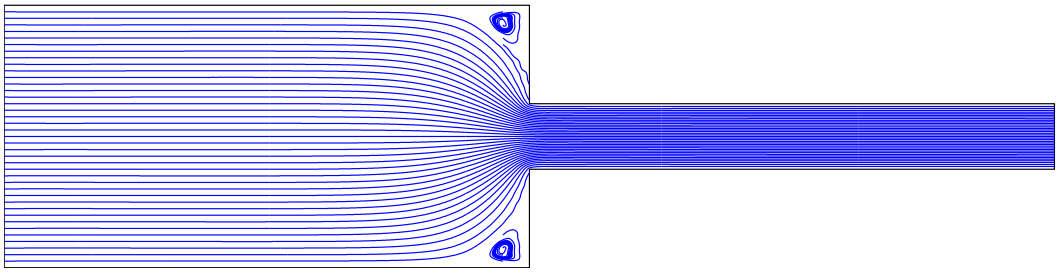
Figure 7.8 The 4:1 contraction flow of Oldroyd-B fluid. Stream-function (ψ) contours of the flow for $We = \{0.4, 0.5\}$ by the present parallel method using 10 CPUs, grid 2, $\Delta t = 10^{-3}$, $CM_{tol} = 10^{-8}$, $ABCM_{tol} = 10^{-6}$ and $\beta = 2$.

Table 7.7 The 4:1 contraction flow of Oldroyd-B fluid. Parallel performance with grid 2, $\Delta t = 10^{-3}$, $CM_{tol} = 10^{-8}$, $CM_{tol} = 10^{-6}$, and $\beta = 2$. N_i - number of iterations; T_p - parallel computation time (minutes) except for the case with 1 CPU which is non-parallel and $T_p \equiv T_s$; S - speed-up; E - efficiency.

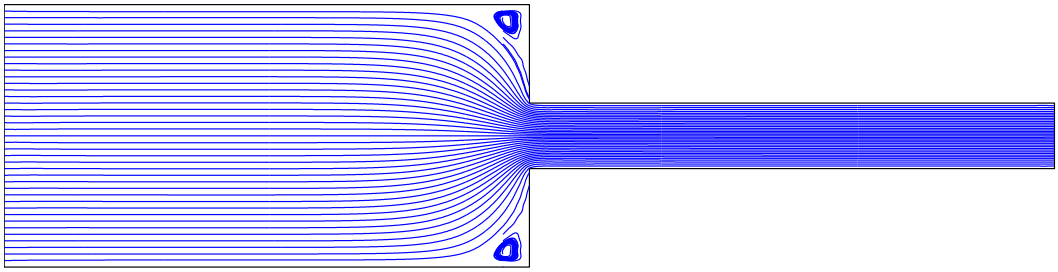
	$We = 0.4$				$We = 0.5$			
CPUs	N_i	T_p	S	E	N_i	T_p	S	E
1	7431	203.36	1.00	100.00	9549	258.14	1.00	100.00
10	10235	66.58	3.05	30.54	23407	136.53	1.89	18.91
20	10844	12.03	16.91	84.55	20484	22.58	9.00	45.02
30	10983	7.01	29.00	96.68	19161	11.19	18.18	60.59
40	10587	4.17	48.73	121.83	23107	8.68	23.43	58.58
50	11383	3.08	65.93	131.86	26522	6.48	31.38	62.77



(a) $We = 0.8$

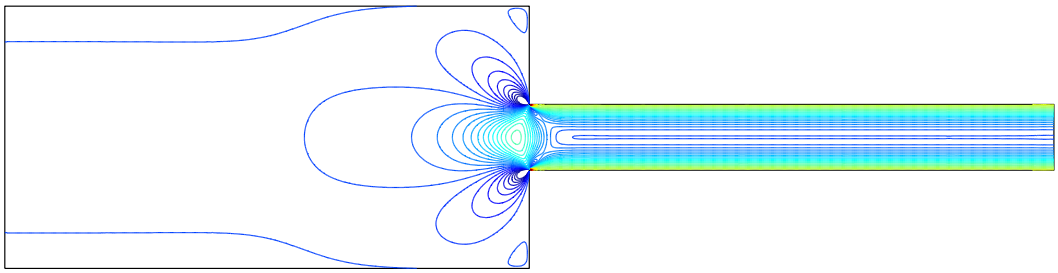


(b) $We = 0.9$

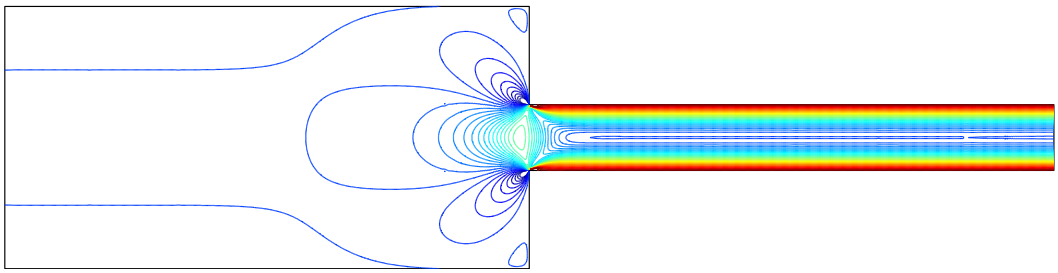


(c) $We = 1.0$

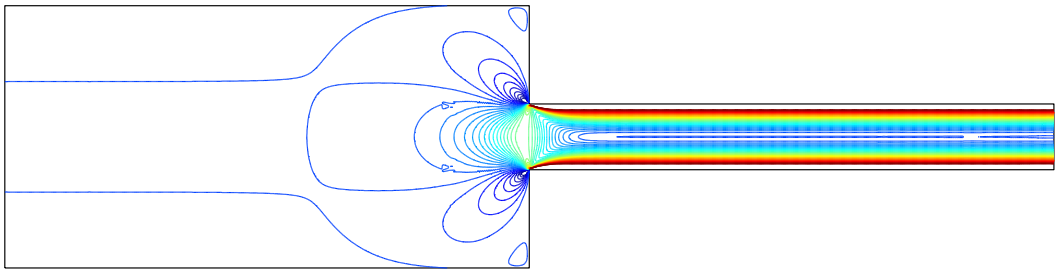
Figure 7.9 The 4:1 contraction flow of Oldroyd-B fluid. Stream-function (ψ) contours of the flow for $We = \{0.8, 0.9, 1\}$ by the present parallel method using 1 CPU, grid 1, $\Delta t = 10^{-3}$, $CM_{tol} = 10^{-8}$, $ABCM_{tol} = 10^{-6}$ and $\beta = 2$.



(a) $We = 0.1$

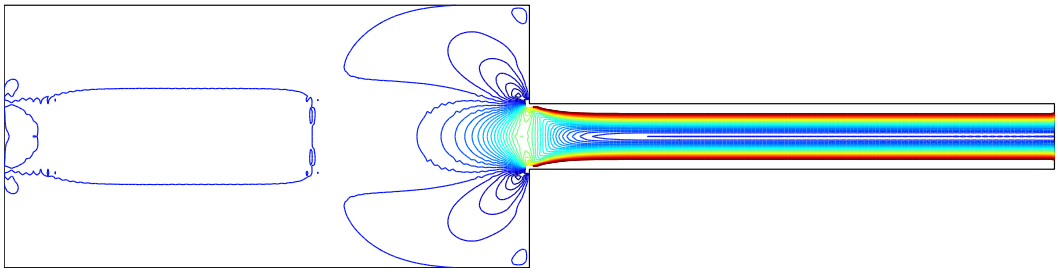


(b) $We = 0.2$

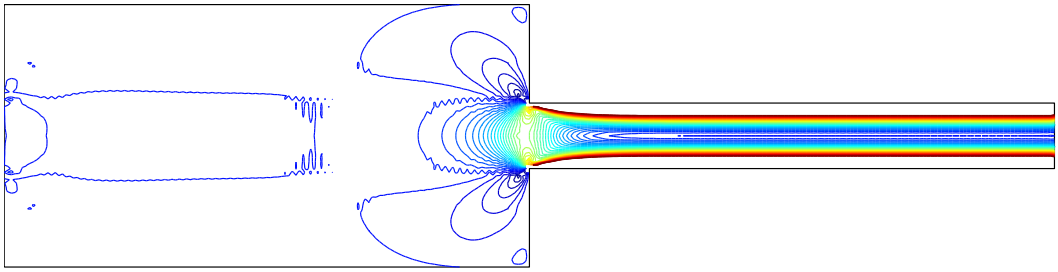


(c) $We = 0.3$

Figure 7.10 The 4:1 contraction flow of Oldroyd-B fluid. τ_{xx} contours of the flow for $We = \{0.1, 0.2, 0.3\}$ with parameters given in Fig. 7.7.

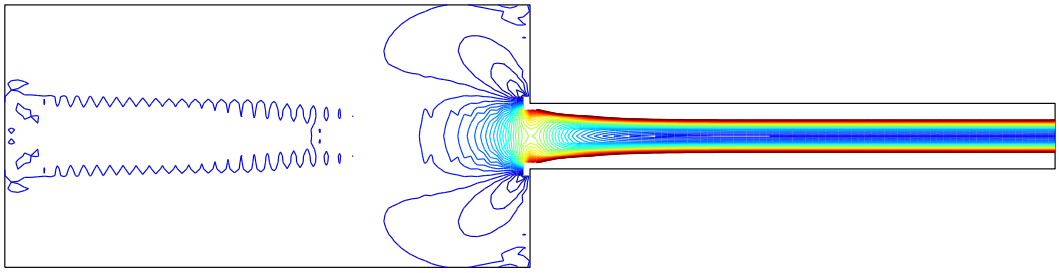


(a) $We = 0.4$

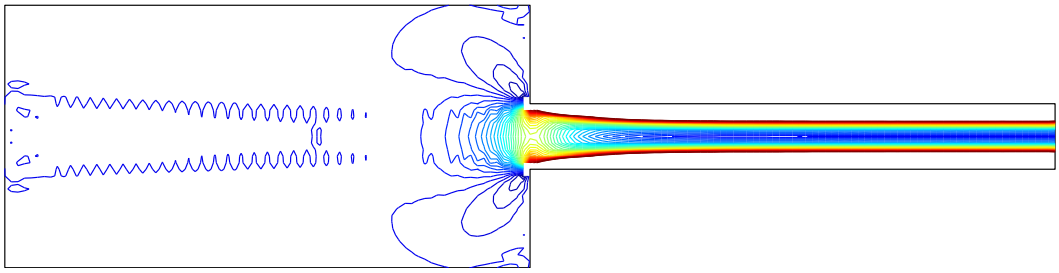


(b) $We = 0.5$

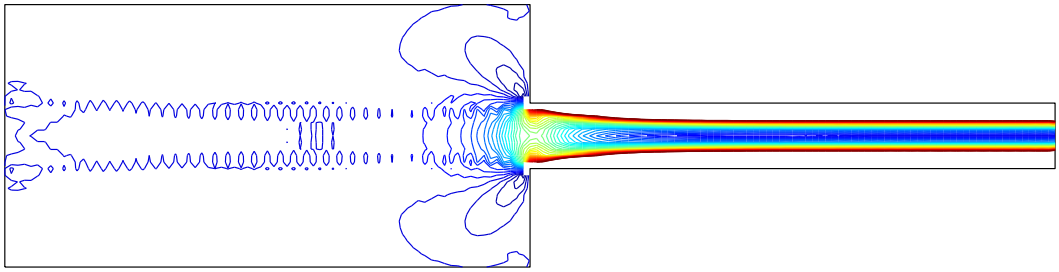
Figure 7.11 The 4:1 contraction flow of Oldroyd-B fluid. τ_{xx} contours of the flow for $We = \{0.4, 0.5\}$ with parameters given in Fig. 7.8.



(a) $We = 0.8$

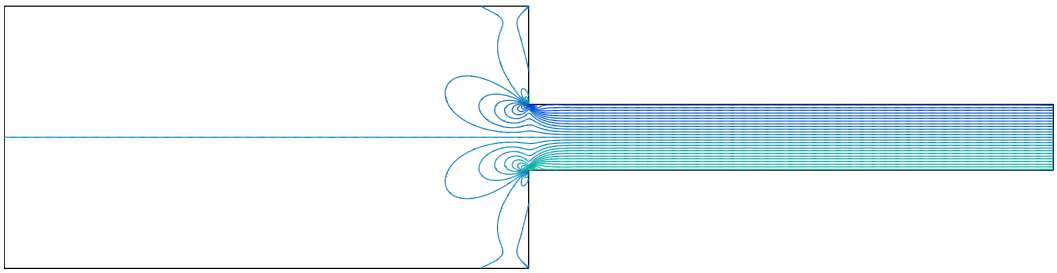


(b) $We = 0.9$

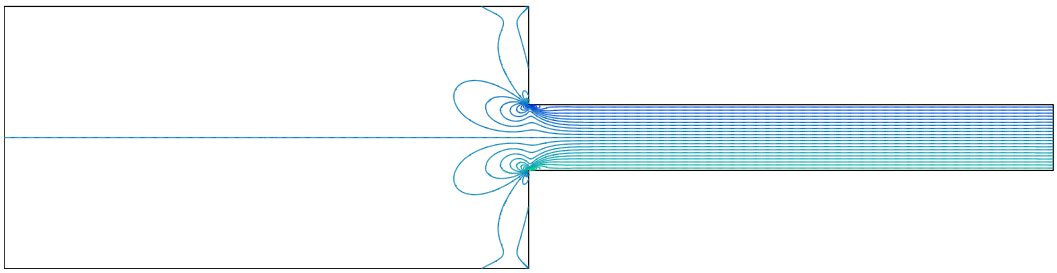


(c) $We = 1.0$

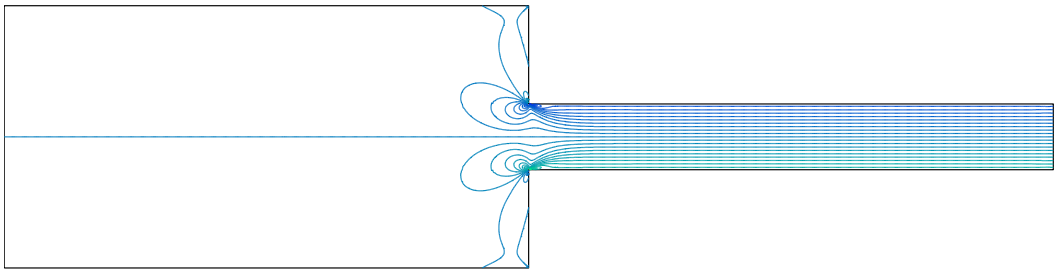
Figure 7.12 The 4:1 contraction flow of Oldroyd-B fluid. τ_{xx} contours of the flow for $We = \{0.8, 0.9, 1\}$ with parameters given in Fig. 7.9.



(a) $We = 0.1$

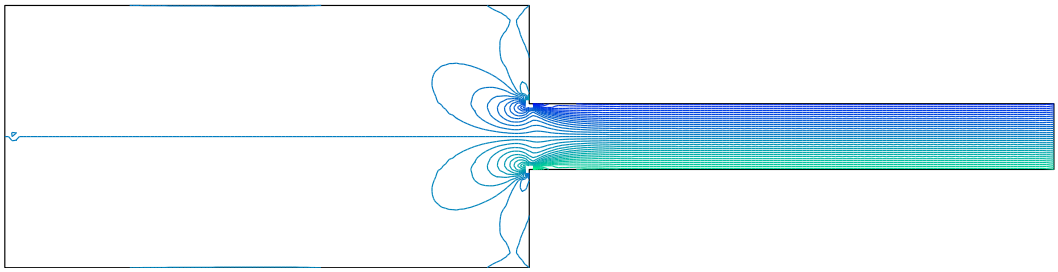


(b) $We = 0.2$

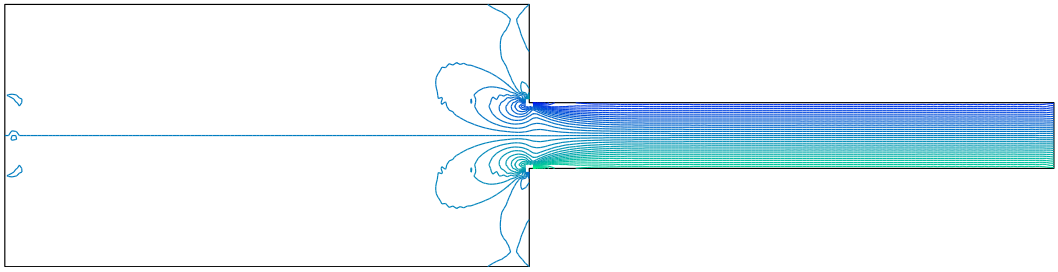


(c) $We = 0.3$

Figure 7.13 The 4:1 contraction flow of Oldroyd-B fluid. τ_{xy} contours of the flow for $We = \{0.1, 0.2, 0.3\}$ with parameters given in Fig. 7.7.

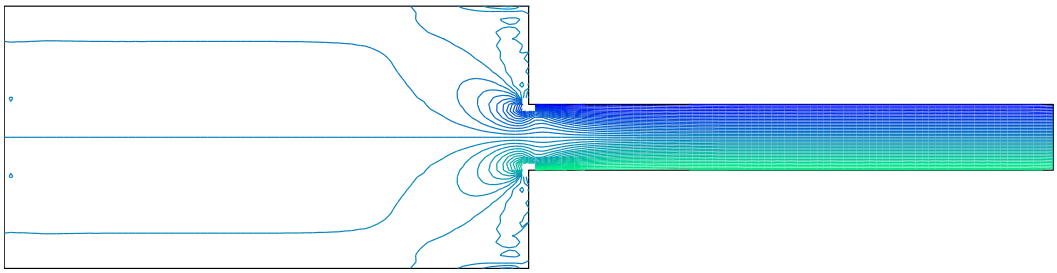


(a) $We = 0.4$

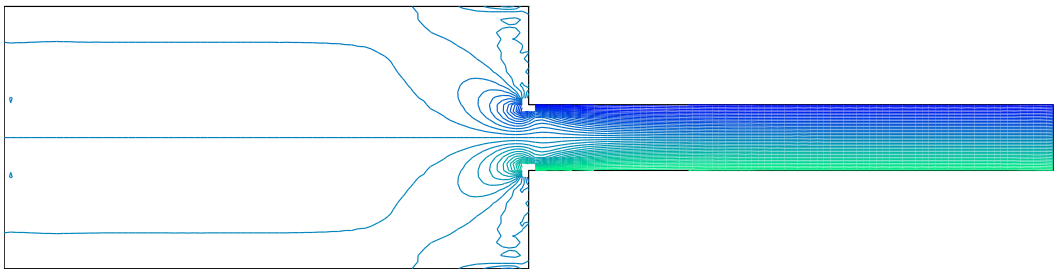


(b) $We = 0.5$

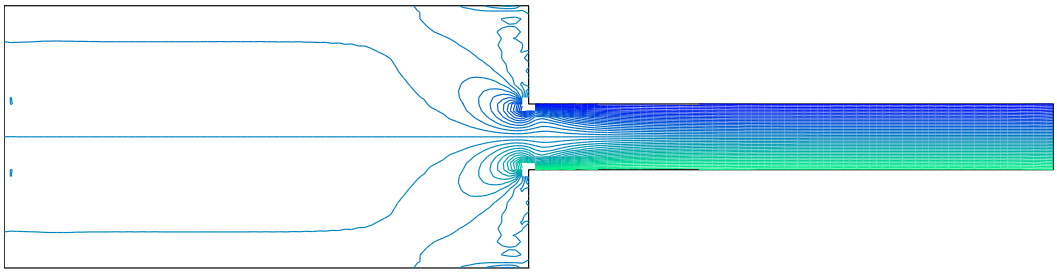
Figure 7.14 The 4:1 contraction flow of Oldroyd-B fluid. τ_{xy} contours of the flow for $We = \{0.4, 0.5\}$ with parameters given in Fig. 7.8.



(a) $We = 0.8$

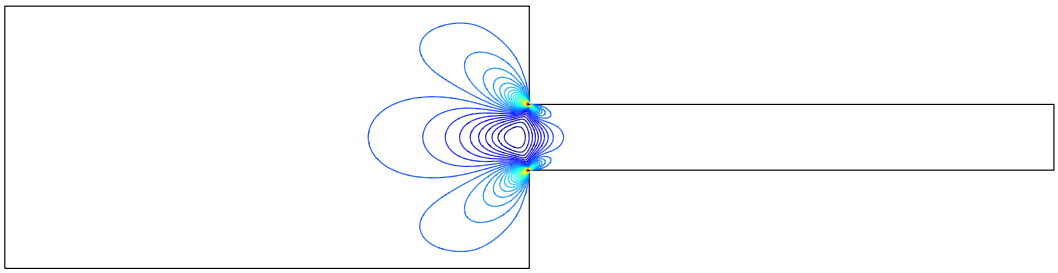


(b) $We = 0.9$

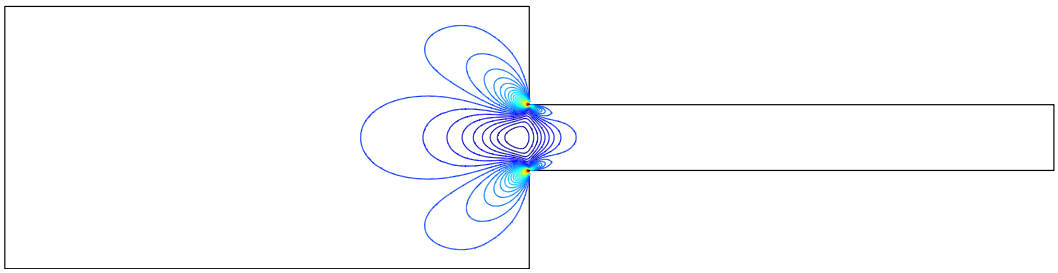


(c) $We = 1.0$

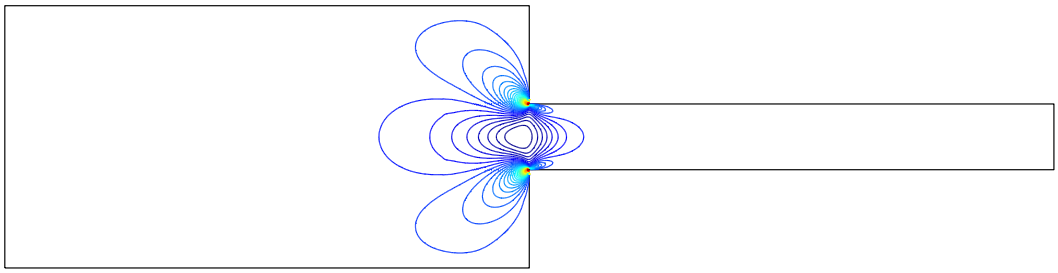
Figure 7.15 The 4:1 contraction flow of Oldroyd-B fluid. τ_{xy} contours of the flow for $We = \{0.8, 0.9, 1\}$ with parameters given in Fig. 7.9.



(a) $We = 0.1$

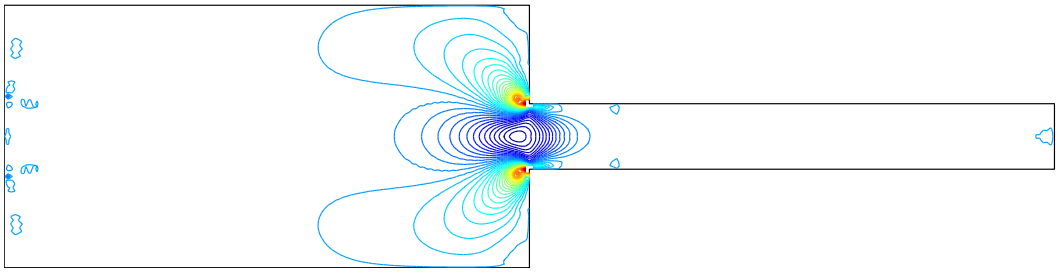


(b) $We = 0.2$

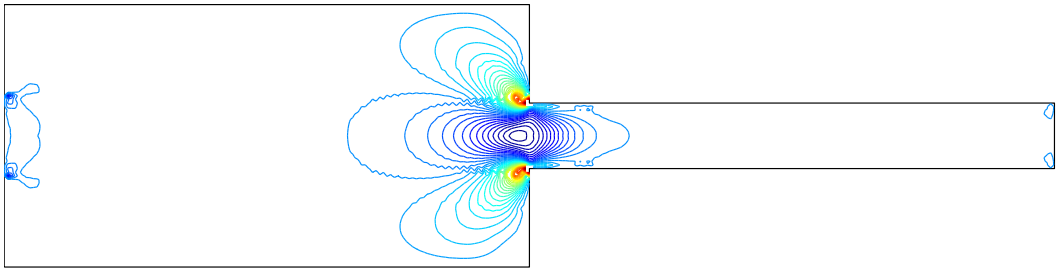


(c) $We = 0.3$

Figure 7.16 The 4:1 contraction flow of Oldroyd-B fluid. τ_{yy} contours of the flow for $We = \{0.1, 0.2, 0.3\}$ with parameters given in Fig. 7.7.

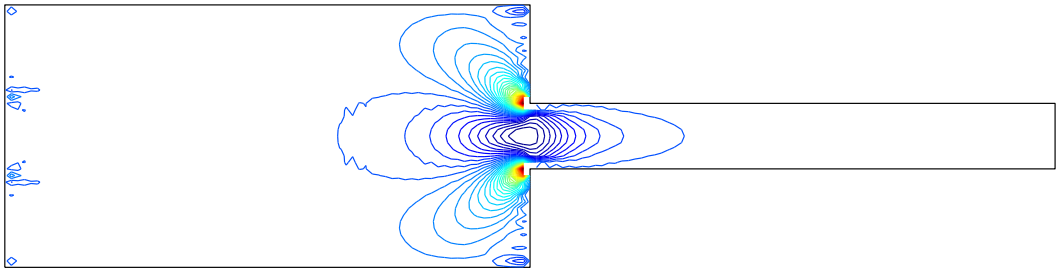


(a) $We = 0.4$

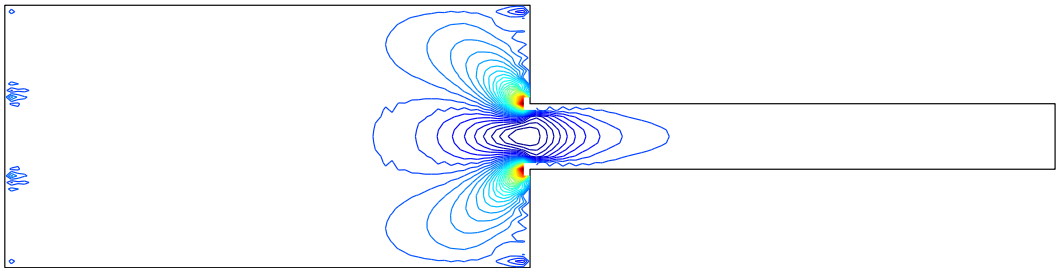


(b) $We = 0.5$

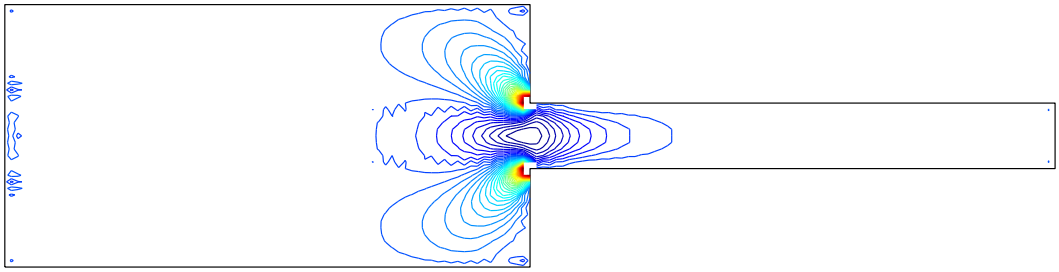
Figure 7.17 The 4:1 contraction flow of Oldroyd-B fluid. τ_{yy} contours of the flow for $We = \{0.4, 0.5\}$ with parameters given in Fig. 7.8.



(a) $We = 0.8$

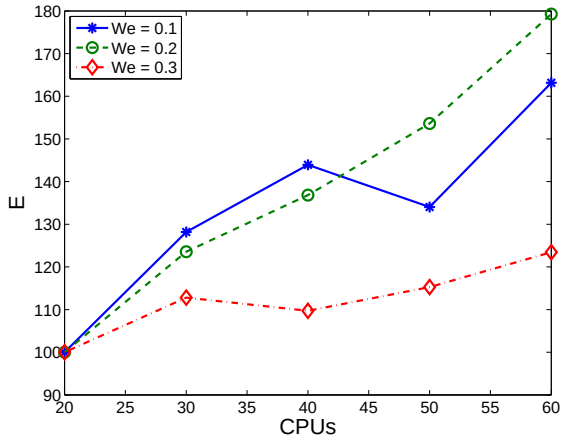


(b) $We = 0.9$

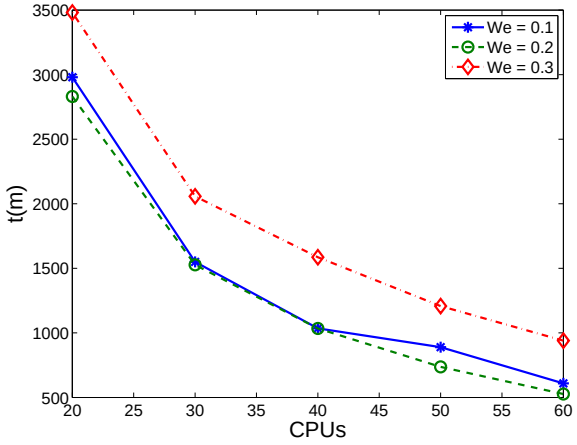


(c) $We = 1.0$

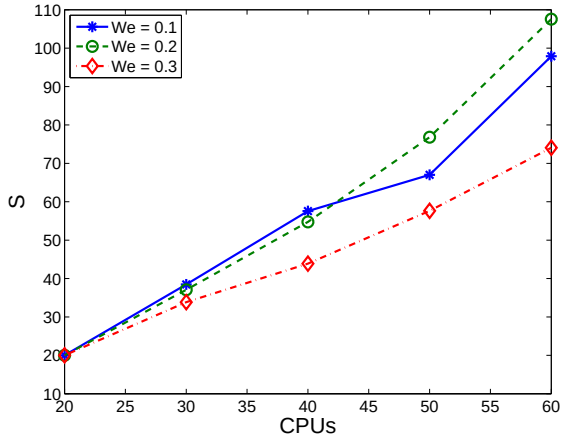
Figure 7.18 The 4:1 contraction flow of Oldroyd-B fluid. τ_{yy} contours of the flow for $We = \{0.8, 0.9, 1\}$ with parameters given in Fig. 7.9.



(a) Efficiency

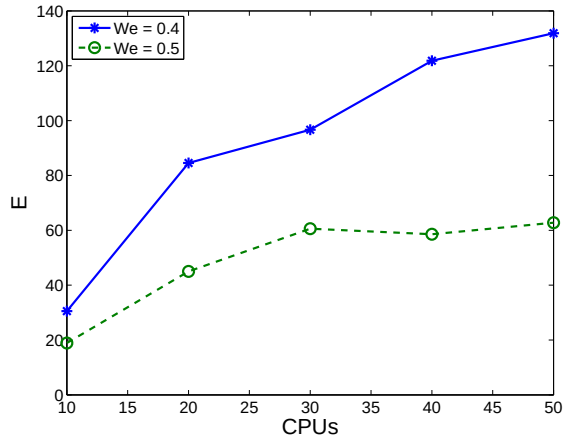


(b) Simulation time

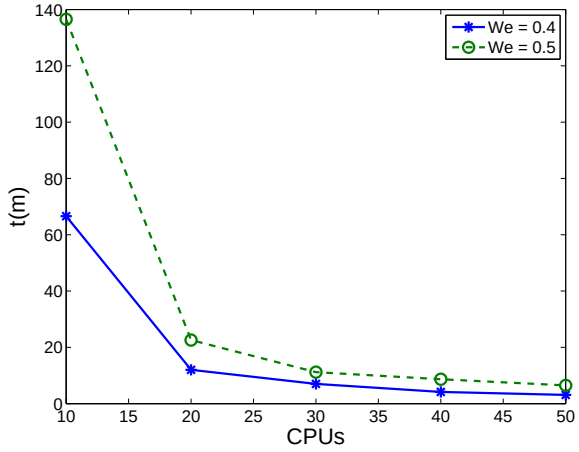


(c) speed-up

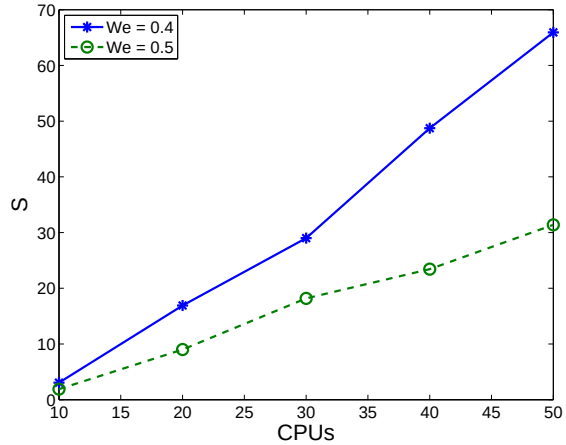
Figure 7.19 The 4:1 contraction flow of Oldroyd-B fluid. The efficiency (a), simulation time (b) and speed-up (c) of the present parallel method for $We = \{0.1, 0.2, 0.3\}$



(a) Efficiency



(b) Simulation time



(c) speed-up

Figure 7.20 The 4:1 contraction flow of Oldroyd-B fluid. The efficiency (a), simulation time (b) and speed-up (c) of the present parallel method for $We = \{0.4, 0.5\}$

7.6 Conclusions

The present parallel computation based on the domain decomposition technique and the local IRBF method is further demonstrated by solving the 4:1 planar contraction flow of Newtonian and Oldroyd-B fluids, using three grids with scales ranging from 12000 to 435000 DoF. Obtained results confirm that the method has high parallel efficiency, speed-up and scalability. Especially, super-linear speed-up is also observed for several numbers of CPUs. Furthermore, a relatively large number of DoF and high non-linearity of the problem (Oldroyd-B fluids) indicate that the present method is robust and suitable for solving large scale problems on a high performance computing system.

Chapter 8

Conclusion

This chapter concludes the thesis with a brief summary of the main contributions and achievements of present research project as well as some suggestions for future works and extensions.

8.1 Research achievements and contributions

With the aim to create efficient parallel numerical methods for analysis of large scale fluid flow problems, the objectives of the present research have been attained through a number of achievements and publications. Key contributions and achievements of the research are summarised as follows.

8.1.1 Research contributions

A successful development of parallel methods based on the DD and IRBF methods

The main contribution of this research project is the development of parallel methods based on DD and IRBF methods. Results show that the parallel methods have the following advantages: (i) high accuracy; (ii) high convergence rate; (iii) high efficiency with super-linear speed-up observed; (iv) high stability; (v) high scalability; and (vi) high adaptability.

A successful development of a new DTD algorithm used in parallel Domain Decomposition computations

Another major contribution is the development of a Bitmap DTD algorithm. The Bitmap DTD has the following advantages: (i) symmetric algorithm; (ii) low detection delay; (iii) low message complexity; (iv) small data overhead; and (v) graph free.

8.1.2 Research achievements

Details of achievements can be summarised as follows.

A successful development of the Bitmap DTD algorithm

This achievement has been presented in Chapter 3. A new DTD algorithm called Bitmap DTD has been introduced to be used with parallel DD methods. The algorithm is necessary for a parallel method in terms of the synchronisation and proper global termination of processes. The new algorithm makes use of a bitmap to carry information about the state of all processes in a distributed system. Judging by the behaviour, this Bitmap DTD is a member of Credit/Recovery DTD class. Nevertheless, there are some advantages of Bitmap DTD in comparison with a classic Credit/Recovery DTD as follows.

- the algorithm is symmetric, which means any process can detect the termination;
- there is no dedicated head process, which means no extra CPU required for the head node and the algorithm is graph free;
- the termination detection delay is low;
- the algorithm has optimal message complexity $O(M)$.

A successful development of a parallel algorithm based on overlapping DD techniques and CLIRBF

This achievement has been presented in Chapter 4. By this method, all subdomain processes run in parallel and the values on artificial boundaries (ABs) are updated after each iteration. The parallel method has the following advantages (i) high accuracy, which is provided by the IRBF; (ii) the IRBF method is not affected by the reduction in convergence rate caused by differentiation; (iii) the condition number of system matrices has been significantly reduced by using CLIRBF; and (iv) the throughput of the proposed method is high, especially in solving large scale problems (large degrees of freedom). The method is verified through several test problems including the benchmark lid-driven cavity flow problem and the numerical results obtained using coarse grids are in very good agreement with benchmark values.

A successful development of a parallel overlapping DD method coupled with a local IRBF Control Volume approach

This achievement has been presented in chapter 5. A parallel **overlapping** DD method coupled with a local IRBF Control Volume approach is presented. The method aims to solve fluid flow problems in complex domains. In fact, the method has been able to simulate several flows in rectangular and non-rectangular domains. For the case of rectangular lid-driven cavity flow, the method produces very accurate results in comparison with the spectral benchmark solutions by

Botella and Peyret (1998). For the triangular lid-driven cavity flow, the results are in great agreement with values obtained by Kohno and Bathe (2006). The speed-up of the method is very high. Furthermore, the method is stable and scalable by the fact that the speed-up grows steadily with respect to number of CPUs. Super-linear speed-up is observed for several Reynolds numbers.

A successful development of a parallel non-overlapping DD method coupled with 2D CLIRBF scheme

This achievement has been presented in Chapter 6. The **non-overlapping** DD method is investigated. A parallel method based on non-overlapping Dirichlet-Neumann DD method and 2D CLIRBF method is developed to solve the incompressible Navier-Stokes equation. Two considered benchmark problems are the lid-driven cavity flow and the natural convection in concentric annuli. The results show a great agreement between the parallel method and benchmark results. Indeed, for the lid-driven cavity flow problem, the efficiency has similar profile to the one obtained by parallel local IRBF-FVM presented in Chapter 5, where an optimal number of CPUs always exists for all Reynolds numbers. In some cases, the super-linear efficiency obtained by this method is even higher than one by the overlapping parallel method reported in Chapter 5.

A successful application of the present parallel method for some large scale problems

This achievement has been presented in Chapter 7. The present parallel computation based on the domain decomposition technique and the local IRBF method is further demonstrated by solving the 4:1 planar contraction flow of Newtonian and Oldroyd-B fluids, using three grids with scales ranging from 12000 to 435000 DoF. Obtained results confirm that the method has high parallel efficiency, speed-up and scalability. Especially, super-linear speed-up is also observed for several numbers of CPUs. Furthermore, a relatively large number of DoF and high non-linearity of the problem (Oldroyd-B fluids) indicate that the present method is robust and suitable for solving large scale problems on a HPC system.

8.2 Possible future works

Some outstanding issues for further investigation include

- Application of the present parallel methods in solving several 3-D large scale fluid flow problems as well as practical large scale ones. The practical problems of our interest include weather prediction, food security and other agricultural problems in the framework of the strategic research funding at the University of Southern Queensland.

- Flow analysis with more realistic models of complex fluids, e.g. kinetic suspesion models, using stochastic macro-micro multi-scale simulation methods.

- Coupling the present parallel methods with specific schemes such as the fractional step method and stabilization techniques DEVSS-G/DG (Kim et al., 2005b) to solve more efficiently the flow of viscoelastic fluids at high Weissenberg number.
- Further developing the present parallel methods using GPGPU HPC.

References

Agoshkov, V. I. and Lebedev, V. I. (1985). Poincaré-Steklov operators and methods of partition of the domain in variational problems. *Computational processes and systems*, 2:173–227.

Alves, M. A., Oliveira, P. J., and Pinho, F. T. (2003). Benchmark solutions for the flow of Oldroyd-B and PTT fluids in planar contractions. *Journal of Non-Newtonian Fluid Mechanics*, 110(1):45–75.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, page 483, New York, New York, USA. ACM Press.

Baaijens, F. P. T. (1998). Mixed finite element methods for viscoelastic flow analysis: a review. *Journal of Non-Newtonian Fluid Mechanics*, 73 (2-3):361–385.

Balls, G. T. and Colella, P. (2002). A finite difference domain decomposition method using local corrections for the solution of Poisson's equation. *Journal of Computational Physics*, 180(1):25–53.

Beatson, R. K., Light, W. A., and Billings, S. (2001). Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM Journal on Scientific Computing*, 22(5):1717–1740.

Becker, R., Hansbo, P., and Stenberg, R. (2003). A finite element method for domain decomposition with non-matching grids. *ESAIM: Mathematical Modelling and Numerical Analysis*, 37(2):209–225.

Bernstein, A. J. (1966). Analysis of programs for parallel processing. *IEEE Transactions on Electronic Computers*, EC-15(5):757–763.

Bjorstad, P. (1983). Fast numerical solution of the biharmonic Dirichlet problem on rectangles. *SIAM Journal on Numerical Analysis*, 20(1):59–71.

Botella, O. and Peyret, R. (1998). Benchmark spectral results on the lid-driven cavity flow. *Computers and Fluids*, 27:421–433.

Bourantas, G. C., Skouras, E. D., Loukopoulos, V. C., and Nikiforidis, G. C. (2010). Numerical solution of non-isothermal fluid flows using local radial basis functions (LRBF) interpolation and a velocity-correction method. *CMES Computer Modeling in Engineering and Sciences*, 64(2):187–212.

Brebbia, C. A., Telles, J. C. F., and Wrobel, L. C. (1984). *Boundary Element Techniques: theory and applications in engineering*. Springer.

Brooks, A. N. and Hughes, T. J. R. (1982). Streamline upwind/Petrov- Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32(1-3):199–259.

Brown, D. (2005). On approximate cardinal preconditioning methods for solving PDEs with radial basis functions. *Engineering Analysis with Boundary Elements*, 29(4):343–353.

Bui, T. T. (2000). A parallel, finite-volume algorithm for large-eddy simulation of turbulent flows. *Computers and Fluids*, 29(8):877–915.

Carew, E., Townsend, P., and Webster, M. (1993). A taylor-petrov-galerkin algorithm for viscoelastic flow. *Journal of Non-Newtonian Fluid Mechanics*, 50(2-3):253–287.

Chandhini, G. and Sanyasiraju, Y. (2007). Local RBF-FD solutions for steady convection-diffusion problems. *International journal for numerical methods in Engineering*, 72(3):352–378.

Chandrasekaran, S. and Venkatesan, S. (1990). A message-optimal algorithm for distributed termination detection. *Journal of parallel and distributed computing*, 8:245–252.

Chandy, K. M. and Misra, J. (1986). How processes learn. *Distributed Computing*, 1:40–52.

Cheng, A. H.-D., Golberg, M. A., Kansa, E. J., and Zammito, G. (2003). Exponential convergence and HâĂŘc multiquadric collocation method for partial differential equations. *Numerical Methods for Partial Differential Equations*, 19:571–594.

Chinchapatnam, P. P., Djidjeli, K., and Nair, P. B. (2007). Radial basis function meshless method for the steady incompressible Navier-Stokes equations. *International Journal of Computer Mathematics*, 84(10):1509–1521.

Conte, S. D. and Dames, R. T. (1958). An alternating direction method for solving the biharmonic equation. *Mathematical Tables and Other Aids to Computation*, 12 (63):198–205.

Covas, J. A., Agassant, J. F., Diogo, A. C., Vlachopoulos, J., and Walters, K. (1995). *Rheological fundamentals of polymer processing*. Kluwer Academic Publishers.

Cowsar, L. C., Mandel, J., and Wheeler, M. F. (1995). Balancing domain decomposition for mixed finite elements. *Mathematics of Computation*, 64(211):989–989.

Crochet, M. J., Davies, A. R., and Walters, K. (1984). *Numerical simulation of Non-Newtonian flow*, volume 1. Elsevier.

Dawson, C. N. and Dupont, T. F. (1994). Explicit/implicit, conservative domain decomposition procedures for parabolic problems based on block-centered finite differences. *SIAM Journal on Numerical Analysis*, 31(4):1045–1061.

Dijkstra, E. W., Feijen, W. H. J., and van Gasteren, A. J. M. (1983). Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters*, 16:217–219.

Dijkstra, E. W. and Scholten, C. S. (1980). Termination detection for diffusing computation. *Information Processing Letters*, 11:1–4.

Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., and White, A. (2003). *Sourcebook of parallel computing*. Morgan Kaufmann.

Emdadi, A., Kansa, E. J., Libre, N. A., Rahimian, M., and Shekarchi, M. (2008). Stable PDE solution methods for large multiquadric shape parameters. *CMES Computer Modeling in Engineering and Sciences*, 25 (1):23–41.

Eymard, R., Gallouet, T., and Herbin, R. (2000). Finite Volume Methods: Schemes and Analysis. *Handbook of numerical analysis*, 7:713–1018.

Fan, Y. R., Tanner, R. I., and Phan-Thien, N. (1999). Galerkin/least-square finite-element methods for steady viscoelastic flows. *Journal of Non-Newtonian Fluid Mechanics*, 84:233–256.

Fasshauer, G. E. (1999). Solving differential equations with radial basis functions: multilevel methods and smoothing. *Advances in Comp. Math*, 11:139–159.

Fattal, R. and Kupferman, R. (2004). Constitutive laws for the matrix-logarithm of the conformation tensor. *Journal of Non-Newtonian Fluid Mechanics*, 123(2-3):281–285.

Fedoseyev, A. I., Friedman, M. J., and Kansa, E. J. (2002). Improved multiquadric method for elliptic partial differential equations via PDE collocation on the boundary. *Computer and Mathematics with Application*, pages 439–455.

Ferreira, V. G., Kurokawa, F. A., Queiroz, R. A. B., Kaibara, M. K., Oishi, C. M., Cuminato, J. A., Castelo, A., Tome, M. F., and McKee, S. (2009). Assessment of a high-order finite difference upwind scheme for the simulation of convection-diffusion problems. *International Journal for Numerical Methods in Fluids*, 60:1–26.

Flynn, M. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909.

Francez, N. (1980). Distributed Termination. *ACM Transaction on Programming Languages and Systems*, 2:42–55.

Franke, R. (1982). Scattered data interpolation: tests of some methods. *Mathematics of Computation*, 38 (157).

Gander, M. J. and Tu, X. (2014). On the origins of iterative substructuring methods. *Domain Decomposition Methods in Science and Engineering XXI*, pages 597–605.

Ghia, U., Ghia, K. N., and Shin, C. T. (1982). High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411.

Golberg, M. A., Chen, C. S., and Karur, S. R. (1996). Improved multiquadric approximation for partial differential equations. *Engineering Analysis with Boundary Element*.

Gupta, M. M. and Manohar, R. P. (1979). Direct solution of the biharmonic equation using noncoupled approach. *Journal of Computational Physics*, 33:236–248.

Gustafson, J. L. (1988). Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation (second Edition)*, volume 842. Prentice Hall.

Haynsworth, E. V. (1968). On the Schur complement. Technical report, DTIC Document.

Hoang-Trieu, T.-T., Mai-Duy, N., and Tran-Cong, T. (2012). Several compact local stencils based on Integrated RBFs for fourth-order ODEs and PDEs. *CMES Computer Modeling in Engineering and Sciences*, 84(2):171–203.

Hou, T. Y. and Wu, X.-H. (1997). A multiscale finite element method for elliptic problems in composite materials and porous media. *Journal of Computational Physics*, 134(1):169–189.

Huang, S.-T. (1988). A fully distributed termination detection scheme. *Information Processing Letters*, 29(1):13–18.

Huang, S.-T. (1989). Detecting termination of distributed computations by external agents. In *[1989] Proceedings. The 9th International Conference on Distributed Computing Systems*, pages 79–84. IEEE Comput. Soc. Press.

Hughes, T. J. R. (1987a). Recent progress in the development and understanding of SUPG methods with special reference to the compressible Euler and Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 7 (11):1261–1275.

Hughes, T. J. R. (1987b). *The finite element method: linear static and dynamic finite element analysis*. Prentice-Hall.

Huilgol, R. R. and Phan-Thien, N. (1997). *Fluid mechanics of viscoelasticity (Rheology series 6)*. Elsevier.

Hussain, S. H. and Hussein, A. K. (2010). Numerical investigation of natural convection phenomena in a uniformly heated circular cylinder immersed in square enclosure filled with air at different vertical locations. *International Communications in Heat and Mass Transfer*, 37(8):1115–1126.

Ilnytskyi, J. and Wilson, M. (2002). A domain decomposition molecular dynamics program for the simulation of flexible molecules of spherically-symmetrical and nonspherical sites. II.ÂăExtension to NVT and NPT ensembles. *Computer Physics Communications*, 148(1):43–58.

Ingber, M., Chen, C., and Tanski, J. (2004). A mesh free approach using radial basis functions and parallel domain decomposition for solving three-dimensional diffusion equations. *International Journal for Numerical Methods in Engineering*, 60:2183–2201.

Jabbarzadeh, A., Atkinson, J., and Tanner, R. (1997). Parallel simulation of shear flow of polymers between structured walls by molecular dynamics simulation on PVM. *Computer Physics Communications*, 107(1-3):123–136.

Jabbarzadeh, A., Atkinson, J., and Tanner, R. (2003). A parallel algorithm for molecular dynamics simulation of branched molecules. *Computer Physics Communications*, 150(2):65–84.

Johan, Z., Hughes, T. J., Mathur, K. K., and Johnsson, S. (1992). A data parallel finite element method for computational fluid dynamics on the Connection Machine system. *Computer Methods in Applied Mechanics and Engineering*, 99(1):113–134.

Kansa, E. J. (1990a). Multiquadrics - A scattered data approximation scheme with applications to computational fluid-dynamics - I Surface approximations and partial derivative estimates. *Computers and Mathematics with Applications*, 19(8-9):127–145.

Kansa, E. J. (1990b). Multiquadrics - A scattered data approximation scheme with applications to computational fluid-dynamics - II solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers and Mathematics with Applications*, 19(8-9):147–161.

Kim, B., Lee, D., Ha, M., and Yoon, H. (2008). A numerical study of natural convection in a square enclosure with a circular cylinder at different vertical locations. *International Journal of Heat and Mass Transfer*, 51(7-8):1888–1906.

Kim, J. M., Kim, C., Chung, C., Ahn, K. H., and Lee, S. J. (2005a). Negative wake generation of FENE-CR fluids in uniform and Poiseuille flows past a cylinder. *Rheologica Acta*, 44:600–610.

Kim, J. M., Kim, C., Kim, J. H., Chung, C., Ahn, K. H., and Lee, S. J. (2005b). High-resolution finite element simulation of 4:1 planar contraction flow of viscoelastic fluid. *Journal of Non-Newtonian Fluid Mechanics*, 129(1):23–37.

Kim, Y., Kim, D. W., Jun, S., and Lee, J. H. (2007). Meshfree point collocation method for the stream-vorticity formulation of 2D incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 196(33-34):3095–3109.

Kohno, H. and Bathe, K.-J. (2006). A flow-condition-based interpolation finite element procedure for triangular grids. *International Journal for Numerical Methods in Fluids*, 51(6):673–699.

Koradi, R., Billeter, M., and Güntert, P. (2000). Point-centered domain decomposition for parallel molecular dynamics simulation. *Computer Physics Communications*, 124(2-3):139–147.

Kosec, G. and Sarler, B. (2008). Local RBF collocation method for Darcy flow. *CMES Computer Modeling in Engineering and Sciences*, 25:197–208.

Le-Cao, K., Mai-Duy, N., and Tran-Cong, T. (2009). An effective intergrated-RBFN Cartesian-grid discretization for the streamfunction-vorticity-temperature formulation in nonrectangular domains. *Numerical Heat Transfer, Part B: Fundamentals*, 55(6):480–502.

Lee, C., Liu, X., and Fan, S. (2003). Local multiquadric approximation for solving boundary value problems. *Computational Mechanics*, 30:396–409.

Lele, S. K. (1992). Compact finite difference schemes with spectral-like resolution. *Journal of Computational Physics*, 103:16–42.

Li, J. and Hon, Y. C. (2004). Domain decomposition for radial basis meshless methods. *Numerical Methods for Partial Differential Equations*, 20(3):450–462.

Li, M., Tang, T., and Fornberg, B. (1995). A compact fourth-order finite difference scheme for the steady incompressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 20:1137–1151.

Lin, H. and Atluri, S. (2001). The meshless local Petrov-Galerkin (MLPG) method for solving incompressible Navier-Stokes equations. *CMES Computer Modeling in Engineering and Sciences*, 2(2):117–142.

Ling, L., Opfer, R., and Schaback, R. (2006). Results on meshless collocation techniques. *Engineering Analysis with Boundary Elements*, 30(4):247–253.

Mai-Duy, N., Le-Cao, K., and Tran-Cong, T. (2008). A Cartesian grid technique based on one-dimensional integrated radial basis function networks for natural convection in concentric annuli. *International Journal for Numerical Methods in Fluids*, 57:1709–1730.

Mai-Duy, N. and Tanner, R. I. (2007). A collocation method based on one-dimensional RBF interpolation scheme for solving PDEs. *International Journal of Numerical Methods for Heat and Fluid Flow*, 17(2):165–186.

Mai-Duy, N. and Tran-Cong, T. (2001). Numerical solution of differential equations using multiquadric radial basis function networks. *Neural Networks*, 14:185–199.

Mai-Duy, N. and Tran-Cong, T. (2003). Approximation of function and its derivatives using radial basis function networks. *Applied mathematical modeling*, 27:197–220.

Mai-Duy, N. and Tran-Cong, T. (2004). Boundary integral-based domain decomposition technique for solution of Navier Stokes equations. *CMES Computer Modeling in Engineering and Sciences*, 6(1):59–75.

Mai-Duy, N. and Tran-Cong, T. (2010). A numerical study of 2D integrated RBFNs incorporating Cartesian grids for solving 2D elliptic differential problems. *Numerical methods for Partial Differential Equation*, 26:1443–1462.

Mai-Duy, N., Tran-Cong, T., and Phan-Thien, N. (2011). A numerical technique based on integrated RBFs for the system evolution in molecular dynamics. *CMES Computer Modeling in Engineering and Sciences*, 79:223–236.

Marchal, J. and Crochet, M. (1986). Hermitian finite elements for calculating viscoelastic flow. *Journal of Non-Newtonian Fluid Mechanics*, 20:187–207.

Marchal, J. and Crochet, M. (1987). A new mixed finite element for calculating viscoelastic flow. *Journal of Non-Newtonian Fluid Mechanics*, 26(1):77–114.

Marini, L. D. and Quarteroni, A. (1989). A relaxation procedure for domain decomposition methods using finite elements. *Numerische Mathematik*, 55(5):575–598.

Marshall, J., Adcroft, A., Hill, C., Perelman, L., and Heisey, C. (1997). A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research*, 102(C3):5753.

Matallah, H., Townsend, P., and Webster, M. (1998). Recovery and stress-splitting schemes for viscoelastic flows. *Journal of Non-Newtonian Fluid Mechanics*, 75(2-3):139–166.

MATLAB (2012). *Parallel Computing Toolbox User's Guide R2012b*. The MathWorks Inc., Natick, Massachusetts, United States, 6.1 edition.

Matocha, J. and Camp, T. (1998). A taxonomy of distributed termination detection algorithms. *The Journal of Systems and Software*, 43:207–221.

Mattern, F. (1987). Algorithms for distributed termination detection. *Distributed Computing*, 2(3):161–175.

Mattern, F. (1989). Global quiescence detection based on credit distribution and recovery. *Information Processing Letters*, pages 195–200.

Mitchell, A. R. and Griffiths, D. F. (1980). *The finite difference method in partial differential equations*. John Wiley, New York.

Moukalled, F. and Acharya, S. (1996). Natural convection in the annulus between concentric horizontal circular and square cylinders. *Journal of Thermophysics and Heat Transfer*, 10(3):524–531.

Ngo-Cong, D., Mai-Duy, N., Karunasena, W., and Tran-Cong, T. (2012). Local moving least square - One-Dimensional IRBFN technique: Part I - Natural convection flows in concentric and eccentric annuli. *CMES Computer Modeling in Engineering and Sciences*, 83(3):275–310.

Pastor, M., Peraire, J., and Zienkiewicz, O. C. (1991). Adaptive remeshing for shear band localization problems. *Archive of Applied Mechanics*, 61 (1):30–39.

Patankar, S. V. (1980). *Numerical heat transfer and fluid flow*. CRC Press.

Pham-Sy, N., Tran, C.-D., Hoang-Trieu, T.-T., Mai-Duy, N., and Tran-Cong, T. (2013). Compact local IRBF and domain decomposition method for solving PDEs using a distributed termination detection based parallel algorithm. *CMES Computer Modeling in Engineering and Sciences*, 92(1):1–31.

Pham-Sy, N., Tran, C.-D., Mai-Duy, N., and Tran-Cong, T. (2014). Parallel control-volume Method Based on compact local Integrated RBFs for the solution of fluid flow problems. *CMES Computer Modeling in Engineering and Sciences*, 100(5):363–397.

Pham-Sy, N., Tran, C. D., Mai-Duy, N., and Tran-Cong, T. (2015a). A symmetric Bitmap Distributed Termination Detection approach for domain decomposition based parallel algorithms. *CMES Computer Modeling in Engineering and Sciences*, Submitted.

Pham-Sy, N., Tran-Canh, D., Mai-Duy, N., and Tran-Cong, T. (2015b). A parallel non-overlapping domain decomposition algorithm using integrated radial basis function for solving Navier-Stokes equations. *CMES Computer Modeling in Engineering and Sciences*, Submitted.

Phan-Thien, N. (1978). A nonlinear network viscoelastic model. *Journal of Rheology*, 22(3):259.

Phan-Thien, N. (1984). Squeezing a viscoelastic liquid from a wedge: an exact solution. *Journal of Non-Newtonian Fluid Mechanics*, 16(3):329–345.

Phan-Thien, N. and Tanner, R. I. (1977). A new constitutive equation derived from network theory. *Journal of Non-Newtonian Fluid Mechanics*, 2(4):353–365.

Phillips, T. and Williams, A. (1999). Viscoelastic flow through a planar contraction using a semi-Lagrangian finite volume method. *Journal of Non-Newtonian Fluid Mechanics*, 87(2-3):215–246.

Pilitsis, S. and Beris, A. N. (1989). Calculations of steady-state viscoelastic flow in an undulating tube. *Journal of Non-Newtonian Fluid Mechanics*, 31 (3):231–287.

Quarteroni, A. and Valli, A. (1999). *Domain decomposition methods for partial differential equations*. Oxford University Press.

Rannacher, R. (1999). *Finite Element Methods for the Incompressible Navier-Stokes Equations*. University of Heidelberg, INF 293/294, D-69120 Heidelberg, Germany.

Raynal, M. (2013). *Distributed algorithms for Message-Passing systems*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Reddy, J. N. (2005). *An introduction to the finite element method*. McGraw-Hill, 3 edition.

Reddy, J. N. and Gartling, D. K. (1994). *The einite element method in heat transfer and fluid dynamics*. CRC Press.

Sato, T. and Richardson, S. M. (1994). Explicit numerical simulation of time-dependent viscoelastic flow problems by a finite element/finite volume method. *Journal of Non-Newtonian Fluid Mechanics*, 51(3):249–275.

Schwarz, H. A. (1869). Ueber einige Abbildungsaufgaben. *Journal für die reine und angewandte Mathematik*, 70:105–120.

Schwarz, H. A. (1972). *Gesammelte mathematische abhandlungen*, volume 2. American Mathematical Soc.

Sharan, M., Kansa, E. J., and Gupta, S. (1997). Application of the multiquadric method for numerical solution of elliptic partial differential equations. *Applied Mathematics and Computation*, 84(2-3):275–302.

Shirazaki, M. and Yagawa, G. (1999). Large-scale parallel flow analysis based on free mesh method: a virtually meshless method. *Computer Methods in Applied Mechanics and Engineering*, 174(3-4):419–431.

Shu, C., Ding, H., and Yeo, K. (2005). Computation of incompressible Navier-Stokes equations by local RBF-based differential quadrature method. *CMES Computer Modeling in Engineering and Sciences*, 7(2):195–206.

Shu, C., Ding, H., and Yeo, K. S. (2003). Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 192:941–954.

Shu, C. and Zhu, Y. D. (2002). Efficient computation of natural convection in a concentric annulus between an outer square cylinder and an inner circular cylinder. *International Journal for Numerical Methods in Fluids*, 38:429–445.

Singh, I. V. and Jain, P. K. (2005). Parallel meshless EFG solution for fluid flow problems. *Numerical Heat Transfer, Part B: Fundamentals*, 48(1):45–66.

Smith, B. F., Bjorstad, P. E., and Gropp, W. D. (1996). *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press.

Smith, G. D. (1978). *Numerical solution of partial differential equations: finite difference methods*. Oxford: Clarendon Press.

Sun, J., Smith, M. D., Armstrong, R. C., and Brown, R. A. (1999). Finite element method for viscoelastic flows based on the discrete adaptive viscoelastic stress splitting and the discontinuous Galerkin method: DAVSS-G/DG. *Journal of Non-Newtonian Fluid Mechanics*, 86:281–307.

Tanner, R. I. (2000). *Engineering Rheology*. Oxford University Press, 2 edition.

Tanner, R. I. and Xue, S.-C. (2002). Computing transient flows with high elasticity. *Korea-Australia Rheology Journal*, 14 (4):143–159.

Tezduyar, T. E., Aliabadi, S. K., Behr, M., and Mittal, S. (1994). Massively parallel finite element simulation of compressible and incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 119(1):157–177.

Thai-Quang, N., Le-Cao, K., Mai-Duy, N., and Tran-Cong, T. (2012). A high-order Compact Local Integrated-RBF scheme for steady-state incompressible viscous flows in the primitive variables. *CMES Computer Modeling in Engineering and Sciences*, 84(6):528–557.

Tran, C.-D., Phillips, D. G., and Tran-Cong, T. (2009). Computation of dilute polymer solution flows using BCF-RBFN based method and domain decomposition technique. *Korea-Australia Rheology Journal*, 21(1):1–12.

Tran-Canh, D. and Tran-Cong, T. (2004). Element-free simulation of dilute polymeric flows using Brownian Configuration Fields. *Korea-Australia Rheology Journal*, 13(1):1–15.

Tseng, T. C. (1995). Detecting termination by weight-throwing in a faulty distributed system. *Journal of Parallel and Distributed Computing*, 25(1):7–15.

Venkatesan, S. (1989). Reliable protocols for distributed termination detection. *Reliability, IEEE Transactions on*, 38(1):103–110.

Yoo, J. Y. and Na, Y. (1991). A numerical study of the planar contraction flow of a viscoelastic fluid using the SIMPLER algorithm. *Journal of Non-Newtonian Fluid Mechanics*, 39(1):89–106.

Yurun, F. and Crochet, M. J. (1995). High-order finite element methods for steady viscoelastic flows. *Journal of Non-Newtonian Fluid Mechanics*, 27:283–311.

Zerroukat, M., Power, H., and Chen, C. S. (1998). A numerical method for heat transfer problems using collocation and radial basis functions. *International Journal for Numerical Methods in Engineering*, 42:1263–1278.

Zhou, X., Hona, Y. C., and Li, J. (2003). Overlapping domain decomposition method by radial basis functions. *Applied Numerical Mathematics*, 44:241–255.

Appendix A

Some Radial Basis Functions

Following is a small list of several common radial basis functions.

• Gaussian-RBF

$$h^j(r, b^j) = h^j(\|\mathbf{x} - \mathbf{c}^j\|, b^j) = \exp\left(-\frac{r^2}{(b^j)^2}\right), \tag{A.1}$$

• Multiquadrics RBF (MQ-RBF)

$$h^j(r, b^j) = h^j(\|\mathbf{x} - \mathbf{c}^j\|, b^j) = \sqrt{r^2 + (b^j)^2}, \tag{A.2}$$

• Inverse multiquadrics RBF (IMQ-RBF)

$$h^j(r, b^j) = h^j(\|\mathbf{x} - \mathbf{c}^j\|, b^j) = \frac{1}{\sqrt{r^2 + (b^j)^2}}, \tag{A.3}$$

• Thin plate splines RBF (TPS-RBF)

$$h^j(r) = h^j(\|\mathbf{x} - \mathbf{c}^j\|) = r^{2s}\log(r), \qquad s = 1, 2, ..., \tag{A.4}$$

• Compact support RBF (CS-RBF)

$$h^j(r) = h^j(\|\mathbf{x} - \mathbf{c}^j\|) = (1 - r)_+^l p(r), \qquad l = 1, 2, ..., \tag{A.5}$$

where s is the order of TPS-RBF; $p(r)$ is a polynomial of Wendland CS-RBF (Wendland, 1995,1998). While the MQ-RBF, TPS-RBF and CS-RBF exhibit global response, i.e. they increase monotonically with increasing distance from the centre, the Gaussian-RBF and IMQ-RBF have local response (localized function) (Haykin, 1999; Beatson and Light, 1997). RBFs are multivariate as function of $\mathbf{x} \in \mathbf{R}^d$, but univariate as function of r, which should be a tremendous computational advantage if the space dimension is large.

Appendix B

Program code of a parallel DD-IRBF algorithm for solving 2D problems with a known analytic solution (Chapter 4, section 4.5.2) using 4 sub-domains

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File                  parDDIRBF2D.m
% Author                Nam.Pham-Sy@usq.edu.au
% Created               02/04/2014
% iRBF                  2D-iRBF, Compac local scheme
% DDM                   overlapping
% Version               0.1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem               Analytical
% Domain                Square
% GE            f = 4*(1-pi^2)*sin(2*pi*x').*sinh(2*y')
%               +16*(1-pi^2)*cosh(4*x').*cos(4*pi*y');
% Solution      u = sin(2*pi*x').*sinh(2*y')
%
             +cosh(4*x').*cos(4*pi*y');
% BC                    Dirichlet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description
% The code solves an analytical problem in parallel
% using 4 CPUs. The original domain is divided into 4
% sub-domains with zero artificial boundary condition.
% The problem in each sub-domain is solved separately
% using routine CLiRBFSPMDTask. This routine is a
% general solver of a PDE using compact local IRBF
% scheme as its approximant. The value of artifical
% boundary from each sub-domain is updated by exchange
% data with its neighbour sub-domains. The termination
% of parallel algorithm is backed by the Bitmap DTD
```

```
28 % algorithm. After the computation is finished the
29 % results is collected from sub-domains to a variable
30 % called uRes.
31 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32 close all;
33 clear all;
34 format short e;
35 tic
36
37 % function aliases
38 f=@(x,y) 4*(1-pi^2)*sin(2*pi*x').*sinh(2*y')+16*(1-pi
      ^2)*cosh(4*x').*cos(4*pi*y');
39 u=@(x,y) sin(2*pi*x').*sinh(2*y')+cosh(4*x').*cos(4*pi*
      y');
40 xA = -0.4; xB = 0.6; yA = -0.6; yB = 0.4; beta = 5;
41 owp = 10;              % overlapping percentage
42 ddTol = 1.e-2;         % domain decomp. tolerance
43 nSub = 51;             % grid-size of sub-domain
44 DD = [2 2];            % DD configuration
45 nRowDD = DD(1);
46 nColDD = DD(2);
47 nSD = nRowDD*nColDD;      % number of sub-domain
48 ow = round(nSub*owp/(100*2))*2+1; % overlapping width
49 nx = nSub*DD(2)-(DD(2)-1)*ow;
50 ny = nSub*DD(1)-(DD(1)-1)*ow;
51 x = linspace(xA, xB, nx);
52 y = linspace(yA, yB, ny);
53
54 % Open parallel pool
55 matlabpool open 4;
56
57 % Enter parallel pool
58 spmd (nSD)
59     iRow = rem(labindex-1,nRowDD)+1;
60     iCol = floor((labindex-1)/nRowDD)+1;
61     leftNb = []; leftAB = [];
62     rightNb = []; rightAB = [];
63     topNb = []; topAB = [];
64     botNb = []; botAB = [];
65     if(iCol-1 > 0)
66         leftNb = labindex-nRowDD;      % Nb = neighbor
67         leftAB = zeros(nSub,1);
68     end
69     if(iCol+1 <= nColDD)
70         rightNb = labindex+nRowDD;
71         rightAB = zeros(nSub,1);
72     end
73     if(iRow-1 > 0)
74         topNb = labindex-1;
```

```
75          topAB = zeros(1,nSub);
76      end
77      if(iRow+1 <= nRowDD)
78          botNb = labindex+1;
79          botAB = zeros(1,nSub);
80      end
81
82      xOffsetStart = (nSub-ow)*(iCol-1)+1;
83      xOffsetEnd = xOffsetStart+nSub-1;
84      yOffsetStart = (nSub-ow)*(iRow-1)+1;
85      yOffsetEnd = yOffsetStart+nSub-1;
86
87
88      % DTD init
89      maxSTE = nSDM + 1;
90      nByte = ceil(nSDM/32);
91      bitmap = zeros(nByte,1);
92      stopCode = zeros(nByte,1);
93      if(nByte>1) stopCode(1:nByte-1) = 2^32-1; end
94      stopCode(nByte) = 2^(nSDM - (nByte-1)*32) - 1;
95      readyCode = zeros(nByte,1);
96      myNByte = ceil(labindex/32);
97      readyCode(myNByte) = 2^((labindex-(myNByte-1)*32)
           -1);
98      STE = -1;  % Step To End
99      leftDDCM = -inf; rightDDCM = -inf;
100                 topDDCM = -inf; botDDCM = -inf;
101
102     iDD = 0; % number of iterations
103     while(1)
104         iDD = iDD+1;
105         % call CLiRBFSPMDTask to solve PDE
106         myResult = CLiRBFSPMDTask(f,u,x(xOffsetStart),x
               (xOffsetEnd),...
107             y(yOffsetStart),y(yOffsetEnd),nSub,nSub,
                   beta,leftAB,rightAB,topABbotAB);
108         myResult = reshape(myResult,nSub,nSub);
109
110         % communication
111         minPosSTE = maxSTE;
112         % my left neighbor
113         if(isempty(leftNb)==0)
114             leftResult = labSendReceive(leftNb,leftNb,
                   ...
115                 [reshape(myResult(:,1:ow),ow*nSub,1);
                       bitmap;STE]);
116             lRecvBitmap = lRes(end-nByte:end-1);
117             lRecvSTE = lRes(end);
118             leftResult(end-nByte:end) = [];
```

```
119             leftResult = reshape(leftResult,nSub,ow);
120
121             bitmap = bitor(bitmap,lRecvBitmap);
122             minPosSTE(lRecvSTE > 0 && lRecvSTE <
                   minPosSTE)  = lRecvSTE;
123             a = norm(myResult(:,1:ow)-leftResult);
124             b = norm(myResult(:,1:ow));
125             c = norm(leftResult);
126             leftCM = a*(b+c)/(2*b*c);
127             leftAB = leftResult(:,1);
128         end
129         % my right neighbor
130         if(isempty(rightNb)==0)
131             rightResult = labSendReceive(rightNb,
                   rightNb,...
132                 [reshape(myResult(:,end-ow+1:end),ow*
                       nSub,1);bitmap;STE]);
133             rRecvBitmap = rRes(end-nByte:end-1);
134             rRecvSTE = rRes(end);
135             rightResult(end-nByte:end) = [];
136             rightResult = reshape(rightResult,nSub,ow);
137
138             bitmap = bitor(bitmap,rRecvBitmap);
139             minPosSTE(rRecvSTE > 0 && rRecvSTE <
                   minPosSTE) = rRecvSTE;
140             a = norm(myResult(:,end-ow+1:end)-
                   rightResult);
141             b = norm(myResult(:,end-ow+1:end));
142             c = norm(rightResult);
143             rightCM = a*(b+c)/(2*b*c);
144             rightAB = rightResult(:, end);
145         end
146         % my top neighbor
147         if(isempty(topNb)==0)
148             topResult = labSendReceive(topNb,topNb,...
149                 [reshape(myResult(1:ow, :),ow*nSub,1);
                       bitmap;STE]);
150             tRecvBitmap = tRes(end-nByte:end-1);
151             tRecvSTE = tRes(end);
152             topResult(end-nByte:end) = [];
153             topResult = reshape(topResult,ow,nSub);
154
155             bitmap = bitor(bitmap,tRecvBitmap);
156             minPosSTE(tRecvSTE > 0 && tRecvSTE <
                   minPosSTE)  = tRecvSTE;
157             a = norm(myResult(1:ow, :) - topResult);
158             b = norm(myResult(1:ow, :));
159             c = norm(topResult);
160             topCM = a*(b+c)/(2*b*c);
```

```
161             topAB = topResult(1, :);
162         else
163             topCM = 0;
164             topAB = [];
165             topReady = 2;
166         end
167         % my bot neighbor
168         if(isempty(botNb)==0)
169             botResult = labSendReceive(botNb,botNb,...
170                 [reshape(myResult(end-ow+1:end, :),ow*
                        nSub,1);bitmap;STE]);
171             bRecvBitmap = bRes(end-nByte:end-1);
172             bRecvSTE = bRes(end);
173             botResult(end-nByte:end) = [];
174             botResult = reshape(botResult,ow,nSub);
175 
176             bitmap = bitor(bitmap,bRecvBitmap);
177             minPosSTE(bRecvSTE > 0 && bRecvSTE <
                    minPosSTE)  = bRecvSTE;
178             a = norm(myResult(end-ow+1:end, :) -
                    botResult);
179             b = norm(myResult(end-ow+1:end, :));
180             c = norm(botResult);
181             botCM = a*(b+c)/(2*b*c);
182             botAB = botResult(end, :);
183         end
184 
185         % Termination Dectection
186         if(STE > 0) % in synchronised termination phase
187             if(minPosSTE < STE) % neighbor's STE is
                    smaller
188                 STE = minPosSTE - 1;
189                 fprintf('i = %d, Received lower
                        minPosSTE = %d. STE = %d\n',iDD,
                        minPosSTE,STE);
190             else
191                 STE = STE - 1;
192                 fprintf('i = %d, STE = %d\n',iDD,STE);
193             end
194             if(STE == 0)
195                 break; % break while (ddm loop)
196             end
197         elseif(STE < 0) % in termination detection
                phase
198             if(minPosSTE < maxSTE)% some of neighbor
                    has STE>0
199                 STE = minPosSTE - 1;
200                 fprintf('i = %d, Received Termination
                        Signal. STE = %d\n',iDD,STE);
```

```
201             else
202                 if(sum(bitand(readyCode,bitmap))==0 &&
                       leftDDCM < ddTol && rightDDCM <
                       ddTol ...
203                     && topDDCM < ddTol && botDDCM <
                           ddTol && CM < CMTol)
204                     bitmap = bitor(readyCode,bitmap);
205                     fprintf('i = %d, Ready to terminate
                           . bitmap = %s\n',iDD,dec2bin(
                           bitmap));
206                 end
207                 if(bitmap == stopCode)
208                     STE = ddRow + ddCol - 2;
209                     fprintf('i = %d, Termination
                           detected. STE = %d\n',iDD,STE);
210                 end
211             end
212             if(STE == 0)
213                 break; % break while (ddm loop)
214             end
215         end
216     end
217     display(iDD);
218 end
219
220 % Collecting results
221 uRes = [];
222 for i = 1:nRowDD
223     iRow = [];
224     for j = 1:nColDD
225         iSD = i + (j-1)*nRowDD;
226         iResult = myResult{iSD};
227         if(j==nColDD)
228             iRow = [iRow, iResult];
229         else
230             iRow = [iRow, iResult(:,1:end-ow)];
231         end
232     end
233     if(i == nRowDD)
234         uRes = [uRes; iRow];
235     else
236         uRes = [uRes; iRow(1:end-ow,:)];
237     end
238 end
239 elapsedTime = toc;
240 [xm, ym] = meshgrid(x, y);
241 nodes = [xm(:),ym(:)]';
242 uNodes = reshape(u(nodes(1, :), nodes(2, :)),ny,nx);
243 Ne = norm(uNodes-uRes)/norm(uNodes);
```

```
244
245 % save data
246 save('result.dat', 'xm', 'ym', 'ny', 'nx', 'uNodes', '
       uRes', 'elapsedTime');
```