2008

Development of a multiblock solver utilizing the lattice Boltzmann and traditional finite difference methods for fluid flow problems

Aditya C. Velivelli
Iowa State University

Development of a multiblock solver utilizing the lattice Boltzmann and traditional finite difference methods for fluid flow problems

by

Aditya C Velivelli

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Iowa State University

Ames, Iowa

2008

TABLE OF CONTENTS

ABSTRACT

This dissertation develops the lattice Boltzmann method (LBM) as a strong alternative to traditional numerical methods for solving incompressible fluid flow problems. The LBM outperforms traditional methods on a standalone basis for certain problem cases while for other cases it can be coupled with the traditional methods using domain decomposition. This brings about a composite numerical scheme which associates the efficient numerical attributes of each individual method in the composite scheme with a particular region in the flow domain. Coupled lattice Boltzmann–traditional finite difference procedures are developed and evaluated for CPU time reduction and accuracy of standard test cases. The standard test cases are numerical solutions of the two-dimensional unsteady and steady convection-diffusion equations and two-dimensional steady laminar incompressible flows represented by the backward-facing step flow problem and the flow problem around a cylinder. Multiblock Cartesian grids and hybrid Cartesian-cylindrical grid systems are employed with the composite numerical scheme. A cache-optimized lattice Boltzmann technique is developed to utilize the full computational strength of the LBM. The LBM is an explicit time-marching method and therefore has a time step size limitation. The time step size is limited by the grid spacing and the Mach number. A lattice Boltzmann simulation necessarily requires a low Mach number since it relates to the incompressible Navier-Stokes equations in the low Mach number limit. For steady state problems, the smaller time step results in slow convergence. To improve the time step limitation imposed by the grid spacing, an improved LBM that adopts a new numerical discretization for the advection term has been developed and the results were computed for a convection-diffusion equation and compared with the original LBM. The performance of traditional finite difference methods based on the alternating direction implicit scheme for the convection-diffusion equation and the vorticity-stream function method for the laminar incompressible flow problems is evaluated against the composite numerical scheme. The composite numerical scheme is shown to take lesser CPU time for solving the given benchmark problems.

CHAPTER 1.

INTRODUCTION

The design and optimization process in various industries is dependent on results from high-fidelity simulations such as computational fluid dynamics (CFD). However, high-fidelity simulations by their very nature take up a significant amount of time. Consequently, interactive engineering design or optimization of many practical systems is currently not possible. Even with parallel computation, faster clock speeds on processors, and increased memory, CFD simulations are yet to be completed within a reasonable time span with regard to interactive design [1,2,3]. Because of this, there is a significant need to develop new algorithms and methods within CFD that can aid in the interactive design process. The requirements for such algorithms would be significant reduction in computation time, sufficient accuracy to support decision making, and appropriate simplicity to enable a non-CFD expert to use them. These new algorithms must take advantage of improvements in computation hardware such as increased or massive parallelization, increased cache memory, and increased clock speed on processors.

Recent advances in CFD have focused on providing more accurate models or developing new models for the many complex flow problems. Advances have also been made in developing efficient and accurate numerical discretization schemes as well as robust solvers for the linear algebraic equation systems that result from discretization of the flow equations. However, none of these advances has improved the wall-clock time for computation [4], an important factor in reducing design costs. In addition, parallelization of the traditional CFD methods has not brought about the required decrease in turnaround time, as most of the methods do not possess locality in their computational nature (which means there are dependencies that require considerable communication between processors on the parallel computer). Several means of achieving faster turnaround time have been developed. These include approximate, reduced-order models, low-fidelity models based on Bayesian statistics, or complementing CFD methods with classical analytical techniques. Each of these has significant limitations within an interactive design environment and may not yield the required accuracy and detail provided by high-fidelity models.

The engineering design process essentially consists of three stages: conceptual design, preliminary design, and final/detailed design. Within each stage, an iterative process takes place until the final design is produced. For instance, consider the conceptual design stage for the exterior body of an automobile. The designer would be interested in evaluating different conceptual designs of the exterior with the intention of obtaining an eye-pleasing shape. At the same time, the designer would like the automobile exterior to be aerodynamically efficient, leading to fuel savings. To achieve this goal, information about the flow field is required. Because wind tunnel testing and CFD are time-consuming, they generally are not able to support the large number of options that are considered at the conceptual design stage [5]. Instead, generic design principles (e.g., dominant design basis) are used, or if additional information and insight are needed, low-order approximation methods are used to decrease the time and cost.

However, these methods may lead to grossly inaccurate results, which may only be detected in the detailed design stage, leading to higher costs.

To reduce the CPU time cost for solving incompressible flow problems, this work combines different levels of physical modeling and discretization using domain decomposition. On different subdomains, the following are modeled and solved:

1. different partial differential equations (the discrete velocity Boltzmann equation and the Navier-Stokes equations)
2. different grids (body-fitted, Cartesian)
3. different time step sizes

This thesis presents a new numerical scheme that couples the lattice Boltzmann method (LBM) with traditional finite difference methods for solving incompressible flow problems on hybrid multiblock grids, leading to a novel fluid flow solver that performs faster computations. The LBM is a fast high-fidelity solver that makes full use of cache memory, parallel computation, and increased processor clock speed. The relative efficiency of the standalone LBM is restricted to simple flow geometries with Cartesian gridding and to situations requiring small numerical time steps for any numerical method. Due to these inherent limitations, the LBM is better utilised when coupled with traditional methods. Using a hybrid multiblock grid brings about accurate and efficient resolution of flow phenomena. The coupled scheme on multiblock grids has been implemented for the two-dimensional Burger's equation, flow across a backward facing step, and flow around a cylinder.

1.1 Objectives

The previous paragraph explained the background for coupling LBM with traditional methods. The main objectives of this study were to:

1. Develop the capabilities of the LBM using cache optimization, parallel computing, and improved numerical discretization schemes
2. Combine LBM with traditional finite difference methods using domain decomposition and show the resulting solver to be faster than efficient existing schemes
3. Execute the composite solver for standard test cases such as the two-dimensional Burger's equation, flow across a backward facing step, and flow around a cylinder

1.2 Dissertation Organization

This chapter has mentioned the need for a faster high-fidelity solver and introduced the background to a composite numerical solver.

In Chapter 2, a brief overview of numerical schemes, numerical grid types, and their relative advantages are presented. Chapter 2 also lists the previous attempts at developing fast solvers for predicting fluid flow and the drawbacks of these attempts.

In Chapter 3, the LBM is introduced with regard to the two-dimensional Burger's equation. A cache-optimization algorithm is developed to take advantage of the LBM's localized computational nature. The cache-optimized LBM is compared with the original LBM and with efficient traditional finite difference methods on parallel processing architectures.

Chapter 4 develops two methodologies to improve the performance of the LBM for steady state problems. In the first methodology, a new spatial discretization is adopted for the convection term to allow the LBM to assume bigger time steps. The second methodology couples the LBM with the alternating direction implicit scheme for solving the two-dimensional steady Burger's equation on a multiblock grid. This approach is significantly faster than using the alternating direction implicit method on the multiblock grids on a standalone basis.

Chapter 5 evaluates the composite or coupled numerical scheme for the backward facing step flow and for flow around a cylinder. The vorticity-stream function formulation is used here as the traditional Navier-Stokes solver.

Chapter 6 presents the conclusions of this study and recommendations for future work.

CHAPTER 2.

BACKGROUND

CFD is an interdisciplinary science that uses computers to solve the partial differential equations (PDEs) describing the conservation laws applied to fluid dynamics. Progress in CFD has been intractably linked to improvements in computer hardware. A CFD simulation consists of the following stages:

- Reading a computer aided design (CAD) model of the flow domain
- Grid generation of the flow domain
- Applying numerical methods to solve the PDEs modeling the fluid flow in the given domain
- Post-processing of the results through visualization

The need for improved interfaces to CAD systems or accurate representation of geometry will not be addressed in this study. Geometry models that meet the requirements of continuity and smoothness needed for flow simulation are assumed to be available.

2.1 Grid Generation

The grid generation process decomposes the given flow field and geometrical objects into discrete points and discrete volumes or elements. Present modes of grid generation are:

- Cartesian

- Structured body-fitted
- Unstructured
- Semi-structured
- Hybrid

2.1.1 Cartesian Grid

Uniform Cartesian grids, which are essentially square grids, were the earliest and also the simplest to be utilized in the grid generation process. However, uniform grids are limited to mapping simple regions. Cartesian grids are used with local refinement (non-uniform grids) to capture gradients in a flow field as well as curved boundaries of the flow domain. The Cartesian grid methods with local refinement have enjoyed success when applied to inviscid flows around complex geometrical configurations [6].

Cartesian grid generation is automatically generated, aligned with the Cartesian axes, ignoring the complexity of the input geometry (this is the same as putting the required geometries into their positions in a Cartesian grid). After the initial stage, the grid generation process has to perform two kinds of refinement: geometry adaptive refinement and solution-adaptive refinement. For both kinds of refinement, there is a widely-used approach that is based on using quadtree or octree data structures [7]. An adaptive octree approach adds spatial refinement to regions that require more discretization to capture either irregular geometrical (body) surface or steep gradients in the solution. The refinement procedure for geometry is implemented by spatial subdivision of the grid cells (quadrilateral cells) near and across the body surface. A spatial query operation is performed to determine cells that contain the body surface. The spatial subdivision is performed by bisecting the cell in each of the coordinate directions. This refinement procedure is applied recursively (i.e., on the newly formed cells) until the minimum cell size becomes less than a specified value. The specified value is dependent on the application of the octree and can be chosen to be proportional to the radius of the curvature of the body surface. To retain smoothness in the grid, the grid cells should become progressively smaller.

The next step is to address the problem of cells that intersect the body (since the body boundary, or surface, is not necessarily located on the grid points). The cells that lie completely inside the body are removed. Most Cartesian grid generation methods adopt the cut-cells method [8] to take care of the cells containing the body surface. Using this method necessitates the use of unstructured solvers such as finite volume methods. To use a finite difference (structured) solver, other approaches should be used, such as the "stair step" approach, which defines the body as aligned with cell edges. However, this approach sacrifices the ability to represent the body surface accurately. Solution adaptive refinement in the Cartesian grid can be performed by superposing a finer grid on the initial coarse grid in a critical region. This method is recursive in that grid patches with finer resolution can themselves be nested within other grid patches.

A different approach is to use a single Cartesian grid with spatially varying grid resolution (a non-uniform rectangular grid where grid spacing monotonically increases or decreases along a Cartesian coordinate direction). A third approach is to use multiblock Cartesian grids. Here, the computational domain is discretized with a set of overlapping uniform Cartesian grid components. Refinement can be performed with the adaptive spatial partitioning and refinement method (ASPaR) [9]. This method can also be used for resolving geometry of body boundary in the flow. To select one of these methods for solution adaptive refinement and geometry adaptive refinement, it should be known which of them would work best for the flow solver of choice on Cartesian grids. Cartesian grids with local refinement are not sufficient to solve viscous flow problems at a high Reynolds number, where occurrence of boundary layers requires high grid resolution. Efforts to counter this problem include development of a Cartesian grid method using local anisotropic refinement [10]. However, this method has only been verified for low Reynolds number flows, and it requires the boundary to be aligned with one of the principal coordinate directions.

2.1.2 Structured Body-Fitted Grid

Structured body-fitted grid generation involves mapping a logical space with a Cartesian grid to the actual physical space, which may be non-rectangular and arbitrary. The mapping can be interpreted as using a curvilinear coordinate system for the physical space. This new coordinate system conforms to the boundary of the physical space unlike the Cartesian system, where nodes of the grid may not coincide with the boundary. Structured grid schemes have proven to be the best when dealing with high Reynolds number flows, which have strong directional, flow gradients. This is because structured grids allow a cell shape that is elongated in the flow direction; i.e., more grid points appear in the direction normal to the flow than in the flow direction. The curvilinear coordinate system transforms the governing equations and thus increases the complexity of the problem, unlike the Cartesian system, where the governing equations remain unaffected.

To generate a structured boundary-fitted grid, solving a PDE such as the Poisson equation (elliptic grid generation) is usually required. The transformations and numerical solution of PDEs involved in generating the structured boundary-fitted grids take up considerable compute time when complex geometries are involved. It is difficult to automate this kind of grid generation since it is usually necessary to decide the grid topology, zoning, and grid-clustering locations manually. Multiblock structured grid generation is utilized when treating very complex domains. In this procedure, grids are generated separately in each block and patched at block faces or allowed to overlap [11].

2.1.3 Unstructured Grid

Unstructured grid generation involves decomposing the domain into tetrahedra. The tetrahedra are generated through Delaunay triangulation, advancing front methods and domain decomposition techniques. Unstructured grids include varying element topology and size, unlike structured boundary-fitted grids and Cartesian grids, where all grid elements are similar in shape (e.g., rectangles or hexahedra). However, surface modeling requirements for unstructured meshes can be demanding and time-consuming [12]. Unstructured grids require greater memory to provide connectivity and topology information. For this reason, flow solvers based on unstructured grids are usually not as efficient as their structured counterparts.

Isotropic tetrahedral grid generation has been automated for flow fields about complex shapes, but when it comes to dealing with high Reynolds number flows with thin boundary layers, unstructured grid generation faces the same problems as Cartesian grid generation. It is very expensive computationally to generate tetrahedral cells with high aspect ratios to resolve such boundary layers or other strong directional flow gradients. New developments in unstructured grids involve polyhedral meshing [13], where a control volume is allowed to possess as many polygonal faces as necessary. This means that a control volume can possess any shape. The polyhedral meshing method is an efficient way of specifying hexahedral cells in an unstructured grid. It provides flexibility in generating grids in critical regions with complex geometry.

2.1.4 Semi-Structured or Prismatic Grid

The prismatic grid is generated from an unstructured triangular grid representation of the body surface. Marching the body surface triangulation outward in distinct steps results in the generation of prismatic cells in the marching direction. In two dimensions, the line segments on the boundary are marched outward, giving rise to quadrilateral cells. The marching direction is the same as the normal vector at each node. The normal vector at each node is computed as the weighted average of the normals of the common faces. Karman [14] uses an iterative procedure (smoothing) to further refine the node normals using a linear combination of a weighted average of the normal vectors of the common faces and a weighted average of the position vectors of the neighboring nodes. This computation works well for many complex shapes. The marching step for advancing the grid to the next layer is computed based on a user-specified value and can be equally spaced. Constraints are imposed on the spacing increment for the marching to avoid a stretched or skewed grid. Prismatic grids are usually used to discretize the boundary layer region, so the total thickness of the prismatic grid will be slightly greater than the boundary layer thickness. The rest of the domain is covered using Cartesian grids or unstructured grids, which leads to hybrid grids.

2.1.5 Hybrid Grid

The problems encountered with the above grid generation strategies have given rise to hybrid grids. Hybrid grid generation strategies consist of combining unstructured grids or Cartesian grids with structured boundary-conforming grids. The region around the body (boundary layer) as well as other high gradient regions such as wakes are meshed with a semi-structured body-conforming scheme such as the prismatic grid. The rest of the domain can be meshed either with Cartesian grids (Figures 2.1 & 2.2) or unstructured tetrahedral grids. In this study, only hybrid Cartesian-prismatic grids are considered. In such a hybrid grid, a prismatic grid covers the body surface and the Cartesian grid overlaps (chimera type) into the outer layer of the prismatic grid. Without overlapping, the Cartesian grid cannot accurately or smoothly represent the interface between the two grids unless the cut-cells method is used. The Cartesian grid resolution should match that of the prismatic grid in the overlap zone.

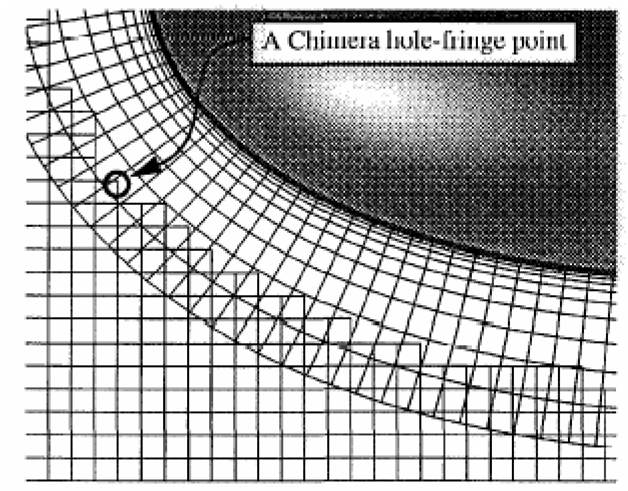


Figure 2.1. A hybrid chimera grid, Meakin [9]

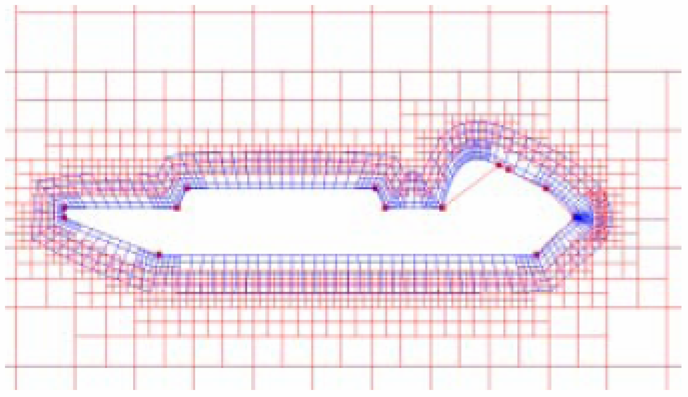


Figure 2.2. An overlapping hybrid grid around an amphibious vehicle, Wang et al. [16]

Interpolation will pass data between the nearest prismatic and Cartesian grid points (solution exchanges at intergrid boundary points). For a given intergrid boundary point associated with the Cartesian grid, the coordinate indices of the corresponding prismatic grid point need to be identified. This is computed with an iterative search procedure whose cost is proportional to the number of intergrid boundary points. Melton et al. [15] suggest a procedure for locating the prismatic cell that contains the centroid of each Cartesian cell to establish links between the innermost layer of Cartesian cells and the prismatic grid and vice versa to establish links between the outermost layer of the prismatic cells and the Cartesian grid.

2.1.6 Grid Changes Corresponding to Geometry Changes

From a design point of view, the surface of the body should be deformed to test different body configurations. For example, with the hybrid grid, the prismatic grid that exists near the body surface should move and deform with the boundary. The innermost layers of the prismatic grid remain attached to the boundary during movement, while the outermost layer of edges remains fixed. To do this, Wang et al. [16] treat the prismatic grid as consisting of root cells that are subdivided using a quadtree structure. They then perform transfinite interpolation on the geometry of the deformed root cell to compute the deformation of cells (leaf cells in a root cell) in the prismatic layer. While the prismatic grid deforms with the body surface and changes its resolution, the Cartesian cells near to and overlapping the outermost layer of the prismatic grid also get refined or coarsened to match the prismatic cell size. However, the prismatic grid is regenerated if the deformation of the prismatic grid violates any of the quality criteria specified [16]. The solution from the previous grid can be interpolated onto the new grid to provide a starting point for the ensuing computations.

The connectivity information in the hybrid grid remains the same, which saves computation time that would otherwise have been spent on figuring out the new connectivity relations. However, regenerating the prismatic grid multiple times can prove expensive. Therefore, the ability to reuse the original hybrid grid for subsequent design geometries would be desirable. Simulations performed on the new geometries can use the solution from the previous design geometry as an initial condition.

McMorris and Kallinderis [17] developed a method to reuse a prismatic/tetrahedral grid for the new geometry, which was obtained with slight changes to the body surfaces of the original geometry. They assume that most changes during the design process are relatively small and that the new geometry maintains the same topology as the original geometry. Retaining the same topology means surfaces cannot be split or added to the geometry. The procedure initially involves mapping the grid points on the boundary of the original geometry onto the new geometry. Kallinderis utilizes the points on the curves of the CAD geometry for mapping onto the new geometry, keeping the same spacing along the curves. The next step maps those points on the boundary surface that lie in the interior (not on the curves) according to the motion of the points on the curve. After this, the points on the interior of the hybrid grid are moved according to the weighted influence of one or more of the nearest boundary points. McMorris et al. [17] used a tetrahedral/prismatic grid, whereas a Cartesian/prismatic grid will be used in this study.

2.2 Numerical Solution of Fluid Flow Equations

To obtain a numerical solution to fluid flow problems, the differential conservation laws represented by PDEs need to be discretized on the grids mentioned above. This in turn gives a series of algebraic equations whose numbers are dependent on the total number of grid points. The algebraic equations can be explicit or implicit in nature depending on the discretization scheme. Explicit means that the algebraic equations can be solved independent of each other. When implicit, they need to be solved as a system of equations. The grid type and the formulation of the discretization procedure are dependent on each other.

2.2.1 Finite Difference Method

The function of continuous arguments representing the dependent variable in the given PDEs is defined at discrete points, such as the nodes of the grid. The derivatives present in the PDE and the boundary conditions are approximated by difference expressions, transforming the PDE into a system of algebraic equations. The relation between the derivative and the difference expression is obtained through a Taylor series expansion. The finite difference method requires the use of a Cartesian or a structured body-fitted grid.

2.2.2 Finite Volume Method

Finite volume methods discretize PDEs by transforming them so that they resemble the conservation laws in an integral form applicable to a region in space (control volume). This region in space can be represented by hexahedral cells in a structured grid or tetrahedral cells in an unstructured grid. The integral form can also be obtained by applying the balance equations, known from first principles, to a control volume. Both approaches, when applied over cells in a given grid, give rise to a system of algebraic equations.

2.2.3 Finite Element Method

The finite element method divides the domain to which the PDE applies into simple pieces (polygons) known as "elements." The solution (dependent variable in the PDE) is then approximated by extremely simple functions on these elements. For instance, the elements can be triangles and the simple functions can be linear. The domain is divided into a finite number of triangles with, for example, N interior vertices. N trial functions can be picked, one for each vertex, such that the trial function is non-zero only at its specified vertex. Inside the triangle, each trial function is a linear function with different sets of coefficients for each triangle. The global solution is approximated as a linear combination of these trial functions. The coefficients of this linear combination are obtained as the solution of an energy minimization problem, which involves solving a system of linear or non-linear algebraic equations. The finite element method is especially suited to handle curved or irregularly shaped domains. It can be used on any type of grid, like the finite volume method.

2.2.4 Explicit and Implicit Methods

PDEs with a time-dependent term are marched in time to solve for the dependent variable. Stationary or time-independent PDEs can also be solved in a time-marching manner by starting the calculation from some initial approximation. The stationary PDE is considered to be a limit of its corresponding non-stationary PDE with stationary boundary conditions. Explicit difference schemes give rise to algebraic equations that can be solved independent of each other. The explicit difference schemes have a strong numerical time step limitation. Certain explicit schemes such as higher order Runge-Kutta schemes and those with local time stepping have been used to improve the time step limitation. To avoid the time step limitation, implicit methods that are unconditionally stable are used. Implicit difference schemes lead to a system of algebraic equations that need to be solved simultaneously. For this, a number of solvers have been developed such as alternating direction implicit (ADI) schemes, Gauss-Seidel schemes (with successive overrelaxation), and acceleration techniques such as multigrid procedures. It is more difficult to parallelize these solvers than those belonging to explicit schemes. Explicit schemes perform more efficiently (better parallel speedups) when parallelized than their implicit counterparts.

2.3 Fast Models for CFD

Despite the advances in CFD theory and computer hardware, designers are still demanding faster models for performing analysis in the conceptual design stage. A number of models have been developed for obtaining fluid flow simulation results with less wall clock time than before. They include both low-fidelity models with improved accuracy and fast high-fidelity models.

2.3.1 Low-Fidelity Models

Low-fidelity models have been used in the conceptual design stage since the 1970s. However, these models either simplified flow physics by solving for potential flow or they reduced the accuracy by using two-dimensional models and coarse grids. To make headway toward the goal of faster and accurate design, low-fidelity models that extract information from high-fidelity models have been and are being developed. They go by various names such as reduced-order or reduced-basis models, Bayesian models, etc. [18,19].

2.3.1.1 Reduced-Basis Method

Reduced-basis methods are reduction methods that reduce the degrees of freedom in the problem of interest. To do so, they construct a low-order approximation space composed of solutions of the PDE (or problem of interest) at selected points in the parameter/design space. The solution for the PDE at other arbitrary points in the parameter/design space is treated as a linear combination of the basis vectors from the low-order approximation space. To determine the coefficients of the linear combination, a finite element calculation is performed. Since this results in a dense system of equations, the reduced-basis method is efficient only if a small number of basis vectors are specified. The reduced-basis method works well in an interpolatory setting. If the basis vectors do not represent all the features (e.g., dynamics of the states encountered in the design process), then the reduced-basis method may fail; i.e., it will not work for an extrapolatory setting.

Peterson [18] has applied the reduced-basis method to compute steady incompressible flow solutions for high Reynolds numbers. Normally, for a given Reynolds number, solutions to the steady Navier-Stokes equations are obtained by applying an iterative method such as Newton's method. The Newton method requires a very good starting estimate for convergence at a high Reynolds number, so the usual solution method involves calculating flow solutions at a lower, but increasing, sequence of values of Reynolds numbers, with each new flow solution used as the starting estimate for the Newton iteration at the next Reynolds number. Such a procedure can become highly expensive for high Reynolds numbers. Peterson's reduced-basis method uses flow solutions at lower Reynolds numbers obtained through the Newton method as the basis vectors to obtain a solution for higher Reynolds numbers. This solution is then used as the starting guess for the Newton method to solve the Navier-Stokes equations at high Reynolds numbers. The reduced solution is usually a good enough initial guess and Newton's method takes very few iterations to converge. Peterson has shown that for a Reynolds number as high as 5000, only 5 reduced-basis vectors were required. The cases solved include a two-dimensional cavity flow and the forward-facing step flow.

One limitation to applying these models for interactive design is that the reduced-basis methods may be based on high-fidelity solutions for just a few different geometries. This may provide insufficient information to explore the entire design space. For instance, a certain design/geometry change may bring about a steep gradient or shock in the solution, which may not exist in the solution corresponding to other closer geometric variants. If the reduced-basis model were a linear combination of prior solutions to such variants, the gradient would not be predicted.

2.3.1.2 Bayesian Methods

A Bayesian approach [19] can be used to predict the output for complex computer codes that have simpler analogues. The simpler code runs much faster, but is less accurate. For instance, the simpler code could be a finite difference or finite element code that uses a coarse grid to model the problem at hand, unlike the complex code that uses a high-resolution grid. The Bayesian approach models the difference between the two codes as another unknown function, and learns about it using data comprising a small number of runs of the slow complex code together with a much larger number of runs of the fast code. This builds an emulator of the slow code that can be used to predict its output when the fast code is executed. The Bayesian approach assumes that different levels of the same code are correlated in some way. This is a disadvantage when the complex code does not possess a simpler analogue. For instance, a numerical method operating on a coarse grid can resolve only some features of the flow and may not have any correspondence with the same numerical method operating on a high-resolution grid and resolving all scales existing in the flow. The Bayesian approach also assumes that each level of code (simple or complex) provides output values that are reasonably close for similar inputs. This limitation is similar to those of reduced-basis methods that require an interpolatory setting to work well. Again, the disadvantage is the number of runs of the high-fidelity code to model the anticipated design changes. The Bayesian approach has been successfully implemented with two codes that simulate oil pressure at a hydrocarbon reservoir well. Both codes use finite element analysis and differ in the resolution of the grid.

2.3.2 Fast High-Fidelity Models

The need for fast high-fidelity models was discussed in Chapter 1 and is obvious due to the drawbacks of current low-fidelity models. According to Bram Van Leer [20], for a high-fidelity model, the ideal computational cost for a problem with N unknowns (N corresponds to the total number of grid points) should be a O(N) operation count (order of N floating point operations). This property implies linear scalability with regard to the number of unknowns or the grid size. Solution of elliptic equations using multigrid techniques leads to convergence in O(N) operations. Steady solutions to Euler equations can also be obtained with O(N) operations [20]. However, the solutions to the stationary Navier-Stokes equations have not reported such a convergence so far (operation count is $O(N^2)$ or worse) [20]. Present-day CFD tries to accelerate computations of high-fidelity models using parallel computers. However, parallelization does not provide scalability with regard to increasing grid size. A few high-fidelity models that may have O(N) convergence are described below.

2.3.2.1 Using First Order PDEs

The Navier-Stokes equations are essentially a system of second-order PDEs. They can, however, be reduced to a system of first-order PDEs without sacrificing their ability to model flow physics accurately [20]. Such PDEs consist of advection terms, local, and stiff source terms, and are called hyperbolic-relaxation equations. The first-order system of equations is always larger than the parent system of PDEs. For instance, the diffusion equation can be reduced to a first-order two-equation system called the hyperbolic heat equation and the five three-dimensional Navier-Stokes equations can be rewritten as a first-order system of fourteen equations. Some advantages of using a first-order system of PDEs as listed by Bram van Leer [20] are:

- First-order PDEs require the smallest possible discretization stencil. This reduces communication in parallel computations.
- Local implicit integration can treat local source terms that are stiff.
- Functional decomposition can be applied to large systems of first-order PDEs. Functional decomposition allots each PDE or group of PDEs to separate processors, while at the same time domain decomposition can be applied to solve the PDEs simultaneously on all the available processors. This allows a larger number of processors to be used without losing linear scalability. For both kinds of decomposition to work together, a distributed memory machine consisting of clusters of memory-sharing processors is required.

The hyperbolic-relaxation system approach has been successfully applied in gas dynamics and requires less CPU time than methods based on solving Navier-Stokes equations. However, the hyperbolic-relaxation approach has not been developed to solve the unsteady Navier-Stokes equations or to obtain solutions to flows involving turbulence.

2.3.2.2 Flow Network Model

The flow network model was originally developed for pipe flow networks, where it is required to calculate changes in static pressure across any branch or element in the network. In the context of CFD, the flow network model (specified as vectorized flow network model or VFNM) treats fluid flow regions as a resistance/flow network. The directional flow network consists of pipes (branches) that join at nodes to make a whole pipe network. The magnitude of velocity inside these pipes is constant and changes only at the nodes. Mass conservation is automatically satisfied at each node (flow into and out of each node is equal). The conservation of momentum law controls the flow rate in a pipe. Kim et al. [21] verified the VFNM using two-dimensional cavity flow as an example. Horizontal and vertical pipes that are equidistant from each other and enjoined at nodes represent the flow domain. The nodes are indexed with Cartesian coordinates. Flow is one-dimensional in each pipe, with the positive direction being associated with those of the Cartesian axes. Mass conservation is applied at the nodes in terms of the horizontal and vertical velocities. Conservation of momentum is considered along the pipes and is applied in terms of shear stresses and pressure differences. The shear stresses are expressed in terms of velocities.

Momentum conservation is also applied along closed loops (a loop starts from a node, passes through some closed branches, and returns to the original node). Changes in the flow domain (geometry) will require changes in the flow network topology (connectivity of the nodes caused by different piping configurations). An automatic scheme for loop equation generation based on topology analysis and a network search algorithm has been developed for this purpose [22]. The VFNM has demonstrated good accuracy in the two-dimensional cavity flow for low Reynolds numbers [21]. It is not known if this method can predict solutions with steep gradients and turbulence. Kim et al. have claimed enormous enhancements in the computation speed compared to traditional methods; however, there has been no study about the scalability with regard to number of unknowns. The VFNM is expected to have O(N) convergence based on the equations given. An advantage in using this method is that changes in geometry can be easily dealt with by changing the flow network topology. The disadvantage is that this method is limited to modeling low Reynolds number flows and steady state phenomena.

2.3.2.3 Using Classical Analytical Techniques

Michal et al. [23] suggest coupling complementary analytic methods and numerical methods to reduce the overall number of grid points and to incorporate more physics into the solution algorithm. The flow domain is partitioned so that the traditional CFD method solves for complex phenomena such as shocks in one partition, while efficient analytic methods model the flow field in the remaining partitions. Analytic solutions are developed by using efficient analytic techniques appropriate to each partition. This strategy reduces the number of grid points considerably. The analytic model is derived from analytic solutions of an asymptotic form of the three-dimensional, steady state Euler equations. Viscous effects are included by coupling the above-defined scheme with an interactive boundary layer method [24]. This approach has good potential for aerodynamic design optimization since the analytic solutions can be differentiated to provide direct estimates of aerodynamic design sensitivities. Airfoil design optimization methods that use the analytic methods coupled with traditional numerical methods have shown reductions in computational cost by two orders of magnitude when compared to traditional CFD methods. The coupled scheme has not been tested for solving the full, unsteady Navier-Stokes equations.

2.3.3 Summary of the Fast Computational Methods

The low-fidelity methods described in Section 2.3.1 can work well in an interpolatory setting; i.e., they can interpolate between a set of previous solutions and find new solutions. They may not find solutions that lie outside the space formed by the previous solutions. Although the reduced-basis method seems to work well for predicting flow fields at higher Reynolds numbers using solutions at lower Reynolds numbers, there is no guarantee that it will work well for geometry changes. The fast high-fidelity models that were described in Section 2.3.2 do not face similar drawbacks.

Bram Van Leer's method of reducing the Navier-Stokes equations to first-order PDEs has not been developed for many practical flows that require the solution to the unsteady Navier-Stokes equations. The vectorized flow network model works for simple, low Reynolds number flows and has yet to be validated for practical flows. Complementing numerical methods with analytic methods has worked well for inviscid flows and for viscous flows, through coupling with the interactive boundary layer method. Again, this scheme has not been tested for unsteady Navier-Stokes equations. This scheme can be improved by replacing the numerical methods used here with more efficient methods, which will be discussed later in this document.

2.3.4 Lattice Boltzmann Method

The lattice Boltzmann method (LBM) is a numerical method for solving problems involving fluid flow. Unlike finite difference, finite volume, and finite element methods that solve macroscopic conservation laws in the form of PDEs, the lattice Boltzmann approach is based on solving the discrete velocity Boltzmann equation from statistical mechanics. The discrete velocity Boltzmann equation is a set of PDEs that describe the evolution of particle distribution functions. Particle distribution functions define the probability of finding a particle at a certain location with a certain velocity and at a certain time. Traditionally, the LBM has been an explicit finite difference approach towards solving the discrete velocity Boltzmann equation.

The LBM possesses some of the positive aspects of the fast high-fidelity methods mentioned previously. For instance, the equations that need to be solved are first-order hyperbolic PDEs, which bestows on the numerical method that attempts their solution all the advantages listed by Bram van Leer [20]. The LBM is very well-suited for Cartesian grids, which are the fastest and easiest to generate. The localized and explicit nature of lattice Boltzmann computations provides them with a distinct advantage over other methods when computations are performed on parallel computers.

2.4 Multiblock and Multi-Solver Techniques

Section 2.1 described different grid types while Section 2.3 focused on various fast high-fidelity numerical methods or their approximations. This section will detail numerical schemes that are implemented on multiblock grids. Many flow problems consist of complex geometries, leading to difficulty in generating a single grid to cover the entire flow domain. The multiblock method of grid generation results in individual grid blocks corresponding to particular regions of the flow domain. The grid components adjacent to walls and obstacles in the flow domain can be fitted to the boundary of those geometries, as discussed in the section on hybrid grids. Therefore, the grid blocks can be of completely different types and can overlap at their interfaces. The multiblock arrangement leads to efficient grid refinement, since local grid refinement can be performed over certain regions of the flow domain by simply covering those regions with high-resolution or fine grid blocks and the rest of the domain with low-resolution or coarse grid blocks. The fundamental principle behind the multiblock technique is to split the flow domain into two or more overlapping subdomains (blocks). The equations of motion or their equivalents are solved on each subdomain subject to specified boundary conditions. Conditions at inter-subdomain boundaries depend on the respective solutions in neighboring subdomains. Information can be exchanged using interpolation that relates the values of variables at the interface nodes of different grid blocks covering the subdomains.

Perng and Street [25] use a volume-averaged formulation to solve a weak form of the two-dimensional Navier-Stokes equations in primitive variables on a staggered overlapping grid system in Cartesian (x, y) coordinates. The equation for pressure is obtained by substituting the Cartesian velocity components into the discretized continuity equation. In a staggered grid, the two velocity components and the pressure are located at different positions in the grid cell. The pressure variable is located in the center of the grid cell, while the velocity components are located on the edges of the grid cells (Figure 2.3).

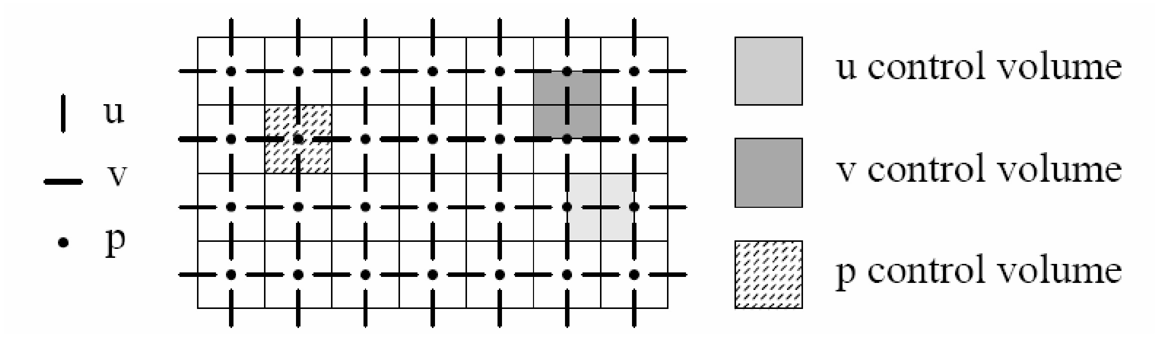


Figure 2.3

The time-explicit integration scheme used for the momentum equations allows them to be solved separately on each subdomain or grid block. This is because the information required to solve the momentum equations at the next time level, $(n+1)\Delta t$ is available from previous calculations at time $n\Delta t$. After the computation of the momentum equations on each subdomain (block), the relevant information is incorporated into the source terms of the pressure equation.

Unlike the momentum equation, the solution to the pressure equation on each grid block requires the velocity (normal to flow boundaries and/or interior boundaries) or pressure (at the boundaries) at time $(n+1)\Delta t$. To specify p^{n+1} or v^{n+1} on non-physical boundaries of subdomains inside the flow domain, the information available in the overlapping zones can be used; i.e., the pressure or normal velocity on the interior boundary of the grid block can be obtained from the solution field in the adjacent subdomain. This overlapping places the non-physical boundary of a subdomain in the interior of the adjacent subdomain. This technique essentially connects the individual pressure fields into a global one. The pressure field obtained in this way is globally consistent over the entire flow domain. Therefore, the resulting velocity profiles in the overlapping zones of different subdomains match exactly. The multigrid method is used to solve the pressure equation on each subdomain. The global solution for pressure is obtained by solving the pressure problem on each subdomain, sequentially cycling through them until convergence on all subdomains is obtained. Perng and Street tested their method for solving isothermal flow in a lid-driven square cavity with a square insert at the lower left corner and isothermal flow in a two-dimensional channel with abrupt expansion and contraction at two 90-degree bends

Brakkee et al. [26] also utilize a finite volume solver on staggered grids. They use an implicit time discretization for the momentum equations, giving rise to a domain decomposition problem for the momentum as well. Brakkee et al. solve the momentum and pressure equations separately over the composite domain, instead of solving these equations simultaneously in the subdomains. They achieve global discretization accuracy by enforcing the discretized momentum and pressure equations across subdomain boundaries. The GMRES method is used to solve the discretized equations.

Brakkee et al.'s scheme is equivalent to applying a block Gauss-Seidel or Jacobi iteration to the global discretization matrix. Their method is limited to matching grid blocks; i.e., the grids must match at the subdomain boundaries. Brakkee et al. tested their scheme for flow over a backward facing step and for flow around a cylinder in a wall-bounded shear flow.

Strikwerda and Scarbnick [27] solve the Stokes equations on overlapping subdomains that are discretized with either Cartesian grids or polar grids. For conservation of mass, they assume an integrability condition for the boundary data (represented by $\vec{b}$) of the composite domain

$$\int \vec{b} \cdot \vec{n} = 0 \tag{2.1}$$

It is difficult to impose this condition explicitly for each subdomain since the finite difference method obtains some of the boundary data for a subdomain using interpolation from other subdomains. Therefore, the authors assign a constant to the divergence of the velocity field in each subdomain. Instead of the continuity equation, the following equation is solved

$$\vec{\nabla} \cdot \vec{u}_i = d_i \quad \text{on} \quad \Omega_i \tag{2.2}$$

The constant d_i is determined by the integrability condition for the subdomain; i.e.,

$$\int \vec{u}_i \cdot \vec{n} = d_i \tag{2.3}$$

The Stokes momentum equation is similar to the Poisson equation, and therefore is discretized with the standard second-order accurate five-point Laplacian. The Stokes momentum equation and the modified velocity divergence equation are solved on each subdomain, using an iterative solution procedure. The (inner) iterative method used on the Cartesian domain is based on point-successive overrelaxation, while the iterative method used on the polar domain is based on line-successive overrelaxation. The inner iteration step consists of updating the velocity using the successive overrelaxation to solve the momentum equation and then updating the pressure at the interior point of the subdomain based on the local velocity divergence equation. The pressure at the boundaries of the subdomain is set by quadratic extrapolation of the interior values. Unlike the previous methods, the pressure on one subdomain does not directly interact with the pressure on the other subdomains. In fact, specifying the pressure as a boundary condition along with velocity results in an overdetermined boundary value problem on the given subdomain.

An outer iteration consists of a single inner iteration on each subdomain with velocity boundary data obtained from the other subdomain. The values of d_1 and d_2 are updated at each step using the integrability conditions. These outer iterations are performed until convergence is obtained. The solution is determined as converged when, on each subdomain, the changes in velocity are small and the changes in pressure are constant. After the convergence, a unique solution for velocity exists while the pressure is determined to within an additive constant. The deviation of the velocity fields from being divergence-free and the deviation from a constant of the difference of the two pressure fields on the overlap is used to indicate the accuracy of the final solution.

The above-mentioned methods dealt with a single numerical scheme operating on multiblock grids. Using multiple solvers on a multiblock grid can be efficient since certain solvers are more suited for computing certain regions of the flow domain and because certain solvers are computationally efficient on fine grid blocks while others are efficient on coarse grid blocks on account of the numerical time step size. It is possible to have different time steps on different grid blocks because the overlapping allows decoupling of the time step.

Mendu et al. [28] divided the computational domain into several blocks and applied different turbulence models (such as standard and low Reynolds number k-e models) in different regions (blocks). They applied their multiblock, multi-model method to the backward facing step flow and obtained more accurate results than conventional single-model computations.

Ikegawa et al. [29] implemented a finite element / finite difference (FEM/FDM) composite scheme for two-dimensional incompressible flow problems. This scheme combines the finite element method, which, according to the authors, is more useful for computing flow in an arbitrarily shaped geometry with the finite difference method that is advantageous in saving computing time and memory. The two methods are combined on an overlapping grid system, where the FEM computes on the near-body grid, while the FDM computes on the outer grid that partially overlaps the near-body grid. The authors solve the momentum equation using the arbitrary boundary marker and cell (ABMAC) method [29], which is an explicit, two-step predictor-corrector type scheme. The finite element mesh is made up of quadrilateral elements, and the velocity components are interpolated within an element using bilinear functions of local coordinates of that element. The pressure is assumed to be constant in each element. The finite difference computations are performed using the ABMAC method on a staggered grid. The variables on the interior boundary of each grid are obtained using the computed values from the other grid system. The interpolation from the FEM grid to the FDM grid can be performed using the bilinear functions of the FEM grid points surrounding the given FDM interior boundary grid point. Ikegawa et al. verified their method for both flow around a cylinder and the backward facing step flow.

Nakahashi [30] implemented an FDM-FEM approach for compressible viscous flows over multiple bodies. In this approach, an implicit finite difference method is applied at the near-body region with structured body-fitted grids, and the remaining flow region is computed using an explicit finite element method. A flow region with multiple bodies is divided into several zones that cover the highly viscous flow fields near the bodies and the connecting zones between the viscous zones. The connecting zones may have complex geometry; therefore, FEM on an unstructured grid is well-suited for them.

The FDM and FEM zones overlap with each other. The unstructured nature of the FEM grid allows the interior boundary points for both FDM and FEM grids to coincide with each other, precluding the need for interpolation treatment at the interior boundary points. The connecting regions for the FDM zones are assumed to be inviscid and, therefore, a finite element method for the Euler equations is considered. The combined approach was tested for a problem where an airfoil was located parallel to a flat plate and for turbine cascade flow.

References [31], [32], and [33] have coupled finite difference and vortex methods for incompressible flow computations. In an inviscid fluid, vorticity is neither created nor destroyed. It can undergo only convection and diffusion. Vorticity is, however, produced at a solid boundary in a viscous fluid and is then carried away by convection and diffusion. The flow field is determined by these processes and, in turn, controls the production of vorticity. Vortex methods are basically discretized representations of these aforementioned processes. Vortex methods involve introducing isolated line vortices, vorticity blobs, vortex balls, or toroidal vortices into the flow field and tracking them numerically using Lagrangian or a mixed Lagrangian-Eulerian scheme. The vortex methods cannot accurately predict the flow field near a solid surface, where viscous effects are dominant. Assigning boundary conditions near a solid surface is not easy. The vortex method, however, is relatively efficient in terms of computation time and accuracy for modeling flow regions where convection effects are dominant. Figure 2.4 shows a representation for the coupled finite difference and vortex method scheme. Ω_1 and Ω_2 are the finite difference and vortex domains. Vortex methods are used in flow regions where the vorticity is confined to areas of small dimension, while the finite difference methods are used near solid walls due to their flexibility in handling boundary conditions.

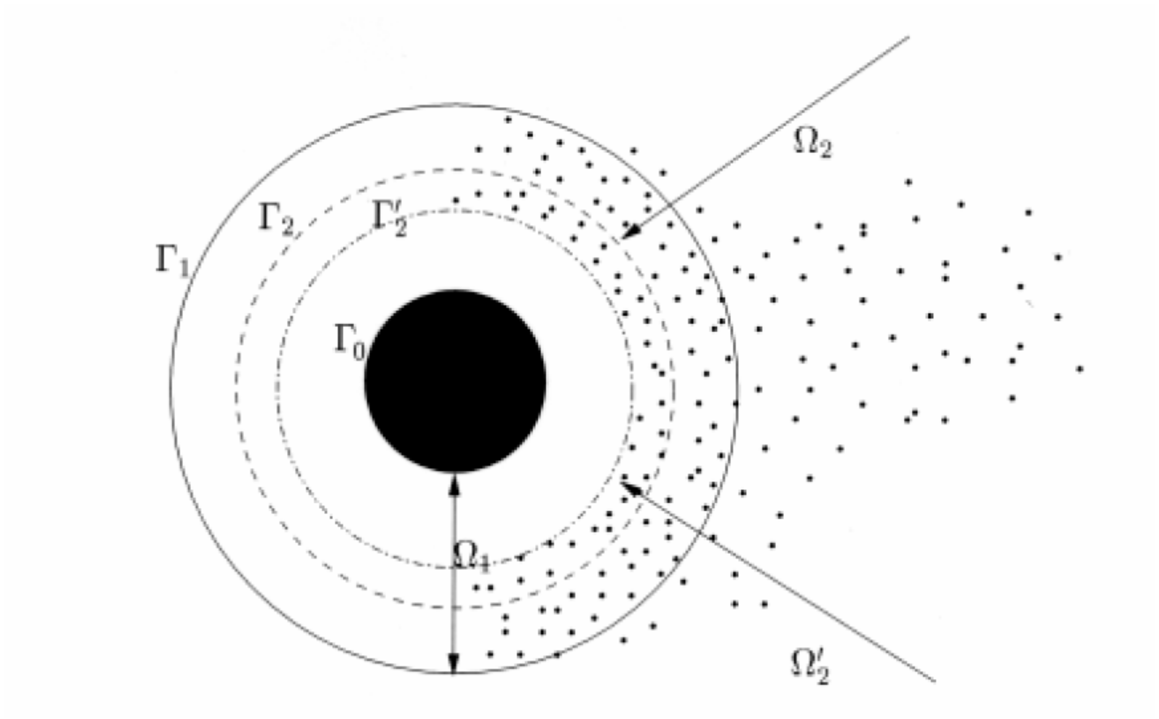


Figure 2.4

2.4.1 Summary of the Multiblock and Multi-Solver Methods

The multiblock methods have implemented traditional numerical methods on the subdomains comprising the flow domain. The multiblock methods provide a single numerical solver for the overlapping grid systems that discretize flow domains consisting of complex geometries. These numerical solvers may not be faster than other available solvers and they do not take into consideration the local attributes of each subdomain or the grid block covering that subdomain. The multi-solver methods aim to provide different solvers such that each individual solver in the multi-solver system is suited for a particular subdomain or the grid block discretizing that subdomain.

CHAPTER 3.

THE LATTICE BOLTZMANN METHOD AND CACHE OPTIMIZATION

The purpose of this research is to develop computationally efficient multi-solver or composite solver methods for multiblock Cartesian grids and hybrid Chimera grids. The Lattice Boltzmann method (LBM) has been selected to function as one part of the composite fluid flow solver due to being a comparatively fast high-fidelity solver on Cartesian grids. Floating-point operations in the LBM involve local data and therefore allow easy cache optimization and parallelization. This chapter presents the LBM and compares it with traditional finite difference methods for solving the two-dimensional Burger's equation on parallel computers.

.

3.1 Introduction to the LBM

The LBM has been used for simulating incompressible turbulence, multiphase and multicomponent fluid flows, particles suspended in fluids, heat transfer, and reaction-diffusion [34]. As discussed in Chapter 2, finite difference, finite volume, and finite element methods are based on discretizations of PDEs derived from continuum laws such as conservation of mass, momentum, and energy. In contrast, the LBM is based on the discrete velocity Boltzmann equation [35] and recovers the macroscopic continuum equations (i.e., the PDEs describing the conservation laws) by using a multi-scale expansion. The discrete velocity Boltzmann equation describes a system of particles statistically in terms of the particle distribution function, $f_i(\vec{x},\vec{v},t)$, where index i represents a specific velocity. The discrete velocity Boltzmann equation with the Bhatnagar-Gross-Krook (BGK) approximation [35] is

$$\frac{\partial f_i}{\partial t} + \vec{c}_i \cdot \nabla f_i = -\omega\left(f_i - f_i^{eq}\right) \tag{3.1}$$

where f_i represents the particle distribution functions, f_i^{eq} is the equilibrium distribution functions, ω is the collision frequency, and $\vec{c}_i$ is used to represent the velocities associated with the distribution functions. To discretize Equation 3.1 on a grid, upwind discretization is applied to the advection term, forward Euler discretization is applied to the time derivative, and downwind discretization is applied to the collision term (right-hand side of the above equation). This discretized version on a uniform lattice or grid is called the lattice Boltzmann equation. When the velocity equals the spatial differential over the time step, the lattice Boltzmann equation written in an explicit form is

$$f_i(\vec{x}+\vec{c}_i\Delta t, t+\Delta t) = f_i(\vec{x},t) - \omega\left(f_i(\vec{x},t) - f_i^{eq}(\vec{x},t)\right) \tag{3.2}$$

where i represents velocity index. A four-speed lattice Boltzmann model is shown in Figure 3.1.

$$\begin{aligned} \vec{c}_1 &= +c, 0 \\ \vec{c}_2 &= 0, +c \\ \vec{c}_3 &= -c, 0 \\ \vec{c}_4 &= 0, -c \end{aligned} \tag{3.3}$$

There are four links per node, each link having length Δx.

$$c = \Delta x / \Delta t \tag{3.4}$$

where Δt is the numerical time step.

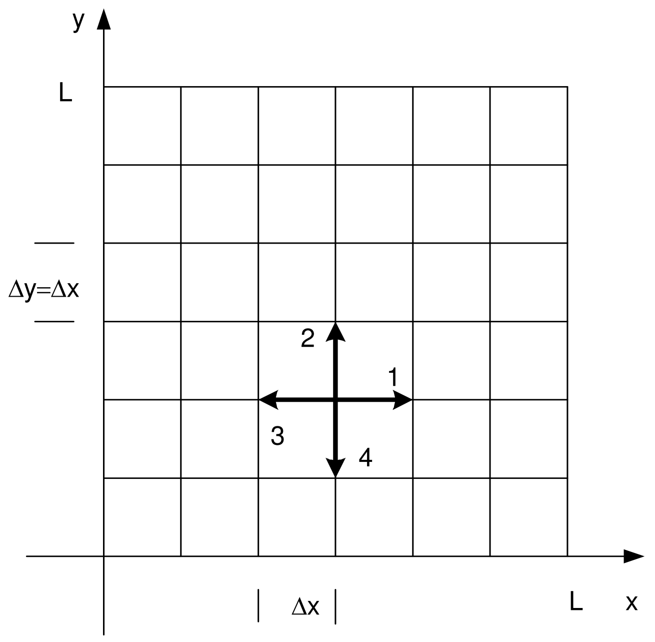


Figure 3.1. Square lattice with the four speeds shown at a node

3.1.1 Applying the Lattice Boltzmann Scheme

In this section and the following sections, the solution to the two-dimensional Burger's equation is chosen to demonstrate the capabilities of the LBM. The two-dimensional unsteady Burger's equation is studied as it contains non-linear convective terms and diffusion terms as well as a time-dependent term. The Burger's equation has been extensively used as a model for testing the efficiency and accuracy of various numerical schemes [36]. The Burger's equation acts as an adequate model for the convection-diffusion equation occurring in energy transport and for the vorticity-stream function approach for solving the incompressible Navier-Stokes equations.

The LBM will be applied to the time-dependent two-dimensional Burger's equation to demonstrate its ability to recover macroscopic equations through a multiscale expansion.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + u\frac{\partial u}{\partial y} = \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{3.5}$$

The dependent variable *u* in Burger's equation is defined as the sum over the distribution functions $f_i(\vec{x},t)$.

$$u(\vec{x},t) = \sum_i f_i(\vec{x},t) = \sum_i f_i^{eq}(\vec{x},t) \qquad i = 1,2,3,4 \tag{3.6}$$

A local conservation law; i.e., the conservation of mass, is inherent in the LBM. The conserved quantity is $\sum_i f_i(\vec{x},t)$. This would mean

$$\sum_i f_i(\vec{x} + \vec{c}_i\Delta t, t + \Delta t) = \sum_i f_i(\vec{x},t) \tag{3.7}$$

The equilibrium distributions f_i^{eq} depend on the conserved macroscopic quantity, *u*. The ansatz method described by Wolf-Gladrow [35] will be used to establish the relation between *u* and the equilibrium distributions.

3.1.2 Ansatz Method

The equilibrium distributions for simulating the two-dimensional diffusion equation on a two-dimensional lattice with four speeds at each node [35] were shown to be $u/4$, *u* being the dependent variable in the diffusion equation. From the lattice Boltzmann model for the Navier-Stokes equations [35], terms quadratic in *u* in the equilibrium distributions were noticed to yield the nonlinear advection term. Therefore, terms quadratic in *u* will have to be added to the terms in the equilibrium distributions for the diffusion equation to yield the nonlinear advection term in the Burger's equation. This suggests the following ansatz:

$$f_i^{eq} = A_1 u + B_1 u^2 \qquad i = 1,2 \tag{3.8}$$

$$f_i^{eq} = A_2 u + B_2 u^2 \qquad i = 3,4 \tag{3.9}$$

A multi-scale expansion, the relationship specified by Equation 3.6, and the equation for conservation of particle distribution functions (Equation 3.7), will recover the Burger's equation. The free parameters of the ansatz will be adjusted after the multi-scale analysis such that the Burger's equation is obtained.

Substituting Equations 3.8 and 3.9 into Equation 3.6, the following is obtained:

$$u = \sum_i f_i^{eq}(\bar{x},t) = 2(A_1 + A_2)u + 2(B_1 + B_2)u^2 \tag{3.10}$$

This suggests the following constraints:

$$A_1 + A_2 = \frac{1}{2} \tag{3.11}$$

$$B_2 = -B_1 \tag{3.12}$$

The multi-scale expansion [35] involves the following: expand the distribution functions f_i around the equilibrium distributions f_i^{eq} using ε as the expansion parameter. This is

$$f_i = f_i^{eq} + \varepsilon f_i^{(1)} + O(\varepsilon^2) \tag{3.13}$$

The left-hand side (LHS) of the lattice Boltzmann equation (Equation 3.2) is expanded using a Taylor series expansion. This is

$$\begin{aligned} f_i(\vec{x}+\vec{c}_i\Delta t, t+\Delta t) &= f_i(\vec{x},t) + \Delta t c_{i\alpha}\partial_{x_\alpha} f_i + \Delta t\partial_t f_i \\ &+ \frac{(\Delta t)^2}{2}\left[\partial_t\partial_t f_i + 2c_{i\alpha}\partial_t\partial_{x_\alpha} f_i + c_{i\alpha}c_{i\beta}\partial_{x_\alpha}\partial_{x_\beta} f_i\right] + O(\partial^3 f_i) \end{aligned} \tag{3.14}$$

Equation 3.14 can be simplified to

$$f_i(\vec{x}+\vec{c}_i\Delta t, t+\Delta t) = f_i(\vec{x},t) + (c_{i\alpha}\Delta t)\partial_{x_\alpha} f_i + \Delta t\partial_t f_i + O(\varepsilon^2) \tag{3.15}$$

Comparing the right-hand side (RHS) of Equation 3.15 with the RHS of the lattice Boltzmann equation (Equation 3.2) and using the expansion (Equation 3.13), an approximation of $f_i^{(1)}$ is obtained,

$$\varepsilon f_i^{(1)} = -\frac{1}{\omega}(c_{i\alpha}\Delta t)\partial_{x_\alpha} f_i - \frac{1}{\omega}\Delta t\partial_t f_i + O(\varepsilon^2) \tag{3.16}$$

Wolf-Gladrow [35] adopted the following scaling for deriving the two-dimensional diffusion equation from the LBM:

$$\begin{aligned} \partial_t &\to \varepsilon^2\partial_t^{(2)} \\ \partial_{x_\alpha} &\to \varepsilon\partial_{x_\alpha}^{(1)} \end{aligned} \tag{3.17}$$

Since Burger's equation can be realized by adding a nonlinear convective term to the diffusion equation, the same scaling as above (Equation 3.17) is adopted for deriving the two-dimensional Burger's equation from the LBM.

From the definition of lattice velocities (Equation 3.3)

$$\sum_i \vec{c}_i = 0 \tag{3.18}$$

$$\sum_i c_{i\alpha}c_{i\beta} = 2c^2\delta_{\alpha\beta} \tag{3.19}$$

where $\delta_{\alpha\beta}$ is the Kronecker delta. Substituting Equations 3.14 and 3.17 into the conservation relation given by Equation 3.7, and followed by further simplification using Equations 3.13 and 3.16 results in:

$$0=\sum_i\left[\varepsilon^2\partial_t^{(2)}f_i+c_{i\alpha}\varepsilon\partial_{x_\alpha}^{(1)}f_i+\frac{\Delta t}{2}\varepsilon^2c_{i\alpha}c_{i\beta}\partial_{x_\alpha}^{(1)}\partial_{x_\beta}^{(1)}f_i^{eq}+O\left(\varepsilon^3\right)\right] \qquad (3.20)$$

where the first term on the RHS of the above equation is

$$\sum_i\varepsilon^2\partial_t^{(2)}f_i\to\partial_t u \qquad (3.21)$$

Substituting the expansion (Equation 3.13) into the second term on the RHS of Equation 3.20 and simplifying using Equation 3.16 results in:

$$\sum_i c_{i\alpha}\varepsilon\partial_{x_\alpha}^{(1)}f_i=\varepsilon\sum_i\partial_{x_\alpha}^{(1)}c_{i\alpha}f_i^{eq}-\frac{\Delta t}{\omega}\varepsilon^2\sum_i\partial_{x_\alpha}^{(1)}\partial_{x_\beta}^{(1)}c_{i\alpha}c_{i\beta}f_i^{eq}+O\left(\varepsilon^3\right) \qquad (3.22)$$

for the two-dimensional four-speed lattice with x, y directions

$$\begin{aligned}\varepsilon^2\sum_i\partial_{x_\alpha}^{(1)}\partial_{x_\beta}^{(1)}c_{i\alpha}c_{i\beta}f_i^{eq}&=\varepsilon^2\partial_x^{(1)}\partial_x^{(1)}\left(c_{1x}c_{1x}f_1^{eq}+c_{3x}c_{3x}f_3^{eq}\right)\\&+\varepsilon^2\partial_y^{(1)}\partial_y^{(1)}\left(c_{2y}c_{2y}f_2^{eq}+c_{4y}c_{4y}f_4^{eq}\right)\end{aligned} \qquad (3.23)$$

Substituting the equilibrium distributions (Equations 3.8, 3.9) into the above equation and taking into account the constraints given by Equations 3.11, 3.12 will result in:

$$\varepsilon^2\sum_i\partial_{x_\alpha}^{(1)}\partial_{x_\beta}^{(1)}c_{i\alpha}c_{i\beta}f_i^{eq}=\frac{c^2}{2}\left(\varepsilon^2\partial_x^{(1)}\partial_x^{(1)}u+\varepsilon^2\partial_y^{(1)}\partial_y^{(1)}u\right)\to\frac{c^2}{2}\left(\frac{\partial^2u}{\partial x^2}+\frac{\partial^2u}{\partial y^2}\right) \qquad (3.24)$$

It is now obvious that the nonlinear terms in Burger's equation will be obtained from the first term in the RHS of Equation 3.22. Considering it:

$$\varepsilon\sum_i\partial_{x_\alpha}^{(1)}c_{i\alpha}f_i^{eq}=\varepsilon\partial_x^{(1)}c_{1x}f_1^{eq}+\varepsilon\partial_x^{(1)}c_{3x}f_3^{eq}+\varepsilon\partial_y^{(1)}c_{2y}f_2^{eq}+\varepsilon\partial_y^{(1)}c_{4y}f_4^{eq} \qquad (3.25)$$

Substituting the equilibrium distributions (Equations 8, 9) into the above expression gives us

$$\begin{aligned}&\varepsilon\sum_i \partial^{(1)}_{x_\alpha} c_{i\alpha} f_i^{eq} = c(A_1 - A_2)\varepsilon\partial^{(1)}_x u + c(B_1 - B_2)\varepsilon\partial^{(1)}_x u^2 \\ &+ c(A_1 - A_2)\varepsilon\partial^{(1)}_y u + c(B_1 - B_2)\varepsilon\partial^{(1)}_y u^2\end{aligned} \tag{3.26}$$

This leads to

$$\begin{aligned}&\varepsilon\sum_i \partial^{(1)}_{x_\alpha} c_{i\alpha} f_i^{eq} \to c(A_1 - A_2)\partial_x u + c(B_1 - B_2)\partial_x u^2 \\ &+ c(A_1 - A_2)\partial_y u + c(B_1 - B_2)\partial_y u^2\end{aligned} \tag{3.27}$$

To obtain the nonlinear part of Burger's equation from this, the following constraints are required in the above expression:

$$A_1 - A_2 = 0 \tag{3.28}$$

$$B_1 - B_2 = \frac{1}{2c} \tag{3.29}$$

Solving Equations 3.11, 3.12, 3.28 and 3.29 simultaneously gives

$$A_1 = A_2 = \frac{1}{4}, B_1 = \frac{1}{4c} \text{ and } B_2 = -\frac{1}{4c} \tag{3.30}$$

The equilibrium distributions are therefore:

$$f_i^{eq} = \frac{u}{4} + \frac{u^2}{4c} \qquad i = 1,2 \tag{3.31}$$

$$f_i^{eq} = \frac{u}{4} - \frac{u^2}{4c} \qquad i = 3,4 \tag{3.32}$$

Using Equations 3.24 and 3.27, the second term in the RHS of Equation 3.20 is simplified to

$$\sum_i c_{i\alpha}\varepsilon\partial^{(1)}_{x_\alpha} f_i \rightarrow u\frac{\partial u}{\partial x}+u\frac{\partial u}{\partial y}-\frac{c^2\Delta t}{2\omega}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{3.33}$$

The third term in RHS of Equation 3.20 can be simplified to:

$$\sum_i \frac{(\Delta t)}{2} c_{i\alpha}c_{i\beta}\varepsilon^2\partial^{(1)}_{x_\alpha}\partial^{(1)}_{x_\beta} f_i^{eq} = \frac{c^2\Delta t}{4}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{3.34}$$

Substituting Equations 3.21, 3.33, and 3.34 into Equation 3.20 yields

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+u\frac{\partial u}{\partial y}=\frac{1}{2}c^2\Delta t\left(\frac{1}{\omega}-\frac{1}{2}\right)\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{3.35}$$

When the diffusion coefficient is

$$\mu=\frac{1}{2}c^2\Delta t\left(\frac{1}{\omega}-\frac{1}{2}\right) \tag{3.36}$$

Equation 3.35 becomes the two-dimensional Burger’s equation (Equation 3.5). Based on this, the LBM can be used to numerically simulate the two-dimensional Burger’s equation by specifying the appropriate values for $\Delta x, \Delta t, \text{and}\, \omega$.

3.1.3 Lattice Boltzmann Algorithm

To implement an explicit time-marching lattice Boltzmann solver for the two-dimensional Burger’s equation, the following substeps are executed at each time step:

1. Using the initial *u*, calculate the equilibrium distributions

$$f_i^{eq}=\frac{u}{4}+\frac{u^2}{4c} \qquad i=1,2 \tag{3.37}$$

$$f_i^{eq}=\frac{u}{4}-\frac{u^2}{4c} \qquad i=3,4 \tag{3.38}$$

and set $f_i = f_i^{eq}$ for the first time step.

2. Compute the RHS (also known as the collision substep) of the lattice Boltzmann equation (Equation 3.2) and propagate the result to the nearest neighbor nodes obtaining $f_i\left(\vec{x} + \vec{c}_i \Delta t, t + \Delta t\right)$ (also known as the propagation substep).
3. Update $u(\vec{x},t)$ from the new distributions according to the definition $u(\vec{x},t) = \sum_i f_i(\vec{x},t)$ (Equation 3.6).

Start the next time step with the calculation of new equilibrium distributions using the new $u(\vec{x},t)$ and proceed to Step 2 of the algorithm. Follow this procedure until the final time.

3.1.4 Initial and Boundary Conditions

In fluid dynamics problems, the initial and boundary conditions are specified in terms of the macroscopic variable. To obtain the initial and boundary conditions in terms of the distribution functions, two different approaches can be adopted. One is an inverse mapping between the distribution function and the macroscopic variable (a direct mapping being Equation 3.6). Here, the distribution functions are made equal to the equilibrium distributions (shown as the initial condition above) or to the sum of equilibrium distributions and a non-equilibrium term such as:

$$f_i = f_i^{eq} + f_i^{neq} \tag{3.39}$$

For the four-speed lattice Boltzmann model for the two-dimensional Burger's equation without the $\frac{\partial u}{\partial y}$ term on the LHS:

$$\begin{aligned} f_1^{neq} &= -\frac{\Delta t}{\omega} \cdot \left(\frac{c}{4} \cdot \frac{\partial u}{\partial x} \right) \\ f_3^{neq} &= -\frac{\Delta t}{\omega} \cdot \left(-\frac{c}{4} \cdot \frac{\partial u}{\partial x} \right) \\ f_2^{neq} &= f_4^{neq} = 0 \end{aligned} \tag{3.40}$$

The non-equilibrium distributions were obtained with a multiscale expansion similar to the manner in which Skordos [37] obtained them for the incompressible Navier-Stokes equations. The second approach for obtaining boundary conditions is similar to that of Palmer et al. [38]. For instance, the f_3 distribution function at the right boundary is expressed as:

$$f_3 = u - f_1 - f_2 - f_4 \tag{3.41}$$

3.1.5 Stability and Accuracy of LBM

The numerical solution for LBM is obtained via a time-marching method. The four-speed model of the discrete velocity Boltzmann equation (Equation 3.1) used to simulate the Burger's equation is:

$$\begin{aligned}
\frac{\partial f_1}{\partial t}+c\frac{\partial f_1}{\partial x}&=-\omega\left(f_1-f_1^{eq}\right)\\
\frac{\partial f_2}{\partial t}+c\frac{\partial f_2}{\partial y}&=-\omega\left(f_2-f_2^{eq}\right)\\
\frac{\partial f_3}{\partial t}-c\frac{\partial f_3}{\partial x}&=-\omega\left(f_3-f_3^{eq}\right)\\
\frac{\partial f_4}{\partial t}-c\frac{\partial f_4}{\partial y}&=-\omega\left(f_4-f_4^{eq}\right)
\end{aligned}\tag{3.42}$$

As shown above, the discrete velocity Boltzmann equation consists of first-order hyperbolic PDEs. For a first-order hyperbolic PDE such as the one-dimensional linear advection equation,

$$u_t + cu_x = 0 \tag{3.43}$$

the stability of an upwind and forward Euler numerical discretization (same as the numerical discretization in the LBM) is defined by the Courant-Friedrichs-Lewy (CFL) condition [39]:

$$CFL = c\frac{\Delta t}{\Delta x} \le 1 \tag{3.44}$$

The above condition holds for a one-dimensional linear advection equation. The LBM has four one-dimensional linear advection equations, two in each dimension, with source terms that couple the four equations. In the LBM, advection speed is assumed to be equal to the spatial discretization over the time step, which should satisfy the CFL condition for the advection part of the equations shown below. But computer experiments by this author have shown that the time step is limited to a little over one-half the spatial discretization when solving the Burger's equation. For low-viscosity cases (occurrence of boundary layer), this limitation is pronounced. When viscosity is not too low, the time step size can be bigger. This would mean that the CFL condition is more limited than that given in Equation 3.44. The reason for this is the non-linearity in the RHS of Equation 3.42 that comes about from the equilibrium distributions. That is, local gradients in the solution field influence the time step size. When simulating the two-dimensional diffusion equation [40], there is no limitation on the time step other than the CFL condition (Equation 3.44), because the equilibrium distributions do not contain any non-linear terms.

A linear stability analysis of the Lattice Boltzmann scheme shows that the collision frequency is limited to $0 < \omega < 2$ [41]. As ω approaches 2, the scheme becomes unstable. This is equivalent to the viscosity approaching zero or becoming negative (refer to Equation 3.36). Sterling and Chen [41] have shown the lattice Boltzmann discretization (Equation 3.2) to result in second-order accuracy both in space and time. This is also shown by the results in Section 3.3.1.

3.2 Cache Optimization

To achieve the scalability and speed potential of the lattice Boltzmann technique, the issues of data reusability in cache-based computer architectures must be addressed. This section examines cache optimization for the LBM in both serial and parallel implementations. The lattice Boltzmann algorithm does not approach peak performance for problem sizes in which the data needed to solve each time step does not fit into the cache memory. In these cases, this data must be obtained from the main memory. Access to the main memory is much slower than access to the cache memory. As a result, to obtain the best possible performance on cache-based computer architectures, problem blocking at various levels is required to match the information flow within the algorithm to the machine's memory hierarchy [42].

Cache-based algorithms block large problems into smaller pieces that fit into the cache. The goal is to create small blocks that can be moved into the cache. Once in the cache, the elements in the blocks can be repeatedly used. The original lattice Boltzmann algorithm operates on contiguous elements in the whole of the computational domain (the whole array). A cache-based lattice Boltzmann algorithm operates on subarrays of the whole array of data. The implementation of the LBM cache optimization algorithm is shown for the two-dimensional diffusion equation, $T_t = \mu(T_{xx} + T_{yy})$ since it is easier to represent the LBM pseudocode for this equation. The LBM algorithm for the two-dimensional diffusion equation is different from that of the two-dimensional Burger's equation in the form of the equilibrium distribution functions. For the two-dimensional diffusion equation mentioned above, all four equilibrium distribution functions are defined as $f_i^{eq} = \dfrac{T}{4}$.

The Fortran implementation of the LBM algorithm for a single time step is shown in Figure 3.2. The first line of the code is the time stepping loop; the rest of the code is executed during each time step. The four distribution functions are represented using two-dimensional arrays, `f1(i,j)`, `f2(i,j)`, `f3(i,j)`, and `f4(i,j)`. The index `i` goes from `1` to `nx` horizontally and `j` goes from `1` to `ny` vertically. Within the code for the collision substep, the value of the equilibrium distribution functions, `feq(i,j)`, is replaced with `T(i,j)*0.25`. The next part of the code describes the propagation substep, the implementation of the boundary conditions, and the update of `T(i,j)` based on the newly computed distribution functions `f(i,j)`.

3.2.1 Cache Blocking for the LBM

As grid size increases, the data handled for computation at each time step can become larger than the cache size. As a result, the data cannot stay cache-resident for repeated use. For instance, when using the LBM to solve the two-dimensional diffusion equation, the largest grid size that can be accommodated within an 8 MB-size cache for a square grid is 512×512. Remembering that there are four distribution functions, each requiring a two-dimensional array,

$$4 \text{ arrays} \times 512^2 \text{ elements} \times 8 \text{ bytes per element} = 8\text{MB}. \qquad (3.45)$$

The LBM is well suited to optimize cache utilization because several time steps can be performed separately on a subsection of the given domain. This can be achieved because the collision substep is completely local, and the propagation substep is almost local (between neighboring grid points). A large domain is divided into subsections. This division of the domain into subsections is termed “block division” in this dissertation. The subsections are updated separately in sequence. The first update accesses data from the main memory and therefore is slow, but for the subsequent updates, the subsection data is cache resident.

```
do time_iter = 1, Niter
           : :
         ::
!-- Perform the collision substep; feq has been replaced with T(i,j)/4.
       do j = 1,ny
          do i = 1,nx
             f1(i,j) = f1(i,j) * (1.0d0 - omega)  &
                  + omega * T(i,j)*0.25d0
             f2(i,j) = f2(i,j) * (1.0d0 - omega)  &
                  + omega * T(i,j)*0.25d0
             f3(i,j) = f3(i,j) * (1.0d0 - omega)  &
                  + omega * T(i,j)*0.25d0
             f4(i,j) = f4(i,j) * (1.0d0 - omega)  &
                  + omega * T(i,j)*0.25d0
          enddo
       enddo

!—Perform the propagation substep
       do j = ny,2,-1
          do i = 1, nx
             f2(i,j) = f2(i,j-1)
          enddo
       enddo
       do j = ny,1,-1
          do i = nx,2,-1
             f1(i,j) = f1(i-1,j)
          enddo
       enddo
       do j = 1,ny-1
          do i = nx,1,-1
             f4(i,j) = f4(i,j+1)
          enddo
       enddo
       do j = 1,ny
          do i = 1,nx-1
             f3(i,j) = f3(i+1,j)
          enddo
       enddo
!-- Boundary conditions are implemented.
       do j = 2, ny-1
          f1(1,j) = -(f2(1,j) + f3(1,j) + f4(1,j))
          f3(nx,j) = -(f2(nx,j) + f1(nx,j) + f4(nx,j))
       enddo
       do i = 2, nx-1
          f2(i,1) = -(f1(i,1) + f3(i,1) + f4(i,1))
          f4(i,ny) = -(f1(i,ny) + f3(i,ny) + f2(i,ny))
       enddo

!-- Calculate new temperature distribution (T)
       do j = 1,ny
          do i = 1,nx
             T(i,j) = (f1(i,j) + f2(i,j) + f3(i,j) + f4(i,j))*0.25d0
          enddo
       enddo
```

Figure 3.2. Fortran code used for implementation of a single time step of the non cache-based LBM.

These subsections are horizontal strips of dimension $N_x \times m_1$, where N_x is the number of grid points in the horizontal direction ($N_x = N_y$), and $m_1 = N_y / p_1$, where p_1 is the number of strips and is chosen such that the data size will not exceed cache memory size (Figure 3.3). The assignment of physical direction associated with the first index in an array is arbitrary. In this dissertation, the x-direction (horizontal direction) in the physical domain is represented by the first index of the array and therefore has greater stride-one access. The y-direction (vertical direction) is represented by the second index in the two-dimensional array. The domain is divided into horizontal strips because they give greater stride-one access to the elements in the arrays than if the domain was divided into vertical strips.

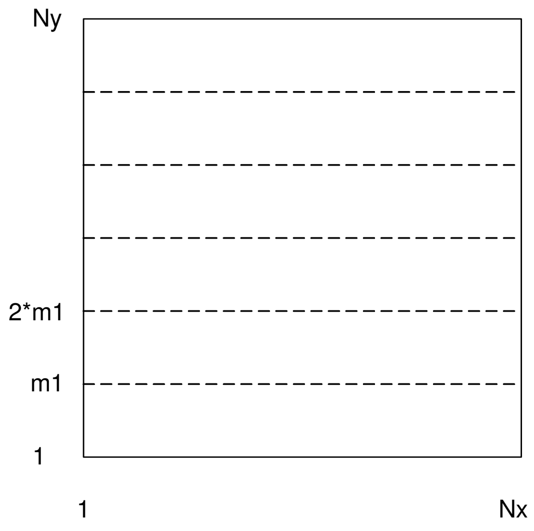


Figure 3.3. Domain decomposition for the cache-blocked serial implementation of the LBM

The time loop is partitioned into tiles (blocks). The number of time steps performed ($tdiv$) on the subsections is the tile size. The choice of tile size significantly impacts the performance of the algorithm. As noted earlier, the propagation substep uses information from the neighboring grid points during each time step. Therefore, data dependencies will arise when the respective subsections are updated separately. Consider a horizontal strip decomposition of the domain (Figure 3.3). The subsections are updated in a bottom–up sequence; i.e., the bottommost subsection is updated $tdiv$ times, and then the next lowest subsection is updated and so on until the topmost subsection is updated $tdiv$ times. During the propagation substep of each time step, f_4 at a node is assigned the value of f_4 at the node directly above it. Therefore, to get the correct value for f_4, each time step must include the same number of nodes directly above the node under consideration as the tile size.

With this block division and update sequence, the propagation of the distribution function f_4 at the upper boundary of a subsection depends on the value of f_4 at the lower boundary of the upper subsection. This data dependency becomes explicit when the subsections are updated separately for $tdiv$ time steps. To handle this data dependency when updating each subsection separately, the following strategy is adopted. In this discussion, the subsection being updated is termed the "current subsection." The collision and propagation substeps are performed on the current subsection $N_x \times m_1$ plus a section of size $N_x \times tdiv$ above the upper boundary of the current subsection. For this reason, the original values of the additional $N_x \times tdiv$ elements (which belong to the upper subsection) included in the computation are saved in a temporary array and then written back (restored to the original values) once the computations with the current subsection are finished.

There are four temporary arrays of size $N_x \times tdiv$, one for each of the distribution functions. The extra computations on the $N_x \times tdiv$ elements are an overhead cost that places a limit on $tdiv$. From experiments performed on an SGI Onyx2™ with 8 MB cache (described later), $tdiv$ was found to have optimum values between 4 and 15 depending on the size of m_1 (e.g., for $m_1 > 80$, $tdiv$ can be as high as 15, and for $m_1 < 26$, $tdiv$ should be 6 or less). The propagation of f_2 also creates explicit data dependencies when subsections are updated from the bottom up. To take care of these data dependencies, the distribution function f_2 at the top boundary of a subsection is saved in a temporary array at each time step for use in computing the next subsection.

The size of the arrays to be used repeatedly in the cache includes the four distribution functions for a subsection, the array containing the saved f_2 values (N_x elements), and the saved $N_x \times tdiv$ elements of the four distribution functions. For practical purposes (and also validated through experiments), it is assumed that the arrays can be kept in only one-half of the cache. When the arrays exceed one-half of the cache, performance declines because of other data competing for space in the cache.

The Fortran implementation of the cache-based serial lattice Boltzmann algorithm for a single time step for the bottommost subsection is shown in Figure 3.4. The time loop is partitioned into blocks of size $tdiv$. Within the time loop, the first section of the code saves the original values of the four distribution functions for the additional $N_x \times tdiv$ elements in the temporary arrays `ff1(i,j)`, `ff2(i,j)`, `ff3(i,j)`, and `ff4(i,j)`, where `i` corresponds to the horizontal direction and `j` corresponds to the vertical direction. The secondary time loop with index `tt` performs `tdiv` time steps on the bottommost subsection. Each time step involves the collision substep and propagation substep. Before `f2(i,m1)` is updated at each time step, it is saved in the array `ftemp2`. After the completion of the secondary time loop, the additional $N_x \times tdiv$ elements that were saved in the temporary arrays are restored back to the arrays holding the values of the four distribution functions. As shown in Figure 3.4, a subsection that fits into the cache is repeatedly used $tdiv$ times. The propagation of f_2 for the next subsection is shown in Figure 3.5. f_2 at the lower boundary of the second subsection (`j=m1+1`) is assigned values from the array `ftemp2` for the corresponding time step. f_2 at the upper boundary of the second subsection is saved in `ftemp2` before being updated.

```
!-- The main time loop is partitioned into blocks/tiles of size tdiv
   do time_iter = 1, Niter, tdiv
           : :
           : :
!-- Save original values of the bottom tdiv rows of subsection 2.
      do j = 1, tdiv
         do i = 1, nx
            ff1(i,j) = f1(i,m1+j)
            ff2(i,j) = f2(i,m1+j)
            ff3(i,j) = f3(i,m1+j)
            ff4(i,j) = f4(i,m1+j)
         enddo
      enddo
!— Perform tdiv time steps on the bottommost subsection
      do tt = 1, tdiv
!-- Perform the collision substep -- feq has been replaced by T(i,j)*0.25
         do j = 1,m1+tdiv
            do i = 1,nx
               f1(i,j) = f1(i,j) * (1.0d0 - omega)+ omega * T(i,j)*0.25d0
               f2(i,j) = f2(i,j) * (1.0d0 - omega)+ omega * T(i,j)*0.25d0
               f3(i,j) = f3(i,j) * (1.0d0 - omega)+ omega * T(i,j)*0.25d0
               f4(i,j) = f4(i,j) * (1.0d0 - omega)+ omega * T(i,j)*0.25d0
            enddo
         enddo
!-- f2 at the top boundary (m1) is saved for each time step (tt).
         ftemp2(1:nx,1,tt) = f2(1:nx,m1)
!-- Perform the propagation substep
         do j = m1+tdiv,1,-1
            do i = nx,2,-1
               f1(i,j) = f1(i-1,j)
             enddo
        enddo
        do j = 1, m1+tdiv
           do i = 1,nx-1
              f3(i,j) = f3(i+1,j)
           enddo
        enddo
        do j = m1+tdiv,2,-1
           do i = 1, nx
              f2(i,j) = f2(i,j-1)
           enddo
        enddo
        do j = 1, m1+tdiv
           do i = nx, 1, -1
              f4(i,j) = f4(i,j+1)
           enddo
        enddo
```

Figure 3.4.   Fortran code used for implementation of a single time step of the cache-based LBM for the bottommost subsection

```
 !-- Boundary conditions are implemented.
         do j = 2, m1+tdiv
            f1(1,j) = -(f2(1,j)+f3(1,j)+f4(1,j))
            f3(nx,j) = -(f2(nx,j)+f1(nx,j)+f4(nx,j))
          enddo
          do i = 2, nx-1
             f2(i,1) = -(f1(i,1)+f3(i,1)+f4(i,1))
          enddo
       enddo
!! end of the time step tile (tt) for 1st subsection
!—write back the original values
        do j = 1, tdiv
           do i = 1, nx
              f1(i,m1+j) = ff1(i,j)
              f2(i,m1+j) = ff2(i,j)
              f3(i,m1+j) = ff3(i,j)
              f4(i,m1+j) = ff4(i,j)
           enddo
        enddo
```

Figure 3.4. continued

```
ftemp2(1:nx,2,tt) = f2(1:nx,2*m1)

do j = 2*m1+tdiv, m1+1+1, -1
     f2(1:nx,j) = f2(1:nx,j-1)
enddo
f2(1:nx,m1+1) = ftemp2(1:nx,1,tt)
```

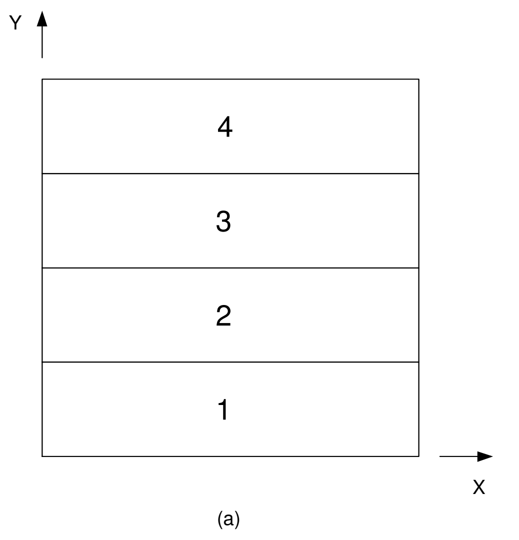
Figure 3.5. Fortran code used for propagation of the distribution function f_2 for the second subsection

3.3 Parallel LBM

In Sections 3.1 and 3.2, the LBM was implemented on a single processor for non cache-based and cache-based versions, respectively. In Section 3.3.1, a parallel version of the LBM is implemented. The parallel version is then optimized for cache utilization in Section 3.3.2. Parallelization of the LBM is carried out using domain decomposition.

3.3.1 Domain decomposition for Parallel Processing

The parallelization is done for a distributed computing environment. The domain is decomposed into subdomains and each subdomain is computed on a separate processor. Type A domain decomposition decomposes the domain into horizontal strips (Figure 3.6a). Type B domain decomposition decomposes the domain into vertical strips (Figure 3.6b). In this discussion, the parallelization is explained in terms of Type A (horizontal strip) decomposition. Type B (vertical strip) decomposition can be performed in a similar manner. The domain is decomposed into p horizontal strips (subdomains), p being the number of processors. The LBM algorithm discussed in Section 3.1.3 is performed on each processor with the data being the subdomain assigned to it. As before, each time step is divided into two substeps, collision and propagation, which are performed on each subdomain. The collision substep is the most compute-intensive. This work is completely performed in-processor with no need for data communication (completely parallel). In the propagation substep, each processor must communicate the outgoing/ingoing particle distributions f_i streaming across the subdomain boundaries. As discussed earlier, the propagation substep requires nearest-neighbor communication.



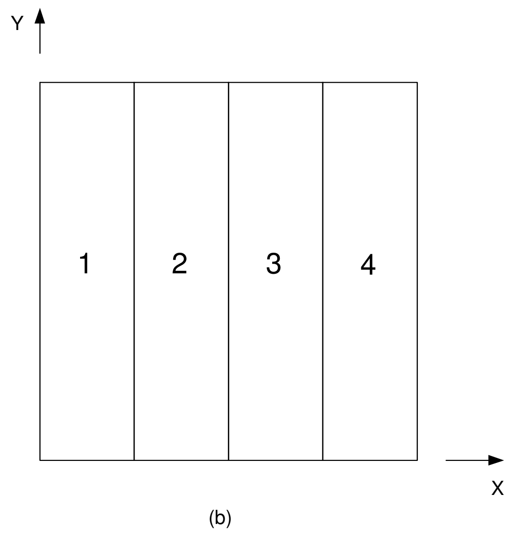


Figure 3.6. Domain decomposition for parallelization: (a) Type A domain decomposition (b) Type B domain decomposition

Although four distribution functions are placed on each node for the square lattice model, only two variables need to be sent across each subdomain boundary. In the case of Type A decomposition, these are f_2 across the top boundary and f_4 across the bottom boundary.

With these changes, the algorithm for parallel implementation of the LBM is:

1. Initially, the equilibrium distributions are calculated (Eqs. 3.31, 3.32) simultaneously on each processor.
2. The RHS of the lattice Boltzmann equation (Equation 3.2) is computed simultaneously on all processors for their respective subdomains—the collision substep.
3. The f_i data at the subdomain boundaries is communicated between processors; the propagation substep is then executed simultaneously on all processors.
4. The new $T(\vec{x},t)$ is calculated in parallel on all processors from the updated distributions f_i.
5. The new $T(\vec{x},t)$ is used to update f_i^{eq} on all processors, and the process is repeated from Step 2 until the final time is reached.

3.3.2 Cache Blocking for the Parallel Lattice Boltzmann Algorithm

The cache-blocking algorithm described in the single processor case (Section 3.2.1) is executed on each processor in the multi-processor case. The data that needs to be blocked for cache optimization is the subdomain belonging to a processor. It is assumed that the subdomain size is greater than the cache memory size. As discussed in Section 3.2.1, the required computations are performed on a subsection of the subdomain for blocks of time steps (block size being $tdiv$). The parallel algorithm given in the above section is implemented in every time step of a block of time steps. It follows that the boundary data is communicated during the start of every time step in a block of time steps. In the serial cache-based LBM, the subsections were created by dividing the domain into horizontal strips (block division) because this gave contiguous and greater stride one-access to the arrays. In the parallel LBM, the type of block division depends on the decomposition used for parallelization.

When the domain is decomposed into vertical strips (each meant for a separate processor-Type B decomposition), the block division used for a subdomain is the same as for the domain in the serial case; i.e., the subdomain on each processor is divided into horizontal strips. In this case, the cache-blocking algorithm used on each processor is the same as in the serial case, as shown in Figure 3.7. When the parallelization involves partitioning the domain into horizontal strips (Type A decomposition), these subdomains should be divided into vertical strips for cache blocking. The subdomains cannot be divided into horizontal strips because then the data dependencies would not be local. For the same reason, for Type B decomposition, the subdomains cannot be divided into vertical strips. For Type B decomposition, the size of a subdomain residing on a processor will be $m \times N_y$, where $m = N_x / p$, p being the number of processors. Similar to the serial case, in the parallel cache-blocking algorithm, a subdomain (on a processor) is divided into horizontal strips of dimension $m \times m_1$, where $m_1 = N_y / p_1$, p_1 being the number of strips. This is chosen to keep each strip within cache memory limits.

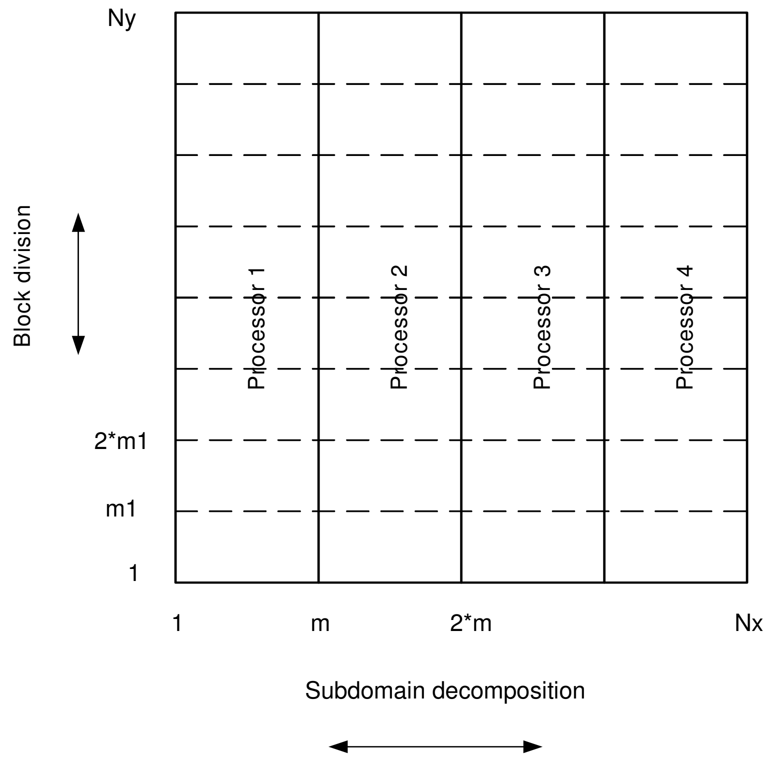


Figure 3.7. Domain decomposition for the cache-blocked parallel LBM

3.3.3 Implementation

The cache optimization study considered uniform grid sizes from N_x = 1200 to 7200. The computations were performed for 50 time steps. The machine used for both serial and parallel cases is an SGI Onyx2™. It is a shared-memory multiprocessing architecture (S2MP). The S2MP architecture uses distributed shared memory (DSM) to enable read and write access into the main memory. This shared memory is accessible to all processors through the NumaLink™ interconnect. The Onyx2™ system has a number of processing nodes linked together by the NumaLink™ interconnect. Each node contains two R12000 microprocessors with 32 KB each of integral instruction and data cache, 8

MB of secondary cache, and 512 MB of main memory per microprocessor. The interconnect bandwidth is 1600 MBps [43]. The program is written in Fortran90 and MPI is used as the message-passing library.

3.3.4 Results

To assess the performance of the cache-blocked LBM for the two-dimensional diffusion equation, the following performance measurements were made:

1. speedup and CPU time obtained through cache optimization of the LBM for single processor and multiple processors,
2. speedup and CPU time obtained through parallelization for both Type A and Type B decomposition of the non-cached LBM, and
3. speedup and CPU time obtained through parallelization for both Type A and Type B decomposition of the cache-optimized LBM.

The number of lattice nodes or grid points was varied from 1200×1200 to 7200×7200. The computations were performed for 50 time steps. In this paper, cache-based speedup is defined as:

$$\text{cache based speedup} = \frac{\text{time for non cache based algorithm}}{\text{time for cache based algorithm}} \qquad (3.46)$$

The accuracy of the results was a function of grid size only, and the results of non cache-based serial, cache-based serial, non cache-based parallel, and cache-based parallel implementations were numerically equivalent.

Table 3.1 shows the results obtained through cache optimization for various grid sizes in the single processor case. As the grid size increases from 1200×1200 to 7200×7200, the number of nodes increases by a factor of 36, from 1.4×10^6 to 52.0×10^6.

Over the same span in grid size, the CPU time for the non-cached serial lattice Boltzmann algorithm increases from 20 s to 880 s, a factor of 44. This occurs because of the additional cost of transferring data between the memory and the CPU. The CPU time for the cache-based version increases by a factor of 57, from 8.1 s to 464 s. The average cache-based speedup for grid sizes from 1200×1200 to 3600×3600 is ~2.4. In contrast, the average cache-based speedup of grid sizes from 4800×4800 to 7200×7200 is ~2.0. The cache-based speedup of the serial lattice Boltzmann algorithm degrades as grid size increases because with increasing grid size, m_1 (number of rows in a subsection) must decrease to ensure that the $m_1 \times N_x$ elements of the subsection stay within the cache size limit. For example, in the 1200×1200 case m_1 is 80, and in the 7200×7200 case m_1 is 15. As discussed earlier, this reduction in m_1 requires a corresponding reduction in the number of time steps that can be completed within the cache, $tdiv$. As $tdiv$ decreases, the number of cache reuses decreases significantly. Hence, the comparative performance of the cache-optimized lattice Boltzmann algorithm decreases as grid size increases.

Table 3.2 shows cache-based speedup for the parallel LBM for Type A and Type B decomposition. For the 1200×1200 cases with four processors, the cache-based speedup is lower because the subdomains nearly fit into the cache, reducing the improvement in performance available from cache optimization. For the 1200×1200 case, when the number of processors is greater than four, the subdomains fit entirely into the cache. For the 2400×2400 case with 24 processors, the subdomains again fit entirely into the cache on each processor. In these cases the impact of cache optimization is negligible. When the number of processors is 12, 16, and 20 for the 2400×2400 grid, the subdomains are larger than the 4 MB cache size limit. However, they still stay within the actual cache size (8 MB). In these cases, there is speedup due to cache optimization but not as much as in larger grids, which do not fit in the cache. In the case of the 3600×3600, 4800×4800, and 6000×6000, the cache-based speedup is lower on four processors. Additionally, it is lower for four and eight processors on the 7200×7200 grid. This occurs for the same reason as the reduced cache-based speedup for the 7200×7200 grid size in the serial case. As the size of the subdomain on each processor increases, the tile size must decrease, reducing the cache reuse and thus limiting the cache-based speedup.

Table 3.1. CPU time and cache-based speedup on a single processor

Nx	CPU time for serial non-cached lattice Boltzmann method (s)	CPU time for serial cached lattice Boltzmann method (s)	Cache-based speedup
1200	20.0	8.1	2.47
2400	83.6	35.3	2.37
3600	190	82.0	2.31
4800	356	175	2.03
6000	586	282	2.07
7200	880	464	1.90

Table 3.2. CPU time and cache-based speedup on multiple processors

Number of Processors	CPU time for non-cached Type A decomposition (s)	CPU time for non-cached Type B decomposition (s)	CPU time for cached Type A decomposition (s)	CPU time for cached Type B decomposition (s)	Type A cache-based speedup	Type B cache-based speedup
NX=1200						
4	5.25	4.75	2.80	1.96	1.88	2.42
8	*	*	1.10	0.82	*	*
12	*	*	0.73	0.55	*	*
16	*	*	0.55	0.41	*	*
20	*	*	0.44	0.34	*	*
24	*	*	0.38	0.30	*	*
NX=2400						
4	33.3	33.3	12.3	8.85	2.71	3.76
8	15.6	15.7	5.95	4.35	2.62	3.61
12	7.76	7.90	3.72	3.12	2.09	2.53
16	5.27	5.16	2.72	2.16	1.94	2.39
20	2.56	2.60	1.95	1.57	1.31	1.66
24	*	*	1.62	1.33	*	*
NX=3600						
4	79.3	76.7	37.0	20.2	2.14	3.80
8	38.3	38.9	13.8	10.0	2.78	3.89
12	26.5	26.1	8.90	6.70	2.98	3.90
16	18.7	19.8	6.62	5.20	2.83	3.81
20	13.5	14.6	5.30	4.10	2.55	3.56
24	11.9	13.0	4.45	3.57	2.67	3.64
NX=4800						
4	137.3	136.6	59.0	42.0	2.33	3.25
8	71.7	69.4	25.2	19.5	2.84	3.56
12	47.4	46.0	16.3	12.0	2.90	3.83
16	34.7	34.9	12.0	9.16	2.89	3.81
20	27.0	28.0	9.56	7.48	2.82	3.74
24	22.3	22.5	8.00	6.35	2.79	3.54
NX=6000						
4	213.3	211.0	104.5	61.0	2.04	3.46
8	106.3	106.9	43.2	28.5	2.46	3.75
12	71.0	72.2	26.5	18.5	2.68	3.90
16	52.7	54.3	19.5	14.0	2.70	3.88
20	42.2	43.6	15.5	11.5	2.72	3.79
24	35.7	37.7	12.8	9.70	2.80	3.89
NX=7200						
4	309.0	306.6	228.0	133.0	1.36	2.30
8	154.0	153.2	87.5	44.5	1.76	3.44
12	102.2	103.3	56.7	27.0	1.80	3.83
16	76.4	77.2	38.6	21.5	1.98	3.59
20	60.8	64.2	23.0	18.4	2.64	3.50
24	52.8	55.5	19.0	13.5	2.78	4.11

The cache-based version of Type B decomposition shows better performance than the cache-based version of Type A decomposition. This is because, to fit the distribution functions into cache, the arrays representing them in Type A decomposition are broken down into blocks in the first index of the array (the first index represents the x-direction or the horizontal direction; refer to Section 3.2.1). The first index is contiguous in memory. Because of this array division, the contiguity in array access is broken and this forces cache misses. With Type B decomposition, the arrays have contiguity in access even when broken into blocks for fitting in cache. This is because they are blocked in the second index, which is not contiguous in memory. Because of this, Type A decomposition shows an average cache-based speedup of 2.6 to 2.8, whereas Type B decomposition shows an average cache-based speedup of 3.6 to 3.8.

Figures 3.8 and 3.9 show parallel speedup for grid sizes 2400×2400 to 7200×7200 for non-cached Type A and Type B domain decomposition, respectively. Parallel speedup is defined as:

$$\text{Parallel speedup} = \frac{\text{CPU time for the serial case}}{\text{CPU time for parallel case}} \tag{3.47}$$

Parallel speedup for the 1200×1200 grid and the 24-processor 2400×2400 grid is not shown because the entire subdomain fits (or nearly fits) into the cache. As shown, without cache blocking parallel speedup for all grids is sublinear except for the 2400×2400 grid when the number of processors is greater than or equal to 12. Additionally, the speedup is not a function of grid size except in the case of the 2400×2400 grid when the number of processors is greater than or equal to 12.

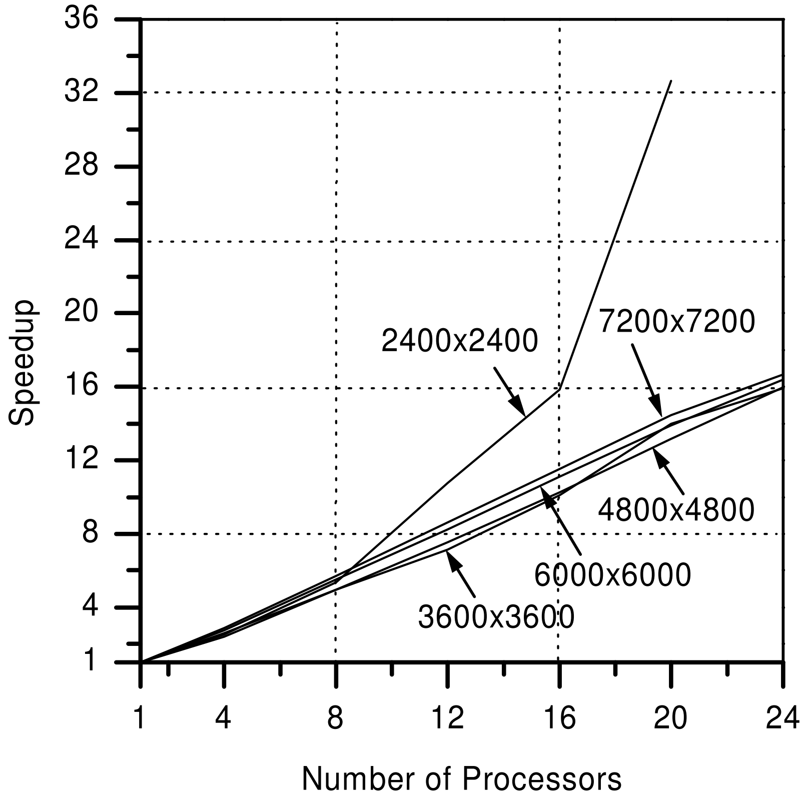


Figure 3.8. Parallel speedup for non-cached Type A domain decomposition

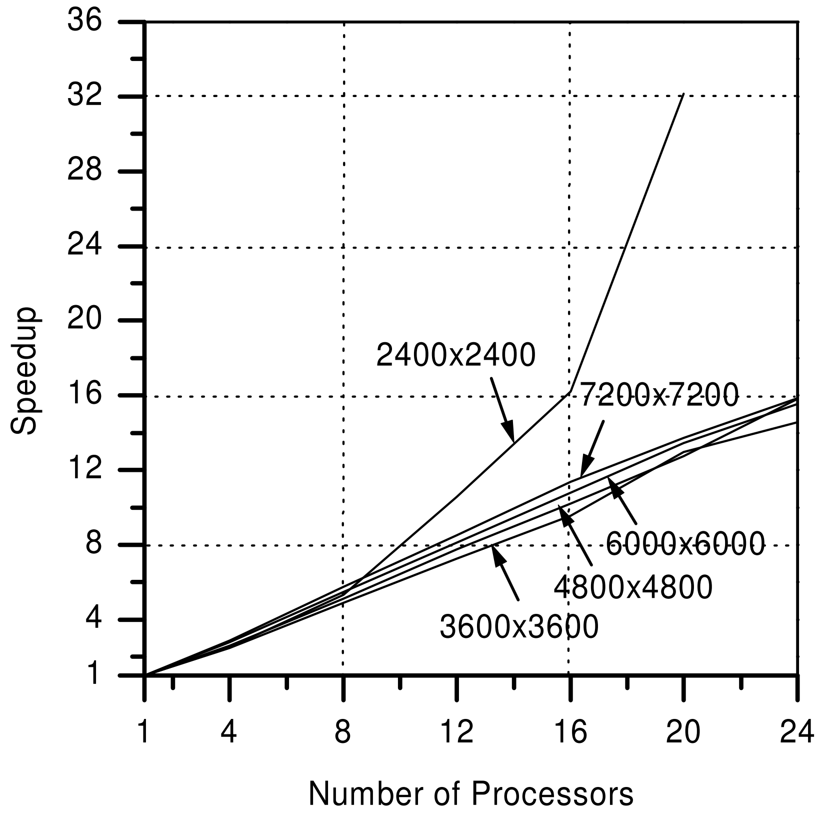


Figure 3.9. Parallel speedup for non-cached Type B domain decomposition

In this case, the subdomains on each processor are not small enough to fit within a 4 MB cache size limit but are within an 8 MB cache size limit. As discussed previously, this allows some cache reuse; thus, significantly improved parallel speedup is observed.

Figures 3.10 and 3.11 show parallel speedup for grid sizes 1200×1200 to 7200×7200 for cached Type A and Type B domain decomposition, respectively. Parallel speedup for the 1200×1200 grid is shown, although cache blocking is not required for more than four processors since the entire subdomain fits into the cache (natural cache blocking). As shown in Figure 3.10, with four processors Type A decomposition is not quite linear. The speedup is a factor of 2 or 3. This occurs because of the initial overhead cost associated with the implementation of the parallel algorithm. Following this, with 4 to 24 processors, the speedup is nearly linear for Type A domain decomposition. The increase in parallel speedup for the 7200×7200 grid between 16 and 24 processors occurs because of the reduced performance of the cached serial case discussed earlier. For both Type A and Type B domain decomposition, the speedup of the 7200×7200 grid is initially lower because the reduced tile size results in fewer cache reuses. For both Type A and Type B domain decomposition, the 3600×3600 grid generally has lower parallel speedup than the rest of the grids. This occurs because it falls between the smaller grids in which the subdomains fit completely into the cache and the larger grids in which the serial cache-based lattice Boltzmann code has reduced tile sizes relative to the parallel cases. The cached Type B domain decomposition has better parallel speedup than the cached Type A domain decomposition because of improved stride-one access. As shown in Figure 3.11, for 1 to 4 processors, speedup for Type B domain decomposition is approximately linear.

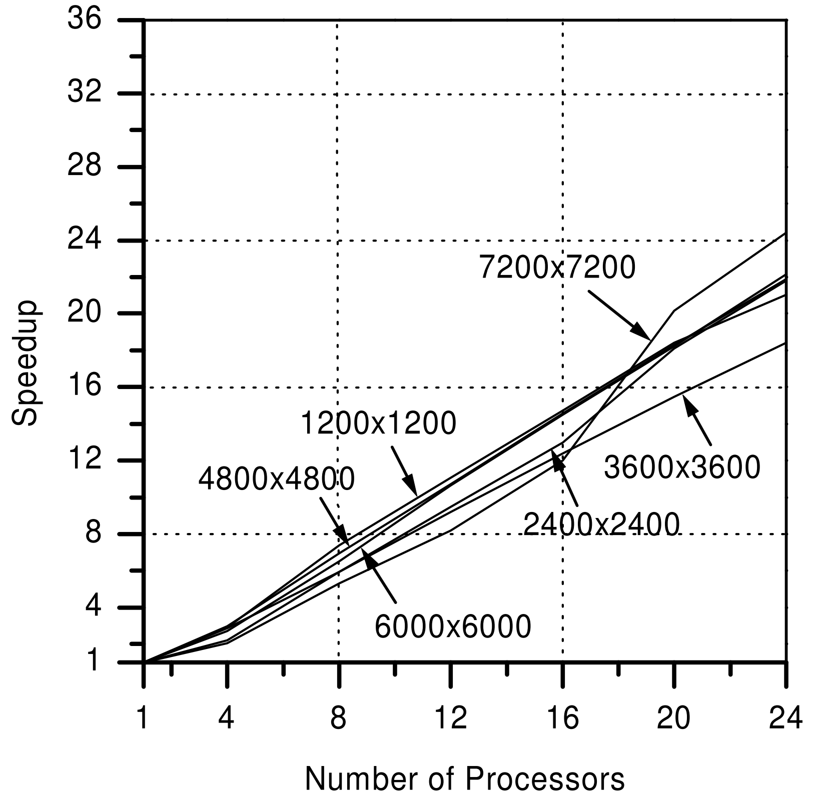


Figure 3.10. Parallel speedup for cached Type A domain decomposition

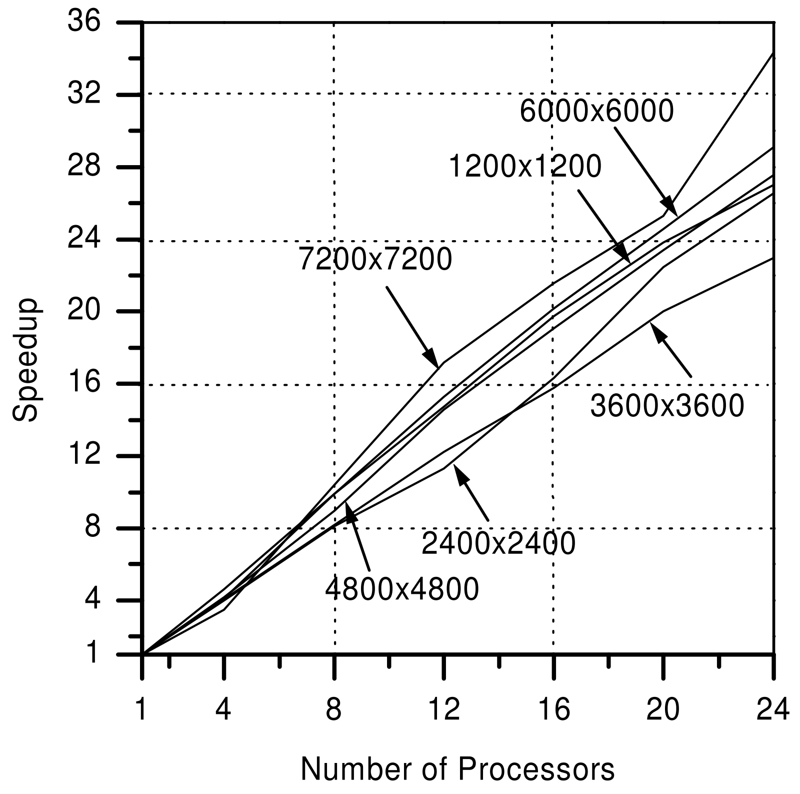


Figure 3.11. Parallel speedup for cached Type B domain decomposition

For greater than four processors, the speedup is generally superlinear with speedup being approximately 28 for 24 processors. This superlinear behavior occurs in the larger grids because the larger tile sizes permitted with the smaller subdomains are able to more than compensate for the additional overhead associated with an increasing number of processors. As in the Type A decomposition, the jump in the speedup for the 7200×7200 grid between 20 and 24 processors occurs because of the reduced performance of the cached serial 7200×7200 case.

3.3.5 General Conclusions about the Cache Optimization

Speedups due to cache optimization were found to be 2.0 to 2.4 for the serial case and 3.6 to 3.8 for the parallel case in which the domain decomposition was optimized for stride-one access (Type B decomposition). Additionally, the cache-optimized LBM in which the domain decomposition was optimized for stride-one access displayed superlinear scalability on all problem sizes as the number of processors was increased. The implementation of the basic lattice Boltzmann algorithm is the same for solving any partial differential equation (PDE) on a uniform grid. The variables that change with each case are the equilibrium distribution functions and the number of speeds (distribution functions) associated with a node on the lattice.

3.4 Comparison of Lattice Boltzmann and Traditional Methods

To evaluate the LBM, it is compared to traditional finite difference methods with regard to accuracy and compute time (on both serial and parallel machines). Both the LBM and traditional methods are compared using the two-dimensional Burger’s equation. Traditional explicit methods that solve convection-diffusion equations suffer from a severely restricted time step on account of both convection and diffusion-related stability criteria [36]. Therefore, their computational performance is not on par with traditional implicit methods. The superiority of the alternating direction implicit (ADI) scheme over traditional explicit schemes such as the DuFort-Frankel scheme for solving the unsteady Burger's equation has already been demonstrated [44]. For this reason, the LBM is compared to the ADI scheme. The LBM does not have as stringent a limitation on time step size as the traditional explicit methods because LBM solves an advection PDE without any diffusion terms (although nonlinear source terms are present). Both implicit finite difference methods and the LBM have a similar time step size limitation when solving unsteady flow problems. When solving steady flow problems with a time marching method, traditional implicit finite difference methods such as the ADI scheme outperform the LBM for low Reynolds number cases as they can take comparatively large time steps. Both unsteady and steady flow cases have been tested in this study. The results for the unsteady Burger's equation will be presented first, followed by the results for the steady two-dimensional Burger's equation.

3.4.1 Test Problem

The two-dimensional Burger's equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+u\frac{\partial u}{\partial y}=\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{3.48}$$

is solved on a square computational domain

$$(0,0)\le(x,y)\le(1,1) \tag{3.49}$$

The exact solution is specified by the following equation:

$$u(x, y, t) = \frac{1}{1 + e^{\frac{(x+y-t)}{2\mu}}} \tag{3.50}$$

The above equation at $t = 0$ specifies the initial condition. The boundary conditions are also specified by the above equation and change with time. With this particular case, the solution has a severe gradient close to the boundary (near left-bottom corner of the computational domain; i.e., near (0,0)) for low-diffusion coefficients or high Reynolds numbers $(\mathrm{Re} = 1/\mu)$.

3.4.2 ADI Method

The ADI method for the two-dimensional Burger's equation [36] is a time-splitting scheme consisting of two substeps in a single time step. Consider an $N \times N$ square grid: for the first half time step N independent tridiagonal equations systems, each system containing N unknowns in the horizontal direction, are solved using Gaussian elimination. The second half time step solves for unknowns in the vertical direction. The ADI scheme has second-order accuracy in both space and time [36]. Time step size needs to be taken such that the truncation error will not be dominated by the temporal component and so that accuracy will not decrease with every time step.

To implement an ADI solver for the two-dimensional Burger's equation, the following substeps are implemented at each time step. The computations are performed on a grid of size NI × NJ with NI = NJ (uniform grid).

- Compute $u_{i,j}^{n}$ according to the given initial condition
- j = 1, 2, …, NJ (vertical direction)

- i = 1, 2, …, NI (horizontal direction)
- Compute the tridiagonal coefficient matrix (NI × NI) for each j ($j = 1,..., NJ$) in grid 1 (Figure 3.12a), and solve for the NI unknowns (includes boundary data) using forward and backward elimination obtaining $u_{*,j}^{n+1/2}, j = 1,..., NJ$.
- Switch to grid 2 (Figure 3.12b), and compute the tridiagonal coefficient matrix (NJ × NJ) for each i ($i = 1,..., NI$) and solve for the NJ unknowns (includes boundary data) using forward and backward elimination obtaining $u_{i,*}^{n+1}, i = 1,..., NI$
- Replace $u^n = u^{n+1}$

  Repeat the above steps in the time loop until final time is reached.

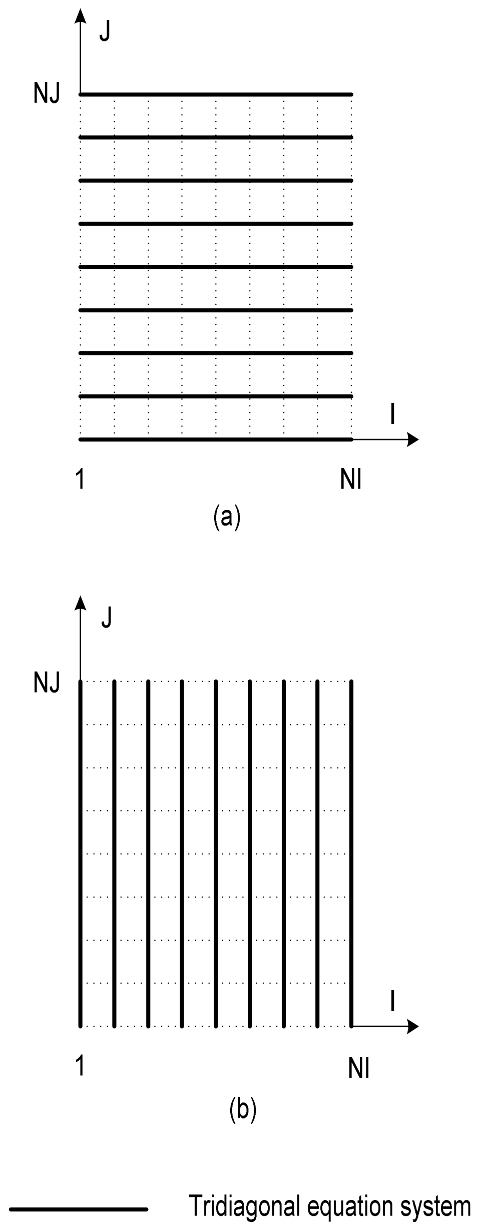


Figure 3.12. The Step 1 and Step 2 grids

3.4.3 Parallel ADI Method

In Section 3.4.2, the ADI method was implemented on a single processor. In this section, a parallel version of the ADI method is implemented for distributed memory systems. Parallelization of the ADI method is carried out using domain decomposition [45]. The domain decomposition when four processors are utilized is shown in Figure 3.13.

The domain or grid size is NI × NI, with NI chosen to be divisible by the number of processors *p*. With the given domain decomposition, the NI number of independent linear equation systems are divided into *p* groups and the tridiagonal matrix from each linear equation system is split vertically into *p* equal parts. Figure 3.13 shows the subdomains associated with each processor. To solve for the unknowns, the processor holding the topmost part of a tridiagonal matrix starts the forward elimination, and upon completion, passes on the last equation or last row of the augmented matrix (coefficient matrix + right-hand side vector) to the succeeding processor. The succeeding processor continues the forward elimination. With the results of forward elimination obtained (an upper triangular matrix), backward elimination for each tridiagonal matrix starts from the processor holding the bottommost part of the tridiagonal matrix. Here, too, once a processor finishes the backward elimination in the subdomain allotted to it, it passes the last unknown variable computed to its preceding processor to continue the backward elimination process. With this domain decomposition, each processor will stay busy eliminating different parts in different groups of equation systems (Figure 3.14).

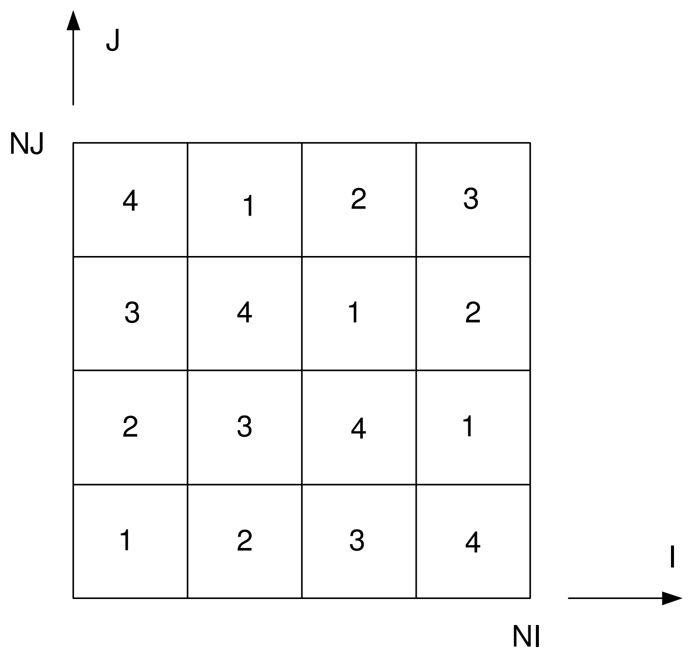


Figure 3.13 Domain decomposition for parallel processing

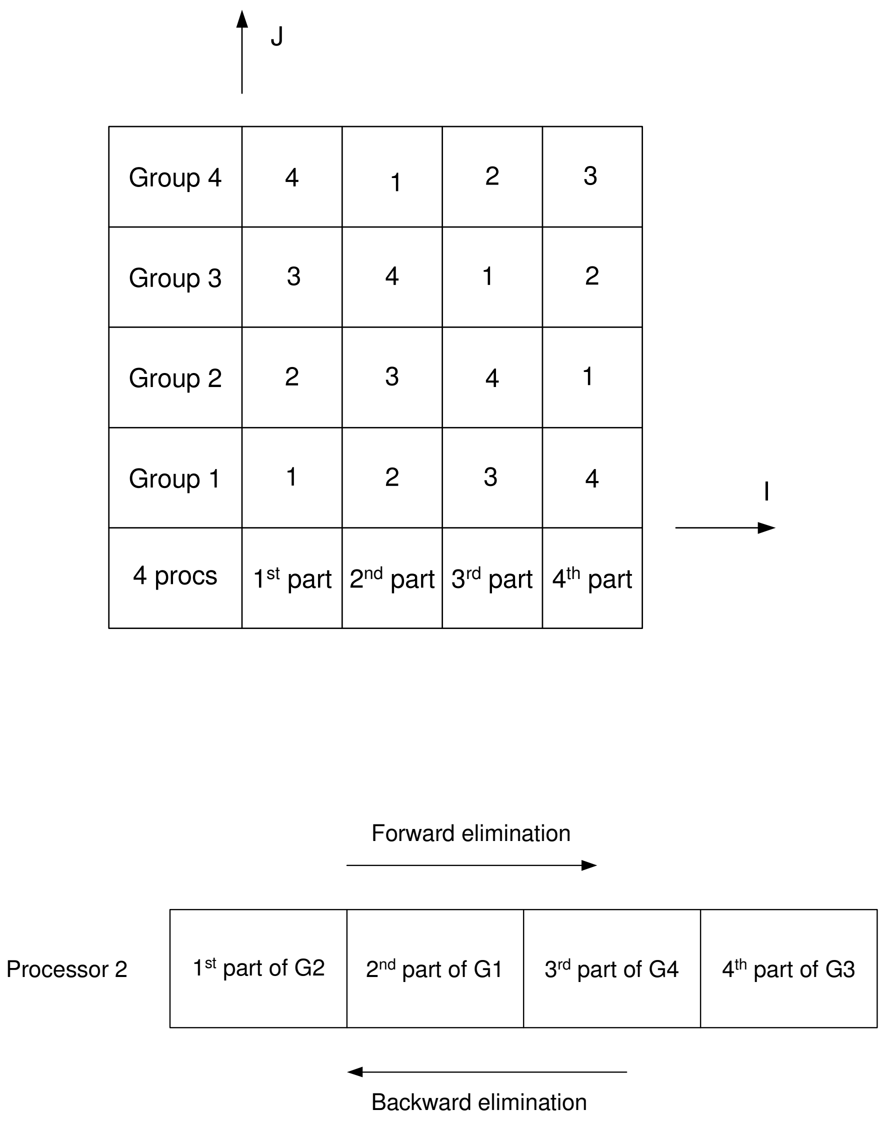


Figure 3.14. Parallel forward and backward elimination

After the first half step (Step 1), new data is available to form the equations for the second half step (Step 2). The computation for the second half step proceeds in the same way. With this division of the domain, only local communication between the left-right (first half step) and top-bottom (second half step) neighbors is required. Communication is required when a processor receives the last equation from the preceding processor or when it sends its last equation to the succeeding processor during forward elimination and when it sends or receives the computed unknown variable with the neighboring processors. Also, communication is required at the beginning of each step for setting up the tridiagonal coefficient matrices for the equation systems at the boundaries of the subdomains (ghost regions).

3.4.4 ADI Algorithm and Cache Optimization

Using the ADI scheme, the largest square grid size that can be accommodated within an 8 MB size cache is around 700×700. The ADI method is not suited for cache optimization because of the global or non-local nature of the scheme. During each time step, tridiagonal matrices are inverted using elimination along the length of the whole domain. This precludes updating any single subsection of the domain separately, which means data cannot be called from the cache repeatedly. For the parallel ADI scheme, cache optimization can occur for the 1200×1200 grid size, since data involved in computation fits into the cache when the number of processors is greater than 12.

3.4.5 Results

To test the computational efficiency, performance, and accuracy of the LBM versus the ADI method, the following measurements are considered:

- Accuracy of both methods for various Reynolds numbers
- Speedup obtained through parallelization of both methods
- Ratio of compute times of the cache-optimized LBM and the ADI method on both single and multiple processors

The results were computed on an SGI Onyx2^{TM}, which is a shared memory multiprocessing architecture with 24 processors, an 8 MB cache, and 512 MB of main memory per processor. MPI was used as the message-passing library.

3.4.5.1 Accuracy

Accuracy has been measured using both L_2-norm error and $L_{\inf}$ or supremum-norm error. These errors are calculated using the following formulas:

$$L_2 - Norm = \frac{\sqrt{\sum_{i=1}^{i\max} \left(\vec{u} - \vec{u}_{exact}\right)^2}}{\sqrt{i\max}} \qquad (3.51)$$

$$L_{\inf} - Norm = \max\left(\vec{u}_i - \vec{u}_{exact,i}\right) \qquad i = 1 \text{ to } i\max \qquad (3.52)$$

where $i\max$ is the total number of grid points. Accuracy is measured for results computed at time = 0.216. The maximum value in the solution at time = 0.216 is unity. The time step size for both methods was chosen to prevent temporal errors from dominating the overall error. For the LBM, the time step size meets the stability criteria (as specified in Section 3.1.5) in all cases. The ADI method requires the same time step size as the LBM to maintain accuracy.

Table 3.3 shows that the error increases progressively with an increasing Reynolds number (inverse of diffusion coefficient) for both the lattice Boltzmann and ADI methods. This is because the solution gradient becomes larger with an increasing Reynolds number. The error for the ADI scheme is slightly lower than that for the LBM. The errors for both methods appear to be converging to the same value as the viscosity decreases. Another pattern that can be observed in Table 3.3 is that the time step size required to maintain accuracy decreases with a decreasing Reynolds number. The LBM uses first-order upwind differencing for the convection term of Equation 3.1, which should result in first-order accuracy. However, second-order accuracy is obtained since the first-order terms in the truncation error have been included as negative viscosity into the overall viscosity (or diffusion coefficient) [41]. The overall order of accuracy estimated using Ferziger's [46] formula was around 1.95 for the numerical solution of both methods.

3.4.5.2 Parallel Speedup

To assess the performance of the parallel LBM and parallel ADI methods, the parallel speedup was measured. The number of grid points varied from 1200×1200 to 7200×7200. The computations were performed for 216 time steps. The parallel speedup is defined as:

$$Parallel\ Speedup = \frac{CPU\ time\ for\ serial\ case}{CPU\ time\ for\ parallel\ case} \qquad (3.53)$$

Table 3.3. L2-norm error and L-infinity or supremum norm error for LB and ADI methods at different viscosities and grid sizes at time = 0.216

Reynolds number	**Final time**				
50	**0.216**				
Grid Size	**Time step**	**LB_L2_error**	**ADI_L2_error**	**LB_sup_error**	**ADI_sup_error**
1200	0.00004	5.93E-05	1.30E-05	5.05E-04	1.09E-04
2400	0.00001	1.48E-05	3.26E-06	1.26E-04	2.73E-05
Reynolds number	**Final time**				
500	**0.216**				
Grid Size	**Time step**	**LB_L2_error**	**ADI_L2_error**	**LB_sup_error**	**ADI_sup_error**
1200	0.0001	2.34E-04	1.86E-04	5.26E-03	4.17E-03
2400	0.00004	1.44E-04	7.48E-05	3.23E-03	1.68E-03
3600	0.00002	8.04E-05	3.75E-05	1.81E-03	8.43E-04
4800	0.00001	3.59E-05	1.88E-05	8.05E-04	4.21E-04
Reynolds number	**Final time**				
5000	**0.216**				
Grid Size	**Time step**	**LB_L2_error**	**ADI_L2_error**	**LB_sup_error**	**ADI_sup_error**
2400	0.0002	1.36E-03	1.07E-03	8.89E-02	6.74E-02
3600	0.0001	7.36E-04	5.77E-04	4.87E-02	3.70E-02
4800	0.00006	4.62E-04	3.57E-04	3.12E-02	2.35E-02
6000	0.00004	3.17E-04	2.41E-04	2.14E-02	1.62E-02
7200	0.00002	1.23E-04	1.23E-04	8.34E-03	8.24E-03
Reynolds number	**Final time**				
50,000	**0.216**				
Grid Size	**Time step**	**LB_L2_error**	**ADI_L2_error**	**LB_sup_error**	**ADI_sup_error**
3600	0.0001	3.40E-03	2.61E-03	0.37	0.31
4800	0.00008	2.17E-03	1.79E-03	0.23	0.213
6000	0.00006	1.36E-03	1.05E-03	0.187	0.138
7200	0.00004	1.06E-03	7.50E-04	0.14	0.1

Figure 3.15 shows the parallel speedup for the ADI method for grid sizes 1200×1200 to 7200×7200 respectively. The parallel ADI method approaches linear speedup for the 1200×1200 grid size, when the number of processors is greater than 12 and the grid size for all cases is 7200×7200. As mentioned in Section 3.4.4, when grid size is 1200×1200 and the number of processors is greater than 12, the data on the processors fits into cache. Hence, better speedup is obtained. With regard to the 7200×7200 case, speedup is good because of poor performance of the single processor case. Figure 3.16 shows the parallel speedup of the cache-optimized LBM. These speedups are super-linear in almost all cases. The 1200×1200 case is an exception because there is not much computation to be performed when number of processors increases.

3.4.5.3 Relative Speed

The compute time comparison for both the LBM and the ADI method is based on the time taken to compute 216 time steps. Such a comparison is possible since both methods obtain best possible accuracy with the same time step size (i.e., they take the same number of time steps to reach the final time). Figure 3.17 shows the relative speed of the cache-optimized LBM versus the ADI scheme. The LBM computation time is less than the ADI computation time by a factor 8 to 8.5 for grids from 2400×2400 onwards when the number of processors is greater than or equal to 8. For the single-processor case, the relative speed varied from 4 to 6 for grid sizes from 1200×1200 to 6000×6000. The 1200×1200 grid proved to be an exception when the number of processors is greater than or equal to 12 because, as mentioned in Section 3.4.4, the subdomains in the parallel

ADI scheme could be accessed from the cache. Hence, the ADI method performed better than usual.

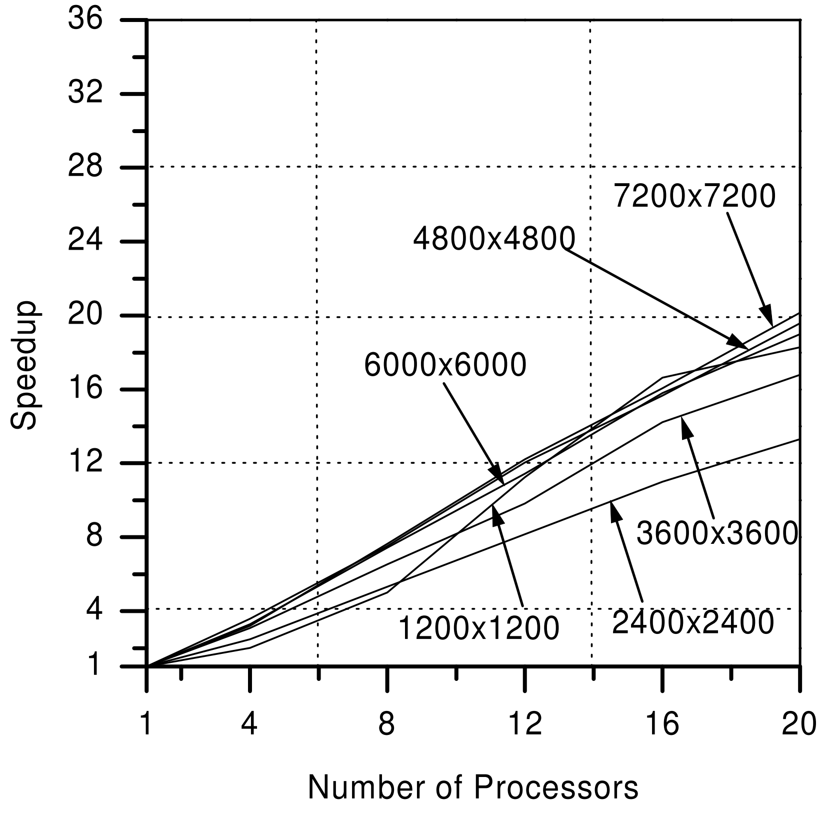


Figure 3.15. Parallel speedup of the ADI method

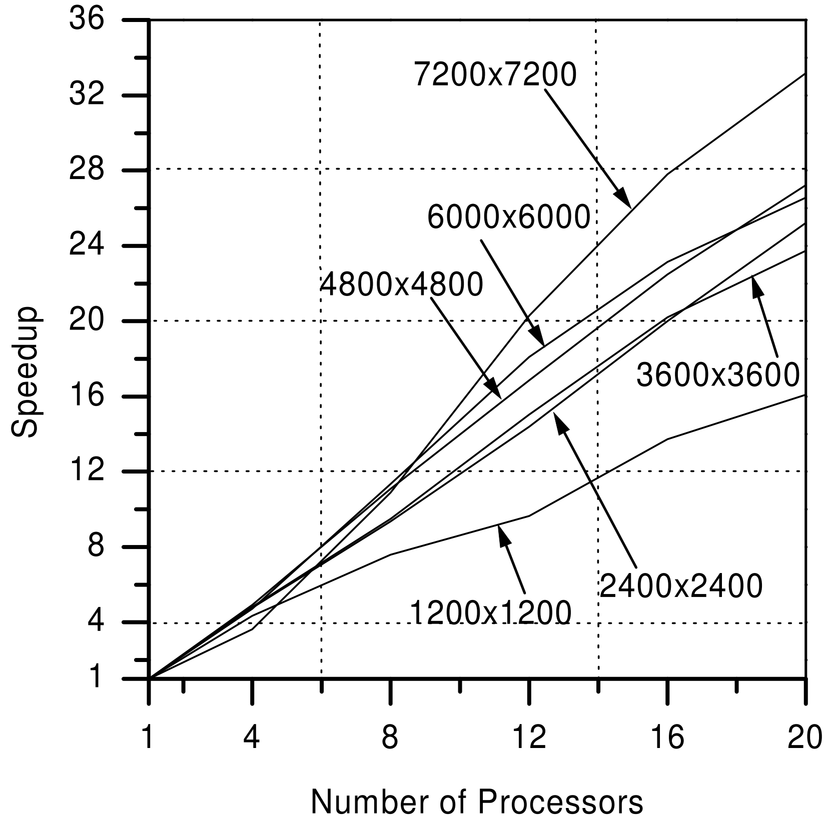


Figure 3.16. Parallel speedup LBM

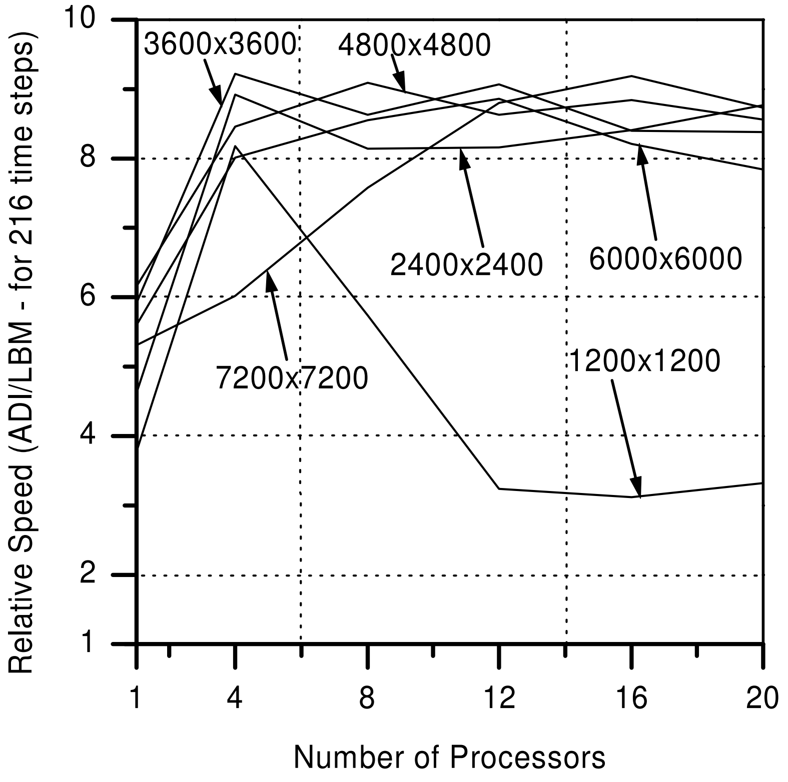


Figure 3.17. Relative speed of LBM versus the ADI method

3.4.6 Conclusions from the Comparison Study for Unsteady Burger's Equation

The LBM takes significantly less compute time (by a factor 8) for the unsteady simulation when it can take the same time step as traditional finite difference methods and not violate any stability criteria. It is expected that the LBM will perform far better for time-marching Navier-Stokes simulations because conventional methods are required to solve a Poisson equation at every time step for computing pressure, whereas the LBM can explicitly calculate pressure from the density (sum of distribution functions).

However, to solve the Navier-Stokes equations, the LBM requires nine distribution functions [35] at each grid point, which would mean greater memory requirements.

3.4.7 Test Case for a Steady Problem

The two-dimensional Burger's equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{3.54}$$

is solved on a square computational domain

$$(0,0)\le(x,y)\le(1,1) \tag{3.55}$$

with the following initial and boundary conditions:

$$u(x,y,t)=1-\left(\frac{2}{3}\right)\times x \text{ at } t=0 \tag{3.56}$$

$$u(x,y,t)=\frac{\left(2-\exp\left(\frac{x-1}{\mu}\right)\right)}{\left(2+\exp\left(\frac{x-1}{\mu}\right)\right)} \text{ at } y=0, y=1 \tag{3.57}$$

The analytical solution to the two-dimensional steady Burger's equation with the above boundary conditions is

$$u(x,y,t)=\frac{\left(2-\exp\left(\frac{x-1}{\mu}\right)\right)}{\left(2+\exp\left(\frac{x-1}{\mu}\right)\right)} \tag{3.58}$$

When the diffusion coefficient μ is small (< 0.002), a steep boundary layer is formed along the right boundary.

3.4.8 Relative Performance of the LBM and the ADI Method

To evaluate the relative performance of the LBM and the ADI method, their accuracy and compute times are compared for a Reynolds number of 5000. The results were computed on an SGI Onyx2^{TM}, which is a multiprocessing architecture with 24 processors, an 8 MB cache, and 512 MB of main memory per processor. MPI was used as the message-passing library and Fortran 90 was the programming language.

3.4.8.1 Accuracy

The numerical solution is compared with the known analytical solution and, therefore, accuracy has been measured using the L_2-norm error. The steady-state solution was obtained through marching the equations of both methods in time until convergence. The convergence criteria utilized with both methods would allow them to converge when the absolute difference between the current time iteration's L2-norm error and the previous time iteration's L2-norm error is less than 10e-6. For the LBM on a 4800×4800 grid, absolute difference was increased to 10e-7 to obtain better accuracy. From Table 3.4, it can be observed that the LBM has slightly better accuracy than the ADI method. Increasing the grid size reduces the error in both cases. The increasing grid size or decreasing grid spacing causes the LBM time step size to go down to keep the scheme numerically stable. For a Reynolds number of 5000, the ADI time step size is 0.004 for all grid sizes. Taking bigger time steps than 0.004 will not allow the ADI method to converge or would require many more time iterations to converge.

3.4.8.2 Parallel Speedup

To assess the parallel performance of both methods, the parallel speedup was measured. Here, the parallel speedup is defined as:

$$Parallel\ Speedup = \frac{CPU\ time\ for\ 4\ processor\ case}{CPU\ time\ for > 4\ processor\ case} \tag{3.59}$$

Table 3.5 shows the parallel speedup for the LBM and the ADI method for grid sizes 2400×2400 to 6000×6000, respectively. The parallel speedups of both methods are linear to superlinear (for bigger grids), thus demonstrating the efficiency of the parallel implementations. The superlinear speedup is achieved since the 4-processor case performs poorly when data size increases to the level of the processor memory size.

3.4.8.3 Relative Speed

The relative speed is defined as the ratio of the computation times of the two methods. Table 3.5 compares the computation times of the LBM with those of the ADI method and Figure 3.18 represents the relative speed graphically. ADI computation time is less than the LBM computation time by a factor 1.7 to 4.8 for grids ranging from 2400×2400 to 6000×6000. This is because the ADI scheme has a time step of 0.004 for all grid sizes, whereas the LBM's time step decreases with increasing grid size (shown in Table 2). Therefore, the LBM takes more time iterations to converge. For grid sizes 2400×2400 to 6000×6000, the number of time iterations taken by the LBM is 6 to 30 times the number of iterations taken by the ADI scheme. This is reflected in the results for relative speed.

3.4.9 Conclusion

From the above comparison, it is obvious that the ADI method outperforms the LBM for solving steady state problems. The LBM performs poorly for steady state problems mainly because it is an explicit method. This leads to small time steps and slower convergence since information transfer across the whole domain does not occur for a single time step.

Table 3.4. L2-norm error for Reynolds number 5000

Reynolds number				
5000				
Grid Size	**LBM time step**	**LBM_L2_error**	**ADI time step**	**ADI_L2_error**
2400x2400	0.0003	4.29E-04	0.004	2.83E-03
3600x3600	0.0002	1.98E-04	0.004	1.08E-03
4800x4800	0.00015	1.17E-04	0.004	5.65E-04
6000x6000	0.0001	6.39E-05	0.004	3.53E-04

Table 3.5. Parallel speedup for the LBM and the ADI method

CPUs	LBM time step	LBM CPU time	LBM speedup	ADI time step	ADI CPU time	ADI speedup
2400x2400	**0.0003**			**0.004**		
4		650.67	1.00		385.47	1.00
8		314.24	2.07		186.46	2.07
12		218.00	2.98		121.82	3.16
16		157.10	4.14		91.20	4.23
20		128.61	5.06		76.15	5.06
3600x3600	**0.0002**			**0.004**		
4		2201.60	1.00		892.07	1.00
8		1088.84	2.02		427.40	2.09
12		714.30	3.08		274.75	3.25
16		544.75	4.04		211.91	4.21
20		434.09	5.07		163.84	5.44
4800x4800	**0.00015**			**0.004**		
4		5525.38	1.00		1660.33	1.00
8		2640.85	2.09		784.78	2.12
12		1688.02	3.27		507.03	3.27
16		1270.91	4.35		372.23	4.46
20		1004.00	5.50		308.76	5.38
6000x6000	**0.0001**			**0.004**		
4		14799.98	1.00		3005.25	1.00
8		5881.04	2.52		1255.62	2.39
12		3990.84	3.71		806.91	3.72
16		2966.75	4.99		618.13	4.86
20		2325.58	6.36		493.52	6.09

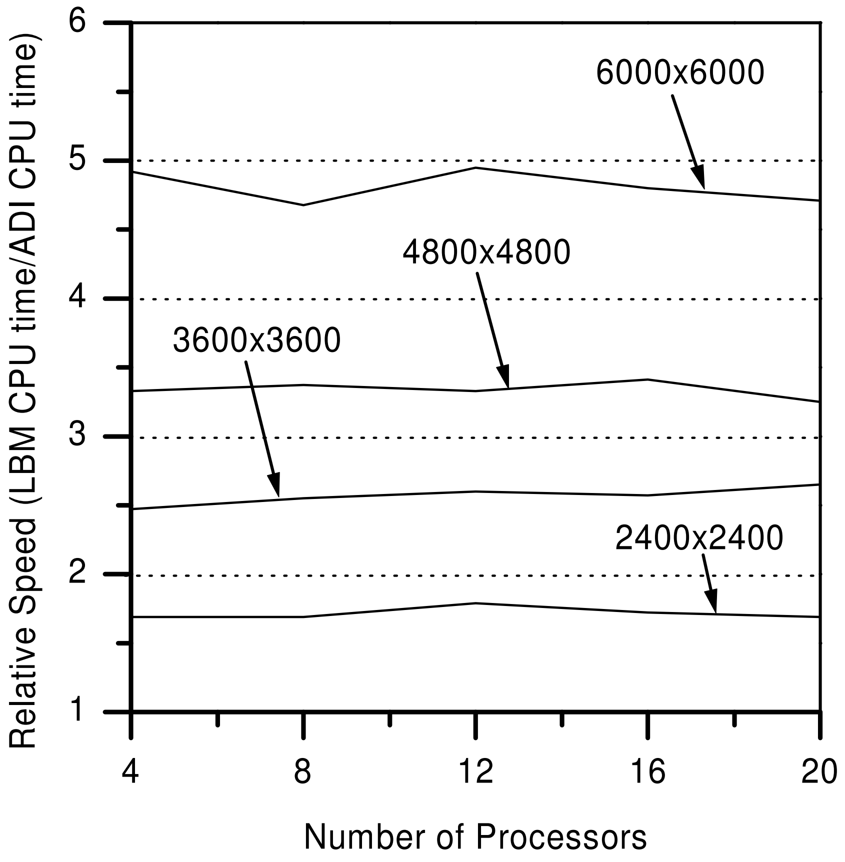


Figure 3.18. Relative speed

CHAPTER 4.

IMPROVING THE PERFORMANCE OF THE LATTICE BOLTZMANN METHOD FOR STEADY FLOW SIMULATION

When solving unsteady flow problems, both explicit and implicit time discretization methods have a similar time step size limitation to maintain accuracy. This contributes to the computational superiority of the lattice Boltzmann method (LBM) over traditional methods for simulating unsteady flow fields since the LBM is an explicit method and does not have to solve systems of simultaneous algebraic equations at each time step, unlike the implicit finite difference methods. When solving steady flow problems with a time marching method, traditional implicit finite difference methods such as the alternating direction implicit (ADI) scheme outperform the LBM as they can take comparatively large time steps. The explicit nature of the lattice Boltzmann discretization limits the time step size, which is also limited by the Courant-Friedrichs-Lewy (CFL) condition and local gradients in the solution, the latter limitation being more extreme. Therefore, this chapter describes:

1. A new explicit discretization for the LBM that can perform simulations with larger time step sizes
2. A coupled LBM-ADI scheme to solve the time-independent, two-dimensional Burger's equation

4.1 Improved LBM

In order to improve the time step size over the limits specified in the previous section, a new explicit discretization will be employed on the discrete velocity Boltzmann equation. In this discussion, the advection speed will not be constrained to equal space discretization over time discretization. Using the explicit Euler/upwind discretization as before (Chapter 3), the lattice Boltzmann equation would now be:

$$f_i\left(\vec{x}+\Delta\vec{x},t+\Delta t\right)=\left(1-CFL\right)\cdot f_i\left(\vec{x}+\Delta\vec{x},t\right)+\left(CFL-\omega\right)\cdot f_i\left(\vec{x},t\right)+\omega\cdot f_i^{eq}\left(\vec{x},t\right) \quad (4.1)$$

It is obvious that the computer implementation of this scheme would not be as efficient as that of the original lattice Boltzmann scheme where CFL = 1. The CFL condition (Equation 3.44) holds for the above equation as well. In the derivation of the CFL condition, it was assumed that the finite difference stencil should cover the point B (Figure 4.1) as well as the domain of dependence. Zhou et al. [47] proposed a method for the one-dimensional advection equation where the above assumptions were discarded and it was perceived that it was not important for the stencil to cover B, but for it to cover B's domain of dependence (X). Determining the relative position of the stencil with respect to B becomes the important issue.

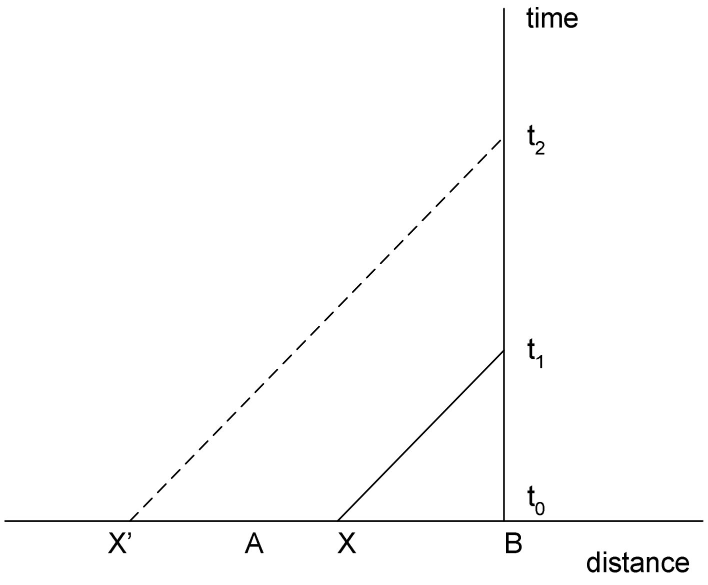


Figure 4.1. Domain of dependence

It is important for the finite difference stencil to cover the point X, representing the domain of dependence of B for a given time step size, to retain stability for the discretized equations. To predict the relative location of the finite difference stencil with respect to B, the following line of reasoning is used. In the case of the linear advection equation, the solution translates with advection speed c without a change in shape. Therefore, if X is the domain of dependence of B for a time step Δt, the distance between X and B in terms of grid units would be $c \cdot \Delta t / \Delta x$, which is the CFL number. Considering that X would be between two grid points $j-1$ and j, and to predict the grid point j for a given CFL number, the following steps are followed [47]:

$$j = k - trunc(CFL), \quad frac = CFL - trunc(CFL) \tag{4.2}$$

where k represents the grid index of point B. The *trunc* function returns the integer part of the CFL number and *frac* represents the fractional part of the CFL number.

$$\begin{aligned} u_k^{n+1} &= u_X^n \\ &= u_{j-1}^n + (1 - frac)(u_j^n - u_{j-1}^n) \\ &= frac \cdot u_{j-1}^n + (1 - frac)u_j^n \end{aligned} \tag{4.3}$$

u_X is approximated by interpolating u_{j-1} and u_j as shown above. With this scheme, the finite difference stencil covers the domain of dependence.

The discrete velocity Boltzmann equation can be said to be composed of two different parts, the advection part (left side) and the relaxation part (right side). By splitting the discrete velocity Boltzmann equation into these two parts, we can apply the above special discretization to the advection part and obtain the improved lattice Boltzmann equation. The improved lattice Boltzmann equation is shown below for the f_1 distribution function, with k, j representing the respective indices (related to Equations 4.2 and 4.3) in the horizontal direction:

$$f_1(k, t + \Delta t) = (1 - frac) \cdot f_1(j, t) + frac \cdot f_1(j - 1, t) - \omega \cdot \left(f_1(k - 1, t) - f_1^{eq}(k - 1, t)\right) \tag{4.4}$$

When CFL ≤ 1, the above equation will be the same as the equation for the f_1 distribution function derived from Equation 4.1.

4.1.1 Improved Stability

The time step size and CFL number for the improved LBM (ILBM) go over their previously defined limits. The results have shown that the maximum time step that can be taken is twice as large as the time step taken by the previous implementation. This would decrease by half the number of time iterations required to reach steady state. The CFL number can go up to 1.75 compared to the previous implementation, where it was limited to 1. Large local gradients inhibit the maximum time step size and CFL number for the ILBM. Zhou et al. [47] performed a stability analysis of the new discretization when applied to the one-dimensional advection equation.

4.1.2 Cache Optimization and Parallelization

The new implementation reduces computational efficiency per time step compared to the original lattice Boltzmann implementation, because there are now five two-dimensional arrays: the four distribution functions, and the macroscopic variable. This will reduce the amount of data that is updated in the cache-based algorithm during each time iteration. Extra communication is required in the parallel version to communicate the macroscopic variable (u) values at the subdomain boundaries. An expanded ghost region is also required for the distribution functions that need to be communicated. The expanded ghost region arises because of the domain of dependence lying within the neighboring subdomain, which requires the finite difference stencil to cover grid points on the boundary as well as their near neighbors. For instance, when $1 < CFL < 2$, updating the $f_1(k, y)$ distribution function would require $f_1(k-1, y)$ and $f_1(k-2, y)$. In the original LBM, the ghost region included only points on the subdomain boundary; i.e., the updating of $f_1(k, y)$ distribution function required just $f_1(k-1, y)$ and therefore the parallel version required only near-neighbor communication.

Previously, the computation of the right side of the lattice Boltzmann equation was of a local nature and, therefore, the right sides of all four equations were computed in the same do-loop; this allowed all of the floating-point operations to be performed in a single do-loop. However, now the computation of the right side involves adjacent grid points as well. Therefore, the computations for each distribution function are performed in separate do-loops. One other disadvantage is that the number of floating point operations increase in ILBM computations.

4.1.3 Results

To test the computational efficiency, performance, and accuracy of the ILBM versus the original LBM, the following measurements are considered:

1. Compute times and compute times ratios of the cache-optimized and parallel versions of both methods, the ILBM and the original LBM
2. Speedup obtained through parallelization of the cache-optimized version of both methods
3. Accuracy of both methods with their maximum possible time steps for certain viscosities
4. Maximum time step and CFL number possible for various grid sizes using the ILBM when the maximum local variation in the macroscopic field increases

The convergence criteria utilized on the ILBM would allow it to converge when the absolute difference of the current time iteration's L2-norm error and the previous time iteration's L2-norm error is less than 10^{-6}. L2-norm error was calculated using the following formula:

$$L_2 - Norm = \frac{\sqrt{\sum_{i=1}^{i\max}\left(\vec{u} - \vec{u}_{exact}\right)^2}}{\sqrt{i\max}} \tag{4.5}$$

where imax is the total number of grid points (imax = nx*ny). The original LBM usually reaches convergence for the given criterion of 10^{-6} around the same or greater final time as the ILBM. While computing the computation time results, the original lattice Boltzmann algorithm has been executed for the same final time as required by the ILBM algorithm to reach its convergence.

Table 4.1 compares the computation times of the ILBM with those of the original LBM. The cases considered are grids ranging from 2400×2400 to 4800×4800, with a Reynolds number of 500 (the Reynolds number here is the inverse of the given viscosity), and the grid size 6000×6000 with a Reynolds number of 2000. The comparison has been performed with cache-based, parallel algorithms. The number of processors ranges from 4 to 20. The ILBM can take a maximum time step that is twice as large as that allowed for the original LBM with the same viscosity. For all cases, the ILBM was on average 1.7 times faster than the original LBM. The ILBM was not twice as fast as the original method despite having a time step that was twice as large because of the extra memory, message passing communication, and floating point operations associated with the ILBM, as discussed in Section 4.1.2. Using 20 or more processors for the 2400 grid case does not result in a comparable increase in performance because of less computational work per processor. This results in a much lower relative speed than the average value. Another cause for this is that the arrays belonging to the original LBM fit into cache for the 20-processor case without any need for cache optimization (Chapter 3). This allows superior performance as recorded by its superlinear speedup, which is not shared by its ILBM counterpart, since it has more arrays that cannot fit into cache unless cache optimization is utilized.

Table 4.2 shows the error associated with both methods. The first column specifies the L2-norm error part I (the error associated with a convergence criterion of

10^{-7}) of the original LBM. The second column represents the L2-norm error part II (the error associated with a convergence criterion of 10^{-6}) of the original LBM. This is the error associated with having the same final time as the ILBM. The results shown for the ILBM are the final converged results; i.e., the results will not change even with further time iterations. The ILBM is less accurate than the original LBM. The modified lattice Boltzmann equation has reduced the ILBM results to first-order accuracy.

Table 4.1. Relative performance of the ILBM versus original LBM

Grid	LBM time step	LBM compute time (seconds)	ILBM time step	ILBM compute time (seconds)	LBM time / ILBM time
Re = 500					
Nx = 2400	**0.0003**		**0.0006**		
4 processors		640.97		363.85	1.76
8 processors		342.84		190.59	1.80
12 processors		223.93		128.08	1.75
16 processors		176.73		97.29	1.82
20 processors		124.41		85.99	1.45
Nx = 3600	**0.0002**		**0.0004**		
4 processors		2211.16		1285.18	1.72
8 processors		1092.80		679.64	1.61
12 processors		738.10		442.46	1.67
16 processors		576.86		351.50	1.64
20 processors		447.36		275.00	1.63
Nx = 4800	**0.00015**		**0.0003**		
4 processors		5559.69		3266.25	1.70
8 processors		2703.50		1550.04	1.74
12 processors		1711.07		1027.23	1.67
16 processors		1309.88		759.37	1.72
20 processors		1057.67		623.85	1.70
Re = 2000					
Nx = 6000	**0.0001**		**0.0002**		
4 processors		16546.68		10062.00	1.64
8 processors		5897.80		3469.89	1.70
12 processors		3834.37		2226.39	1.72
16 processors		2941.03		1741.27	1.69
20 processors		2435.38		1440.00	1.69

Table 4.2. L2-norm error for both methods

Grid Size	LBM_L2_error - I	LBM_L2_error - II	ILBM_L2_error
Re = 500			
2400	5.07E-05	3.10E-04	3.85E-03
3600	4.52E-05	2.35E-04	2.18E-03
4800	4.38E-05	2.10E-04	1.63E-03
Re = 2000			
6000	1.73E-05	1.71E-04	2.36E-03

Table 4.3 shows the parallel speedup defined as:

$$Parallel\ Speedup = \frac{CPU\ time\ for\ 4\,processor\ case}{CPU\ time\ for > 4\ processor\ case} \qquad (4.6)$$

The parallel speedup is linear or just under linear for the 2400×2400 and 3600×3600 grids for both the original LBM and ILBM cases with one exception: the 2400×2400 grid on 20 or more processors. The cause for this is that the arrays on each processor completely fit into cache without any need for cache optimization. The 4800×4800 and 6000×6000 grids show superlinear speedup for both the original LBM and ILBM cases. The reason for the superlinear speedup is poor performance of the four-processor case, since it has a greater amount of data per processor. This can be corroborated with the fact that the 6000×6000 case has a higher superlinear speedup than the 4800×4800 case.

Table 4.4 lists the maximum CFL number and time step size allowed for the given grids with the given Reynolds numbers. Higher Reynolds numbers result in greater local gradients that do not allow the ILBM to succeed. The original LBM can still resolve these greater local gradients.

4.1.4 Conclusions

The results indicate that the ILBM performs better than the original LBM method for the given cases. For grids smaller than 6000×6000, the ILBM performs well only for cases with Reynolds numbers less than or equal to 500. For the 6000×6000 grid, the ILBM can perform well for Reynolds numbers up to 5000. The accuracy has been reduced to first order for the ILBM.

Table 4.3. Parallel speedup of original LBM and ILBM

Grid	LBM compute time	ILBM compute time	LBM speedup	ILBM speedup
Re = 500				
Nx = 2400				
4 processors	640.97	363.85	1.00	1.00
8 processors	342.84	190.59	1.87	1.91
12 processors	223.93	128.08	2.86	2.84
16 processors	176.73	97.29	3.63	3.74
20 processors	124.41	85.99	5.15	4.23
Nx = 3600				
4 processors	2211.16	1285.18	1.00	1.00
8 processors	1092.80	679.64	2.02	1.89
12 processors	738.10	442.46	3.00	2.90
16 processors	576.86	351.50	3.83	3.66
20 processors	447.36	275.00	4.94	4.67
Nx = 4800				
4 processors	5559.69	3266.25	1.00	1.00
8 processors	2703.50	1550.04	2.06	2.11
12 processors	1711.07	1027.23	3.25	3.18
16 processors	1309.88	759.37	4.24	4.30
20 processors	1057.67	623.85	5.26	5.24
Re = 2000				
Nx = 6000				
4 processors	16546.68	10062.00	1.00	1.00
8 processors	5897.80	3469.89	2.81	2.90
12 processors	3834.37	2226.39	4.32	4.52
16 processors	2941.03	1741.27	5.63	5.78
20 processors	2435.38	1440.00	6.79	6.99

Table 4.4. Maximum time step and CFL number

Grid size	Reynolds number	Max CFL	Max time step
2400	500	1.75	0.0006
	2000	1.5	0.0005
3600	500	1.75	0.0004
	2000	1.5	0.0003
	5000	1.4	0.0003
4800	500	1.75	0.0003
	2000	1.5	0.0002
	5000	1.4	0.0002
6000	5000	1.6	0.0002

4.2 Coupling LBM with the ADI method for solving the two-dimensional Burger's equation

Section 3.4.8 showed the ADI method to be superior to the LBM when solving the steady Burger's equation. As the grid resolution increases, the performance of the LBM progressively decreases since the time step size decreases with grid spacing. However, the LBM outperforms the ADI method and other traditional methods when solving unsteady flows using Cartesian grids because the time step size is decided by accuracy requirements and is usually lower than the time step size required for numerical stability. The results from Section 3.4.5 showed the LBM to be faster than the ADI method by a factor of 8 to 8.5 when solving the two-dimensional unsteady Burger's equation, where both methods possessed the same numerical time step size. Most fluid flow problems possess rapid change in solution in narrow regions of the flow domain (such as boundary layers). These singular layers can be localized in space by dividing the computational domain into subdomains.

To adapt the grid to the behavior of the exact solution, greater grid resolution (fine grids) is required in the subdomains containing the singular layers while the rest of the subdomains can be discretized with coarser grids. This brings about an opportunity to use explicit methods on the coarse grid region and an implicit method on the fine grid region. The coarse grid allows the explicit method to take on bigger time steps that can be equal to those of the implicit method. Based on these observations, a hybrid LBM-ADI method is developed in this section to harness the computational efficiency of the LBM and the faster convergence properties of the ADI method for solving steady flow problems.

4.2.1 Applying the Hybrid Scheme to Burger's Equation with Multiblock Grid

The steady two-dimensional Burger's equation with the initial and boundary conditions mentioned in Section 3.4.7 will be used as the test problem. In this problem, there exists a steep gradient or boundary layer along the right boundary. A multi-block Cartesian grid will be used to discretize the domain. A high-resolution grid block will cover the region along the right boundary and a coarse grid block will cover the rest of the domain as shown in Figure 4.1. The LBM will be applied on the coarse grid block and the ADI method will be applied on the fine grid block. To enable transfer of information between the two methods, the two grid blocks overlap at the interface.

4.2.2 Coupling Procedure

The coupling procedure should enable the matching of the solution between adjacent subdomains. The Schwarz alternating method [48] is used as the coupling procedure on overlapping subdomains. This leads to the following method for communicating information between adjacent subdomains:

1. The variables on the coarse grid block are updated using the LBM. The distribution functions on the interface boundary of the coarse grid are computed from the macroscopic variable (u) at the corresponding position in the neighboring fine grid block.
2. The ADI solver computes the macroscopic variable in the fine grid region covering the boundary layer. The interface boundary conditions are specified using the macroscopic variables computed in Step 1 at the corresponding coarse grid locations. Since the grid spacing for both blocks is different, linear interpolation is performed to obtain the interface boundary conditions at the fine grid points that do not have a corresponding coarse grid point.

As outlined in the above two steps, the coupling procedure essentially consists of solving two Dirichlet problems on overlapping subdomains. With the traditional Schwarz alternating method, Steps 1 and 2 have to be executed alternately until the difference between the current solution and the solution from the previous iteration on the interface boundary of each grid block is less than a given tolerance. This procedure occurs at very time step. But to make full use of the cache-efficient nature of the LBM, both Steps 1 and 2 need to be executed separately for a certain number of time steps with the same interface boundary conditions. There are two strategies that can be adopted for making use of the LBM's cache-efficient nature. One strategy is to separately execute Steps 1 and 2 for *tdiv* number of time steps (Section 3.2), and then update the interface boundary conditions before proceeding to the next installment of *tdiv* time steps until the convergence criteria for the Schwarz alternating method is achieved. This procedure will be carried on until the steady state conditions are reached. A high-level description of this strategy (termed Strategy 1) is given below.

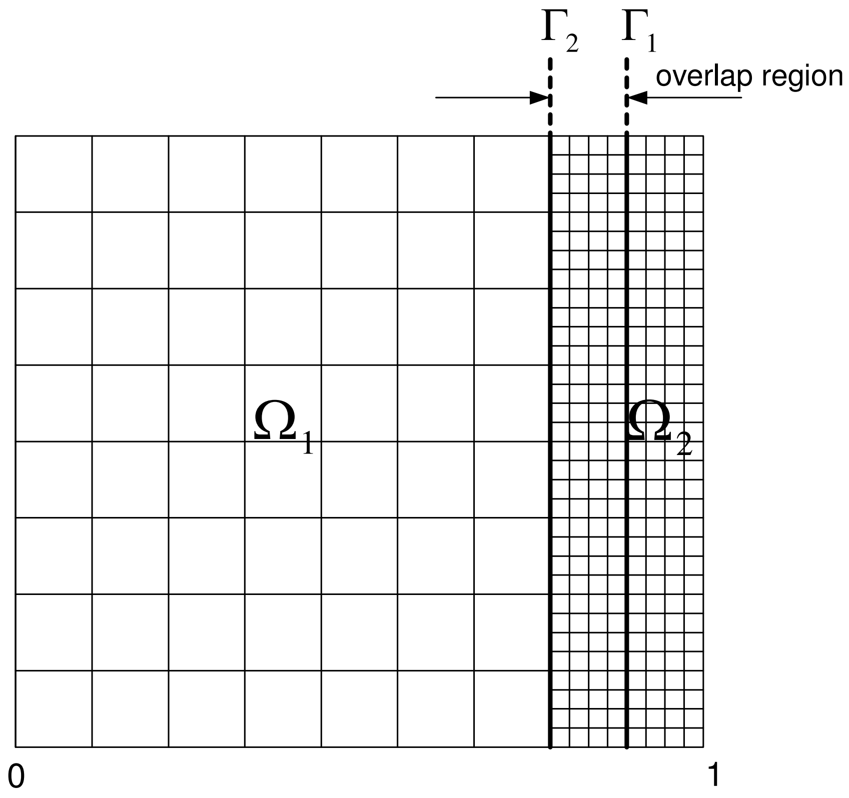


Figure 4.1. Overlapping grids

Let Ω_1, Ω_2 represent the two overlapping grid blocks (or subdomains), and $\partial\Omega_1$, $\partial\Omega_2$ represent their boundaries. The part of $\partial\Omega_1$ lying in Ω_2 is represented with Γ_1 (artificial boundary or internal boundary or interface boundary of Ω_1). Likewise, Γ_2 represents the interface of Ω_2.

```
1.  Initialize the distribution functions on $\Omega_1$ and the macroscopic variable on $\Omega_2$
2.  Loop for all time steps (until final time or until steady state conditions are reached in the whole domain)
    2.1. Loop for Schwarz iterations (to update interface boundary conditions until they do not change significantly)
         a. Distribution functions$|\Gamma_1$ are computed using u$|\Omega_2$ near $\Gamma_1$
         b. Loop for $tdiv$ number of time steps (for LBM cache optimization)
         c. Update distribution functions on $\Omega_1$ using LBM (boundary conditions remain unchanged)
         d. End loop for $tdiv$ time steps
         e. Compute u$|\Gamma_2$ from the distribution functions$|\Omega_1$ near $\Gamma_2$
         f. Loop for $tdiv$ number of time steps (so that $\Omega_2$ will have the same final time as $\Omega_1$)
         g. Perform ADI computations to update u on $\Omega_2$ (boundary conditions remain unchanged)
         h. End loop for $tdiv$ time steps
    2.2. Loop ends for Schwarz iterations when the solution on both interface boundaries does not change significantly with the next iteration
3.  End loop for all time steps when steady state solution is reached
```

The second strategy is to execute Steps 1 and 2 independently without updating the interface boundary conditions or communicating any information between the subdomains until steady state conditions are reached in the respective subdomains. After this, the Schwarz alternating method is applied so that Steps 1 and 2 are executed alternately and the interface boundary conditions are updated accordingly until the solution at both interface boundaries does not change significantly with new Schwarz iterations or updates. A high-level description for this strategy (termed Strategy 2) is given below.

1. Initialization of the distribution functions on Ω_1 and the macroscopic variable on Ω_2
2. Loop for time stepping until steady state conditions are reached in Ω_1 (keeping the same boundary conditions)
3. Loop for time stepping until steady state conditions are reached in Ω_2 (keeping the same boundary conditions)
4. Loop for Schwarz iterations (to update interface boundary conditions until they do not change significantly)
    a. Distribution functions$|\Gamma_1$ are computed using $u|\Omega_2$ near Γ_1
    b. Update distribution functions on Ω_1 using the LBM (boundary conditions remain unchanged)
    c. Compute $u|\Gamma_2$ from the distribution functions$|\Omega_1$ near Γ_2
    d. Perform ADI computations to update u on Ω_2 (boundary conditions remain unchanged)

5. Loop ends for Schwarz iterations when the solution on both interface boundaries does not change significantly with the next iteration

4.2.3 Parallel Implementation

The coupled LBM-ADI method is implemented on a parallel computer with the parallel LBM and the parallel ADI method being executed alternately using the Schwarz method. Figure 4.2 shows the parallel domain decomposition for the respective solvers. At the beginning of each Schwarz step, information needs to be transferred from one grid block to the other. This requires global communication using mpi_scatter when transferring information from the lattice Boltzmann grid block to the ADI grid block or mpi_gather when transferring information from the ADI grid block to the lattice Boltzmann grid block.

4.2.4 Method for Comparison

The LBM-ADI method is a coupled explicit-implicit method. To quantify its computational performance and accuracy, it should be compared with traditional finite difference methods that are applied over the whole domain. The LBM-ADI method is compared with the ADI method implemented on the multiblock grid (ADI-ADI method). The ADI-ADI procedure implements the ADI method separately on all grid blocks in the domain. The coupling procedure described in Section 4.2.2 is therefore also applied here. In the parallel implementation, the communication between the two subdomains is non-global, unlike the coupled LBM-ADI method.

4.2.5 Results

The computations were performed on an SGI Onyx 2 using 20 processors. The results were computed for a Reynolds number of 50,000. The gradient covers the region from $x = 0.9996$ to $x = 1.0$. The region $x = 0.9875$ to $x = 1.0$ along the right boundary was meshed with a 480×38401 grid block and the rest of the domain was meshed with a 2380×2401 grid block. The refinement factor between the two grid blocks is 1:16; i.e., grid spacing for the coarse grid block (LBM) is 16 times the grid spacing for the fine grid block (ADI method).

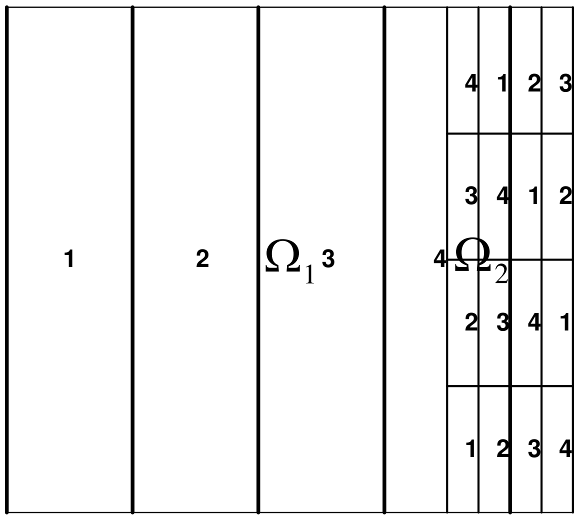


Figure 4.2. Domain decomposition for the parallel LBM and parallel ADI methods

At a Reynolds number of 50,000, the biggest time step that the ADI method can take is 0.0002. Computational experiments found that using a time step greater than 0.0002 for the ADI scheme at Re = 50,000 makes it take considerably longer to converge, if it converges at all. The LBM can take a time step of 0.0002 since the grid spacing for the coarse grid block is 0.000416 (Section 3.1.5). Table 4.5 shows the L_2-norm error and computation time for the LBM-ADI method and the ADI-ADI method. The error in the fine grid block covering the boundary layer subdomain is the same for both methods, since the ADI method is solving this region. The error differs slightly for the two strategies. The results for Strategy 1 will be outlined first.

For Strategy 1, the total ADI-ADI CPU time when using 20 processors is 6634 seconds. The LBM-ADI CPU time is 5387 seconds. Here, the time spent on parallel communication is included with the CPU time. The total number of grid points discretizing the whole domain is 24.1 million (Table 4.5). The coarse grid has 23.66% of the total number of grid points. This percentage will provide an estimate of the CPU time appropriated to the coarse grid block and the fine grid block for computations of the ADI-ADI method using Strategy 1. The CPU time spent on the coarse grid is 1570 seconds (23.66% of 6634). Therefore, CPU time for the fine grid block is 5064 seconds.

Table 4.5. Results using Strategy 1

Grid points (total)	**Coarse grid block points**	**Fine grid block points**
2.41E+07	5.71E+06	1.84E+07
ADI-ADI CPU time	**CPU time on coarse grid**	**CPU time on fine grid**
6634 seconds	1570 seconds	5064 seconds
ADI-ADI error	**L2-norm error on coarse grid**	**L2-norm error on fine grid**
	3.32E-08	2.59E-03
LBM-ADI CPU time	**CPU time on coarse grid**	**CPU time on fine grid**
5387 seconds	323 seconds	5064 seconds
LBM-ADI error	**L2-norm error on coarse grid**	**L2-norm error on fine grid**
	1.48E-09	2.59E-03

Since the LBM-ADI method completes the same number of iterations for convergence as the ADI-ADI method, both methods will share the same CPU time for the fine grid block (the difference in parallel communication times between the subdomains for the two methods is neglected). This allows an estimate of the CPU time spent on the coarse grid block for the LBM-ADI method, where the LBM is in operation. This estimate is 323 seconds. This shows that although the LBM-ADI method reduces the overall computation time by a factor 1.23, the coarse grid block is reduced by a factor 4.86. It is expected that an increase in the subdomain size covered by the coarse grid block relative to the fine grid block will reduce the overall computation time by a bigger factor because there will be an increase in the percentage of the computation performed by the LBM. The overall CPU time reduction factor is small (1.23) because the distribution functions are updated alternately with the variables in the ADI method when using Strategy 1. Therefore, the

LBM is unable to realize its full computational efficiency, which is possible if the distribution functions were the only data being updated.

Using Strategy 2, the LBM-ADI method is 4.5 times faster than the ADI-ADI method; i.e., CPU time reduction factor is 4.5 (refer to Table 4.6) This increase in the reduction factor relative to Strategy 1 is mainly due to the LBM being allowed to update the information on the coarse grid subdomain until steady state conditions are achieved in that region without communicating with the neighboring subdomain; i.e., the LBM no longer alternates with the ADI method for the majority of the computations.

Table 4.6. Results using Strategy 2

Grid points (total)	**Coarse grid block points**	**Fine grid block points**	
2.41E+07	5.71E+06	1.84E+07	
ADI-ADI Method	**Coarse grid iterations**	**Fine grid iterations**	**Schwarz iterations**
	4679	111	311
Total CPU time	**CPU time on coarse grid**	**CPU time on fine grid**	**CPU time for Schwarz iterations**
1888.32 seconds	1355.72 seconds	121.08 seconds	411.19 seconds
ADI-ADI error	**L2-norm error on coarse grid**	**L2-norm error on fine grid**	
	7.90E-06	3.06E-03	
LBM-ADI Method	**Coarse grid iterations**	**Fine grid iterations**	**Schwarz iterations**
	4968	121	98
Total CPU time	**CPU time on coarse grid**	**CPU time on fine grid**	**CPU time for Schwarz iterations**
419.31 seconds	173.07 seconds	132.12 seconds	113.64 seconds
LBM-ADI error	**L2-norm error on coarse grid**	**L2-norm error on fine grid**	
	1.14E-06	2.59E-03	

Table 4.6 shows the total number of iterations for convergence to steady state on the two subdomains and the number of Schwarz iterations for convergence on the interface boundaries. The LBM-ADI method reduces the CPU time spent on the coarse grid by a factor of 7.8 relative to the ADI-ADI method (Table 4.6). This is to be expected since the LBM is 8 times faster than the ADI method when the number of iterations is similar (Section 3.4).

4.2.6 Conclusions

From the results for both strategies, it is expected that an increase in the size of the region represented by the LBM will result in increased reduction in the CPU times relative to traditional methods. The results presented here were for steady state problems. The methods outlined above can also be used for numerical solution of unsteady flow fields. In certain cases where very high grid resolution is required in resolving boundary layers, the time step size for the LBM may be below the time step size required for time accuracy. Therefore, an implicit method can be used in the boundary layer subdomain, while the LBM can be used in the outer subdomain for unsteady flows. When using the coupled LBM-ADI method for unsteady flows, Strategy 1 is the viable choice.

CHAPTER 5.

COUPLED LBM–TRADITIONAL METHODS FOR INCOMPRESSIBLE FLOW PROBLEMS

From Chapter 4, it can be seen that using multiple solvers on a multiblock grid is efficient since certain solvers are more suited for computing certain regions of the flow domain. This chapter presents a multiblock, multi-solver technique for solving incompressible flow problems. The lattice Boltzmann method (LBM) and traditional finite difference methods are coupled to solve the backward facing step flow problem and the flow-around-cylinder problem.

In the backward facing step flow problem, the region near the corner or separation point requires a high-resolution grid to identify/resolve the recirculating eddies near the bottom of the step. The LBM is computationally efficient relative to traditional methods when solving time marching problems given that the time step size is similar for both methods. When using a multiblock arrangement, the LBM can possess the same time step size as traditional finite difference methods over some grid blocks. This means that the LBM is a better choice for solving in those grid blocks. In grid blocks where the LBM time step size is significantly less compared to traditional methods, the latter should be used as the solver.

5.1 LBM for Navier-Stokes Equations

The previous chapters used the LBM for two-dimensional Burger’s equation to prove its computational superiority. Here, the LBM for Navier-Stokes equations will be formulated. The LBM for the two-dimensional Navier-Stokes equations uses nine distribution functions at each point in the grid (Figure 5.1). One of the nine distribution functions possesses a zero velocity.

$$\begin{aligned}
\vec{c}_0 &= 0, 0 \\
\vec{c}_1 &= +c, 0 \\
\vec{c}_2 &= 0, +c \\
\vec{c}_3 &= -c, 0 \\
\vec{c}_4 &= 0, -c \\
\vec{c}_5 &= +c,\ +c \\
\vec{c}_6 &= -c,\ +c \\
\vec{c}_7 &= -c, -c \\
\vec{c}_8 &= +c, -c
\end{aligned} \tag{5.1}$$

$$c = \Delta x / \Delta t \tag{5.2}$$

where Δt is the numerical time step and Δx is the spatial discretization.

5.1.1 Relating the Lattice Boltzmann Equation to the Navier-Stokes Equations

The incompressible Navier-Stokes equations can be derived from the lattice Boltzmann equation (Equation 3.2) using a multiscale expansion [35]. The two-dimensional incompressible Navier-Stokes equations are presented in vector form:

$$\begin{aligned}
&\nabla \cdot \vec{u} = 0 \\
&\partial_t \vec{u} + (\vec{u}\nabla)\vec{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u}
\end{aligned} \tag{5.3}$$

The dependent variables in the Navier-Stokes equations are defined in terms of the distribution functions $f_i(\vec{x},t)$ and the viscosity is related to the collision frequency. The density is defined as

$$\rho(\vec{x},t) = \sum_i f_i(\vec{x},t) \qquad i = 0,\ldots 8 \tag{5.4}$$

The pressure can be computed using the density as

$$p = c_s^{\,2} \rho$$

Where c_s is the speed of sound and is given by $c_s = \dfrac{c}{\sqrt{3}}$

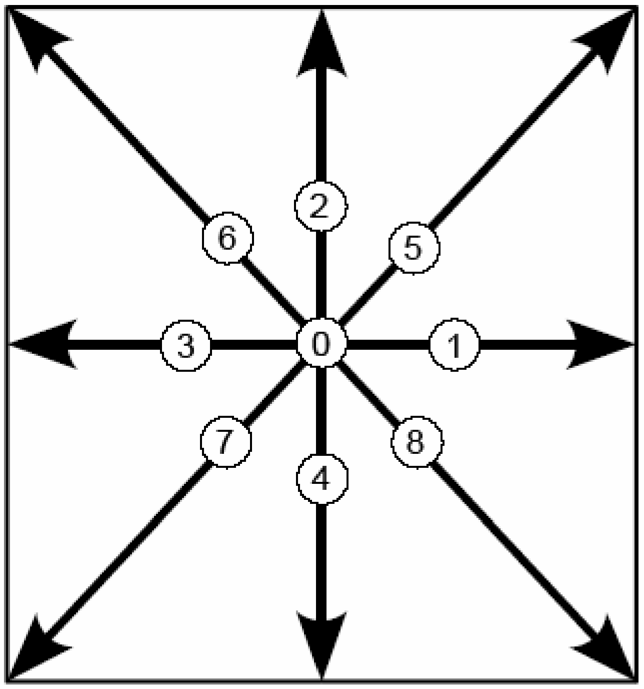


Figure 5.1. Schematic of a nine-speed lattice Boltzmann model

The momentum density is defined as follows:

$$\rho(\vec{x},t)\vec{u}(\vec{x},t) = \sum_i \vec{c}_i f_i(\vec{x},t) \qquad i = 0,\ldots 8 \tag{5.5}$$

The equilibrium distributions have been defined as follows [35]:

$$
\begin{aligned}
f_i^{eq} &= \frac{4}{9}\rho\left[1 - \frac{3}{2}\frac{\vec{u}^2}{c^2}\right] && i=0 \\
f_i^{eq} &= \frac{1}{9}\rho\left[1 + 3\frac{\vec{c}_i\cdot\vec{u}}{c^2} + \frac{9}{2}\frac{(\vec{c}_i\cdot\vec{u})^2}{c^4} - \frac{3}{2}\frac{\vec{u}^2}{c^2}\right] && i=1,2,3,4 \\
f_i^{eq} &= \frac{1}{36}\rho\left[1 + 3\frac{\vec{c}_i\cdot\vec{u}}{c^2} + \frac{9}{2}\frac{(\vec{c}_i\cdot\vec{u})^2}{c^4} - \frac{3}{2}\frac{\vec{u}^2}{c^2}\right] && i=5,6,7,8
\end{aligned}
\tag{5.6}
$$

For more accurate computation of steady flows, the equilibrium distribution functions have been modified slightly [49].

The kinematic viscosity is defined as

$$\nu = \frac{1}{3}c^2\Delta t\left(\frac{1}{\omega} - \frac{1}{2}\right). \tag{5.7}$$

5.1.2 Lattice Boltzmann Algorithm

To implement an explicit time marching lattice Boltzmann solver, the following substeps are required at each time step:

1. Using the initial $\vec{u}$, calculate the equilibrium distributions (Equation 5.6) and set $f_i = f_i^{eq}$ for the first time step
2. Compute the right-hand side (RHS) of the lattice Boltzmann equation (Equation 3.2), called collision substep, and propagate the result to the nearest neighbor nodes obtaining $f_i(\vec{x} + \vec{c}_i\Delta t, t + \Delta t)$. The unknown distribution functions at the boundary are computed using non-equilibrium distribution functions or bounceback conditions (specified in the next section)
3. Update the density and the momentum density from the new distributions according to the given definitions (Equations 5.4 and 5.5)

Start the next time step with the calculation of new equilibrium distributions using the new macroscopic variables (Equation 5.6) and proceed with Step 2 of the algorithm. Follow this procedure until the final time is reached.

5.1.3 LBM for Backward Facing Step Flow

The backward facing step flow geometry is shown in Figure 5.1. The figure shows solid walls at boundary B1, the step wall, and at B2 and B3, the base wall and the upper wall. The no-slip boundary conditions at the walls are imposed through the bounceback boundary conditions [35]. Here, the collision step is not performed at the walls, whereas the propagation step is still implemented. The implementation of the propagation step for some of the distribution functions on the walls requires information from outside the domain and therefore they remain unknown. Following the propagation step, these unknown f_i are assigned the value of the f_i of the opposite direction. For instance, at a point on the upper boundary, the unknown distribution functions are f_4, f_7, and f_8. Therefore, on the upper wall:

$$\begin{aligned} f_4(i,j) &= f_2(i,j) \\ f_7(i,j) &= f_5(i,j) \\ f_8(i,j) &= f_6(i,j) \end{aligned} \tag{5.8}$$

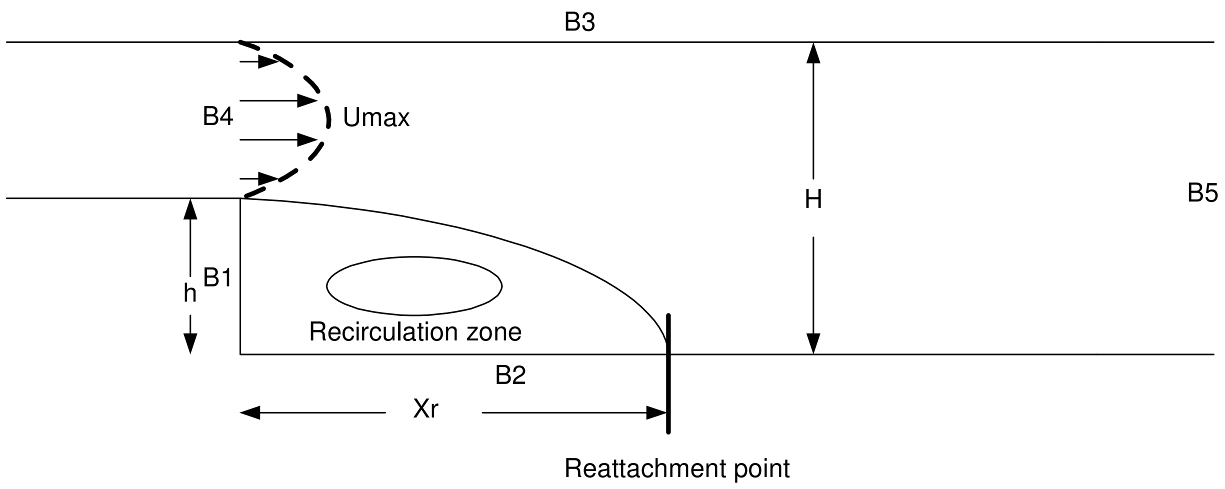


Figure 5.2. Schematic of backward step flow geometry; Xr is the re-attachment length and h is height of the backward step.

Assigning the inflow and outflow boundary conditions in the lattice Boltzmann algorithm requires a different methodology. In traditional CFD methods, the initial and boundary conditions are specified in terms of the momentum or pressure variables. To obtain the initial and boundary conditions in terms of the distribution functions, an inverse mapping between the distribution function and the macroscopic variable (a direct mapping being Equations 5.4 and 5.5) is required. The distribution functions at the inflow and outflow are defined as the equilibrium distributions or the sum of equilibrium distributions and a non-equilibrium term such as:

$$f_i = f_i^{eq} + f_i^{neq} \tag{5.9}$$

Skordos [37] obtained the non-equilibrium distributions for a seven-speed LBM using a multiscale expansion. For the nine-speed lattice Boltzmann model for the Navier-Stokes equations, these can be given in tensor form as:

$$\begin{aligned}
f_i^{neq} &= -\frac{\Delta t}{\omega}\cdot\frac{1}{3c^2}\left(c_{i\alpha}^2\frac{\partial u_\alpha}{\partial x_\alpha}\right) && i=1,2,3,4 \\
f_i^{neq} &= -\frac{\Delta t}{\omega}\cdot\frac{1}{12}\left(\frac{\partial u_\alpha}{\partial x_\beta}\right) && i=5,7 \\
f_i^{neq} &= \frac{\Delta t}{\omega}\cdot\frac{1}{12}\left(\frac{\partial u_\alpha}{\partial x_\beta}\right) && i=6,8 \\
f_i^{neq} &= 0 && i=0
\end{aligned} \qquad (5.10)$$

Equation 5.10 can also be used for specifying boundary conditions at the walls. The density at the outlet (B5) is fixed, whereas the density at the inlet (B4) is extrapolated from density values within the flow domain. The velocity at the inlet is given by the problem-specified parabolic profile and it remains constant throughout the time marching to steady state conditions. The velocity at the outlet is defined such that there is zero velocity gradient at the outlet.

5.1.4 Stability of LBM for Backward Facing Step Flow

Computer experiments have shown that the time step is limited to a little over one-fourth of the spatial discretization when solving the backward facing step flow for the given parameters. The reason for this is the requirement to have a low Mach number ($|\vec{u}|/c$) for the simulation. A high Mach number results in compressibility errors since the multiscale expansion that relates the LBM to the Navier-Stokes equations assumes a small Mach number.

5.1.5 Cache Optimization of LBM for Navier-Stokes Equations

The LBM for the two-dimensional incompressible Navier-Stokes equations uses nine distribution functions, each requiring a two-dimensional array. Considering an 8 MB cache on a single processor, the biggest square grid size that can be accommodated within the cache for repeated use is around 250×250, since

$$9\,\text{arrays} \times 250^2\ \text{elements} \times 8\ \text{bytes/elements} = 4.4\text{MB}. \tag{5.11}$$

On a single processor, grid sizes that are larger than 250×250 can be accommodated in the cache by dividing the grid into subsections that can fit in cache and applying the cache optimization algorithm mentioned in Section 3.2.

5.2 Vorticity-Stream Function Formulation

The vorticity-stream function method is a simple and efficient traditional procedure for solving two-dimensional incompressible flow problems [50]. It consists of a vorticity transport equation and a Poisson equation for the stream function that allows the stream function to automatically satisfy the conservation of mass constraints:

$$\frac{\partial \zeta}{\partial t} + u\frac{\partial \zeta}{\partial x} + v\frac{\partial \zeta}{\partial y} = \nu\left(\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2}\right) \tag{5.12}$$

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\zeta \tag{5.13}$$

The vorticity is defined as:

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{5.14}$$

and the velocity components are defined in terms of the stream function as:

$$\begin{aligned} u &= \frac{\partial \psi}{\partial y} \\ v &= -\frac{\partial \psi}{\partial x} \end{aligned} \tag{5.15}$$

The solution for these equations is computed using a time marching procedure [50]:

1. Assign initial and boundary values to ψ, and ζ at all grid points
2. A finite difference analog of the vorticity transport equation is used to calculate the vorticity at the next time level, $t+\Delta t$
3. The Poisson equation is solved using a cyclic reduction technique with the new interior values of the vorticity in the source term of the equation
4. The new velocity components are calculated from the stream function
5. The final step is to update the boundary values of the vorticity from the new ψ at the neighboring interior points

This computational procedure is repeated from step 2 onwards until the convergence criteria are met.

5.2.1 Numerical solution

The diffusion terms on the RHS of Equation 5.12 were discretized using central differences. The spatial derivatives in the convection terms of the vorticity transport equation were initially discretized using central differences. However, this discretization was numerically unstable for higher grid resolution and higher Reynolds numbers due to loss of diagonal dominance in the matrix obtained from the discretization. Consequently, a first-order upwind difference scheme was adopted for the convection terms. To obtain a tridiagonal representation of the matrix resulting from applying the discretization at all grid points, a time splitting technique (the alternating direction implicit scheme or ADI scheme) was implemented. The forward Euler discretization of the time derivative was split into two parts; i.e., two half-steps are performed to advance one time step. The finite difference analog is given below:

$$\begin{aligned}
&\text{Step 1}: \quad \frac{\zeta_{i,j}^{n+1/2}-\zeta_{i,j}^{n}}{\Delta t/2}+u_{i,j}^{n}\frac{\nabla_x\zeta_{i,j}^{n+1/2}}{\Delta x}+v_{i,j}^{n}\frac{\nabla_y\zeta_{i,j}^{n}}{\Delta y}=\nu\left(\frac{\delta_x^2\zeta_{i,j}^{n+1/2}}{(\Delta x)^2}+\frac{\delta_y^2\zeta_{i,j}^{n}}{(\Delta y)^2}\right)\\
&\text{Step 2}: \quad \frac{\zeta_{i,j}^{n+1}-\zeta_{i,j}^{n+1/2}}{\Delta t/2}+u_{i,j}^{n}\frac{\nabla_x\zeta_{i,j}^{n+1/2}}{\Delta x}+v_{i,j}^{n}\frac{\nabla_y\zeta_{i,j}^{n+1}}{\Delta y}=\nu\left(\frac{\delta_x^2\zeta_{i,j}^{n+1/2}}{(\Delta x)^2}+\frac{\delta_y^2\zeta_{i,j}^{n+1}}{(\Delta y)^2}\right)
\end{aligned} \tag{5.16}$$

The operators ∇_x and ∇_y represent backward differencing while δ_x^2 and δ_y^2 represent central differencing. The upwind differencing will be a backward difference if the advection coefficients are positive or a forward difference if the advection coefficients are negative. Considering positive $u_{i,j}$ and $v_{i,j}$, the operators are defined as follows:

$$\begin{aligned}
\nabla_x\zeta_{i,j}&=\zeta_{i,j}-\zeta_{i-1,j}\\
\nabla_y\zeta_{i,j}&=\zeta_{i,j}-\zeta_{i,j-1}\\
\delta_x^2\zeta_{i,j}&=\zeta_{i+1,j}-2\zeta_{i,j}+\zeta_{i-1,j}\\
\delta_y^2\zeta_{i,j}&=\zeta_{i,j+1}-2\zeta_{i,j}+\zeta_{i,j-1}
\end{aligned} \tag{5.17}$$

Applying step 1 and step 2 in equation 5.16 to all the grid points in the flow domain results in tridiagonal systems of linear algebraic equations. First, the equations resulting from step 1 are solved to obtain the vorticity at time level $n+1/2$. To do this, a tridiagonal matrix is solved for each *j* row of points. The vorticity at time level $n+1/2$ is used in the discretization of step 2. During step 2, a tridiagonal matrix is solved for each *i* row of grid points, obtaining the vorticity at time $t+\Delta t$ for all interior grid points. On an $N \times N$ square grid, both half time steps consist of Gaussian elimination computations for solving N independent tridiagonal equation systems, each system containing N unknowns.

The Poisson equation for the stream function is solved using FISHPACK [51]. FISHPACK uses a five-point finite difference approximation to the stream function equation (Equation 5.13) and solves the resulting algebraic equations using the cyclic reduction techniques. The FISHPACK solver allows the specification of both Dirichlet and Neumann boundary conditions.

5.2.2 Boundary Conditions for the Stream Function and the Vorticity

The stream function is taken as zero for the step wall and the base wall (B1 and B2); i.e., ψ_{B1} and $\psi_{B2} = 0$. The stream function at the inlet (B4) is specified such that a parabolic velocity profile is obtained for the horizontal velocity component, u, while the vertical velocity component is zero. The following analytical expression is used for the stream function at the inlet:

$$\left(\psi_{1,j}\right)_{B4} = \frac{K}{2\nu}\left(-\frac{y^3}{3} + \frac{3y^2}{2} - 2y + \frac{5}{6}\right) \qquad (5.18)$$

where K is a constant, ν is viscosity and y is the vertical distance from the base. Equation 5.18 is applied only at the beginning of the time marching procedure. For subsequent time step computations, the stream function at the inlet is defined as follows to avoid instability:

$$\left(\psi_{1,j}\right)_{B4} = \frac{\left(4\psi_{2,j} - \psi_{3,j}\right)}{3} \qquad (5.19)$$

The above expression states that the horizontal gradient of the stream function is zero, which means the vertical velocity component is zero at the inlet. The stream function for the upper boundary (whose j grid index is ny) is taken as equal to the stream function value at the inlet, thereby indirectly stating that the vertical velocity component is zero on B3 since the horizontal gradient in the stream function on B3 is zero:

$$\psi_{i,ny} = \psi_{1,ny} \qquad (5.20)$$

The stream function at the outlet, B5 (whose i grid index is nx), is extrapolated from the interior points so as to provide outflow boundary conditions that have zero horizontal gradient for the vertical velocity component.

$$\left(\frac{\partial^2\psi}{\partial x^2}\right)_{B5} = 0 \quad \text{or} \quad \left(\psi_{nx,j}\right) = 2\psi_{nx-1,j} - \psi_{nx-2,j} \tag{5.21}$$

The wall vorticity conditions are defined using the stream function values at and near the wall. The vorticity boundary conditions at the step wall are:

$$\left(\zeta_{1,j}\right)_{B1} = \frac{2\left(\psi_{1,j} - \psi_{2,j}\right)}{\left(\Delta x\right)^2} = -\frac{2\psi_{2,j}}{\left(\Delta x\right)^2} \tag{5.22}$$

The vorticity at the separation point or corner point (grid indices, $i = 1, j = jc$) is specially handled [50]

$$\zeta_{1,jc} = -\frac{2\psi_{1,jc+1}}{\left(\Delta y\right)^2} - \frac{2\psi_{2,jc}}{\left(\Delta x\right)^2} \tag{5.23}$$

The vorticity conditions at the base wall and the upper wall are given as:

$$\begin{aligned} \zeta_{i,1} &= \frac{2\left(\psi_{i,1} - \psi_{i,2}\right)}{\left(\Delta y\right)^2} \\ \zeta_{i,ny} &= \frac{2\left(\psi_{i,ny} - \psi_{i,ny-1}\right)}{\left(\Delta y\right)^2} \end{aligned} \tag{5.24}$$

The vorticity at the inlet, B4 is defined as:

$$\begin{aligned} \left(\zeta_{1,j}\right)_{B4} &= \left(\frac{\partial v}{\partial x}\right)_{B4} - \left(\frac{\partial u}{\partial y}\right)_{B4} \\ \left(\frac{\partial v}{\partial x}\right)_{B4} &= -\left(\frac{\partial^2\psi}{\partial x^2}\right)_{B4} = -\frac{\left(\psi_{1,j} - 2\psi_{2,j} + \psi_{3,j}\right)}{\left(\Delta x\right)^2} \\ \left(\frac{\partial u}{\partial y}\right)_{B4} &= \left(\frac{\partial^2\psi}{\partial y^2}\right)_{B4} = \frac{K}{2\nu}\left(3 - 2y\right) \end{aligned} \tag{5.25}$$

The vorticity outflow conditions are:

$$\zeta_{nx,j} = \zeta_{nx-1,j} \tag{5.26}$$

5.2.3 Vorticity-Stream Function Method and Cache Optimization

The largest square grid size that can be accommodated within an 8 MB size cache (where only 4 MB is available for repeated use) is around 450×450. This is because there are only three two-dimensional arrays in this method: the vorticity arrays at the two time levels and the stream function array. For bigger grids, the vorticity-stream function method is not suited for cache optimization due to the global nature of the Gaussian elimination computations for the vorticity transport equation and the cyclic reduction technique for the Poisson equation. During each time step, tridiagonal matrices are inverted using elimination along the length of the whole domain. Due to this, sections of the domain cannot be updated separately, which means data cannot be called from the cache repeatedly.

5.3 Numerical Performance

To compare the performance of the LBM and the vorticity-stream function method, the computation time and accuracy of both methods is evaluated. The results are computed on an SGI Onyx2™ with 8 MB cache and 512 MB main memory per processor. Fortran 90 was the programming language.

The maximum velocity $U_{\max} = 1$, the step height = h, the downstream channel width is H, resulting in an expansion parameter $r = H/h$, and the recirculation length of the primary vortex, or in other terms the reattachment length, is X_r. The downstream boundary is located at $x = 30h$. Two cases were evaluated for the computation times of both methods. In the first case, computations were performed for a Reynolds number of 150. The second case was for a Reynolds number of 500 and step height. The expansion ratio for both cases was 1.96. The Reynolds number is defined as:

$$\mathrm{Re} = \frac{4U_{\max} h}{3\nu} \tag{5.27}$$

5.3.1 Accuracy

To ascertain the accuracy of the LBM and the vorticity-stream function method, the reattachment length behind the step is used as a reference. Increasing the Reynolds number leads to an increase in the reattachment length. To show this relationship, Figure 5.3 plots the (normalized) reattachment length divided by the length of the domain on the vertical axis against the Reynolds number on the horizontal axis. These results were obtained from Armaly et al. [52].

For test case 1, the grid sizes that were tested and the resulting reattachment length are given in Table 5.1. For the lower grid resolution case where the grid spacing is $\Delta x = 0.05$, the reattachment length for the vorticity-stream function method (represented as VSM in the table) is lower than the experimental values that give a reattachment length near 4 [52]. This is due to the first-order upwind discretization of the spatial derivatives, which leads to results of first-order accuracy. The $\Delta x = 0.05$ grid spacing is unable to resolve the secondary vortex or recirculating eddy at the bottom of the step since the region containing this structure is from 0 to 0.1 in the x direction and 0 to 0.1 in the y direction (Figure 5.4). The next two higher-grid resolutions for the vorticity-stream function method give accurate reattachment length results and also capture the structure of the secondary vortex at the bottom of the step. The vorticity-stream function computations were performed until a final time of 76.0 (the term “final time” here is the total number of iterations multiplied by the numerical time step and is, therefore, dimensionless) was reached. The flow structure remained unchanged with further iterations. For the grid size 3001×201 ($\Delta x = 0.01$), 19000 iterations were performed to reach the final time since the time step size is 0.004.

The largest grid spacing ($\Delta x = 0.05$) in Table 5.1 gives an LBM reattachment length that is the same as experimental results [52]. The reattachment length given by the LBM increases slightly with increasing grid resolution. The LBM resolved the primary recirculating region and gave the correct reattachment length at a final time 76.0, but failed to resolve the secondary vortex at the bottom of the step at this final time. However, with further iterations up to a final time of 156.0 (which is equivalent to a total number of iterations of 39000 for $\Delta x = 0.01$), the LBM was able to resolve this secondary vortex. The reason for the LBM taking up more iterations than the vorticity-stream function approach, despite both approaches possessing the same time step size, presumably lies in the LBM’s explicit computational nature. The explicit computational nature precludes obtaining information from the whole computational domain as done by implicit methods.

Table 5.1. Re = 150, reattachment length

grid spacing	grid size	Xr for LBM	Xr for VSM
0.05	601x41	4.2	3.6
0.02	1501x101	4.25	4.05
0.01	3001x201	4.30	4.0

Test case 2 consists of results for Reynolds number 500 (Table 5.2). Armaly et al.'s results (Figure 5.3) show the reattachment length to be closer to 10 for a Reynolds number of 500. The reattachment length resulting from LBM computations is closer to experimental values than that given by the vorticity-stream function method. The results obtained by the vorticity-stream function method deteriorate for a Reynolds number of 500 as compared to the results for Reynolds number 150 for the same grid spacing. This is due to the first order upwind spatial discretization. Steady state conditions are attained when the computed reattachment length is close to the experimental results [52] and when the secondary vortex at the bottom of the step is resolved. The vorticity-stream function method attained steady state conditions at a final time 213.0. Although LBM does not reach steady state (requiring resolution of bottom secondary vortex) at final time 213.0, it gives a reasonably accurate reattachment length. Therefore, further iterations are performed and the accuracy of the LBM increases; i.e., there is an increase in the reattachment length with a greater final time along with resolution of the secondary vortex at the bottom of the step. Table 5.2 shows the LBM reattachment length for a final time 264.0 when it was able to resolve the secondary vortex at the bottom of the step. The secondary vortex at the bottom of the step covered the region from 0 to 0.14 in the x direction and 0 to 0.14 in the y direction. For Reynolds number 500, a secondary vortex also appears on the upper wall. The horizontal starting location for this vortex is near the horizontal end location of the primary recirculating region (at the base wall).

Table 5.2. Re = 500, reattachment length

grid spacing	grid size	Xr for LBM	Xr for VSM
0.05	601x41	9.3	7.0
0.02	1501x101	9.45	8.15
0.01	3001x201	9.5	7.8

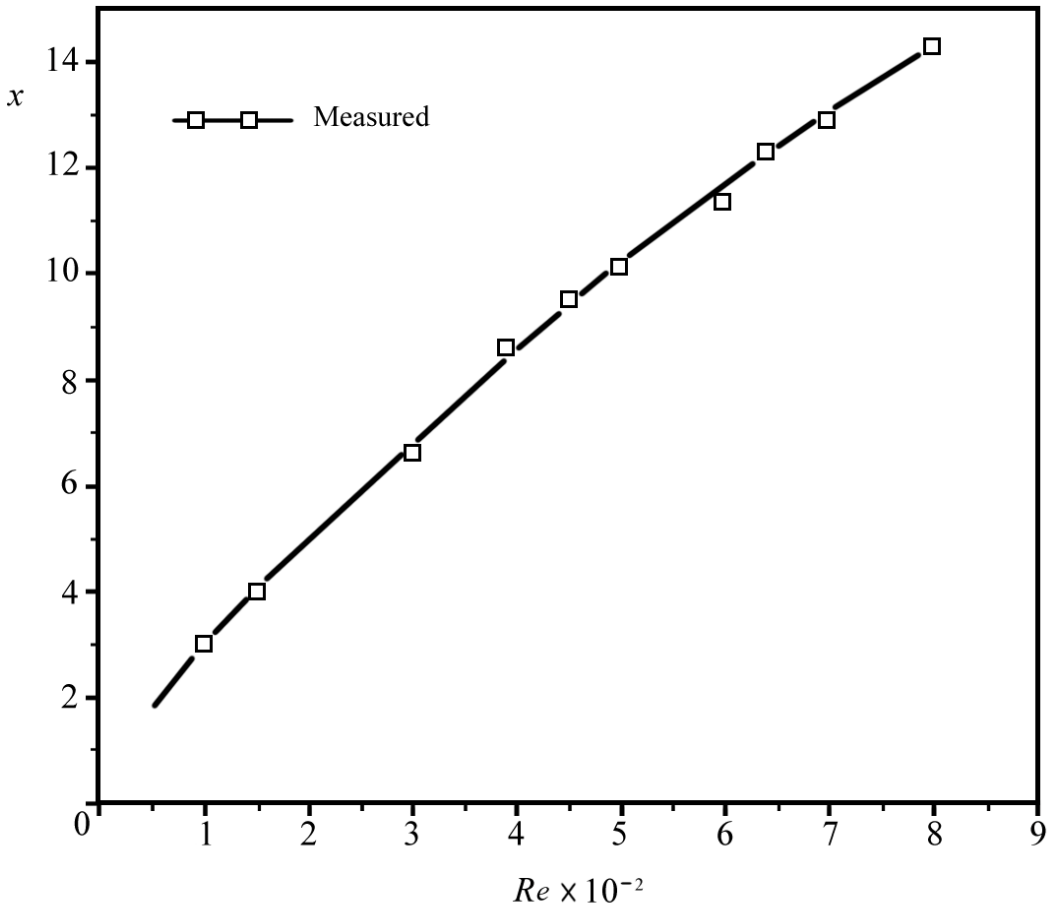


Figure 5.3. Experimental reattachment length versus Reynolds number [52]

5.3.2 Compute Time

To compare the computational performance of both methods, the CPU time was measured. The CPU time is measured until steady state conditions are reached. Steady state conditions are reached according to the criteria specified in the previous section. The "CPU time" is different from the "final time" (Δt multiplied by the number of iterations required for steady state conditions). The CPU time is specified as seconds in wall-clock units. It should be noted here that even with similar time step sizes for both methods, the LBM took considerably more iterations than the vorticity-stream function method. For Re = 150, the LBM final time was 156.0, which is twice as large as the final time for the vorticity-stream function method. For Re = 500, the LBM final time is 264, while the final time for the vorticity-stream function method is 213.0. Tables 5.3 and 5.4 show the CPU time for the LBM and the vorticity-stream function method for grid sizes 601×41 to 3001×201 corresponding to grid spacing from $\Delta x = 0.05$ to $\Delta x = 0.01$, respectively. For the 601×41 grid (the largest grid spacing in the tables), the LBM final time is taken as the final time for the vorticity-stream function method since this grid size does not resolve the secondary vortex at the step bottom.

The CPU time for the vorticity-stream function method is less than the LBM CPU time by a factor 3.2 for $\Delta x = 0.05$ at Re = 150 and a factor 2.89 at Re = 500. The vorticity-stream function method is more efficient due to possessing a much bigger time step for $\Delta x = 0.05$. Increasing grid resolution places a restriction on the time step size for the vorticity-stream function method to retain numerical stability. This is due to the geometry of the backward facing step flow. Previous works have reported that small grid spacing and the existence of high wall-normal velocity in the corner region introduces a severe time step limitation. The LBM's time step size decreases because it is a time-explicit scheme and also due to the increasing Reynolds number. This is obvious from Table 5.3 and Table 5.4, where the LBM time step size is 0.008 for a grid spacing of 0.02 and Reynolds number 150, whereas the time step decreases to 0.006 for the same grid spacing at Reynolds number 500.

With grid spacing 0.02 (or grid size 1501×101), the vorticity-stream function method has a bigger time step size $\Delta t = 0.01$ than the LBM. However, the LBM CPU time is less than that of the vorticity method by a factor of 1.28 for Reynolds number 150 and by a factor 1.6 for Reynolds number 500. This is due to the computational efficiency of the LBM afforded by cache optimization. For grid spacing 0.01 (grid size is 3001×201), both methods possess the same numerical time step size. Here, the LBM (with cache optimization) is about 1.89 times faster than the vorticity-stream function method for Re = 150 and 3.16 times faster for Re = 500. The CPU time reduction factor is less for Re = 150 due to the considerable difference in final times of the two methods at this Reynolds number (in other words, a greater number of iterations are required by the LBM to reach steady state conditions).

Table 5.3. Re = 150, CPU time in seconds

Grid spacing	grid size	LBM time step (and iterations)	VSM time step (and iterations)	LBM CPU time (seconds)	VSM CPU time (seconds)
0.05	601x41	0.0125 (12480)	0.1 (1560)	48	15
0.02	1501x101	0.008 (19500)	0.01 (7600)	938	1204
0.01	3001x201	0.004 (39000)	0.004 (19000)	8397	15901

Table 5.4. Re = 500, CPU time in seconds

Grid spacing	grid size	LBM time step (and iterations)	VSM time step (and iterations)	LBM CPU time (seconds)	VSM CPU time (seconds)
0.05	601x41	0.0125 (21120)	0.09 (2933)	133	46
0.02	1501x101	0.006 (44000)	0.01 (21300)	2114	3392
0.01	3001x201	0.003 (88000)	0.003 (71000)	18816	59469

5.4 Coupling LBM and the Vorticity-Stream Function Method for Backward Facing Step Flow

The results and discussion from the previous section show the vorticity-stream function method to be superior to the LBM when solving on coarse grids, where the vorticity-stream function method possesses a much bigger time step size than the LBM. However, the LBM outperforms the vorticity-stream function method and other traditional methods when it possesses the same numerical time step as the latter methods. Most fluid flow problems involve flow structures, whose resolution requires small grid spacing. For example, the step region of the backward facing step flow problem consists of secondary recirculating eddies or vortices. One such secondary recirculating eddy or vortex lying at the bottom of the step is shown in Figure 5.4. To resolve this eddy, a grid spacing of at least $\Delta x = 0.02$ is required. For better resolution and accuracy, the grid spacing in this region should be $\Delta x = 0.01$. The flow regions consisting of such structures are therefore discretized with high-resolution grids (fine grids), while the remaining flow regions are discretized with coarser grids. The LBM can be used as the solver for those flow regions where the time step of the LBM and the vorticity-stream function method are the same. In regions where the LBM's time step is significantly smaller than that of the vorticity-stream function method, the latter will be used as the solver. A coupled LBM/vorticity-stream function method holds the promise of harnessing the computational efficiency of the LBM and the faster convergence properties of the vorticity-stream function method for solving the backward facing step flow problem.

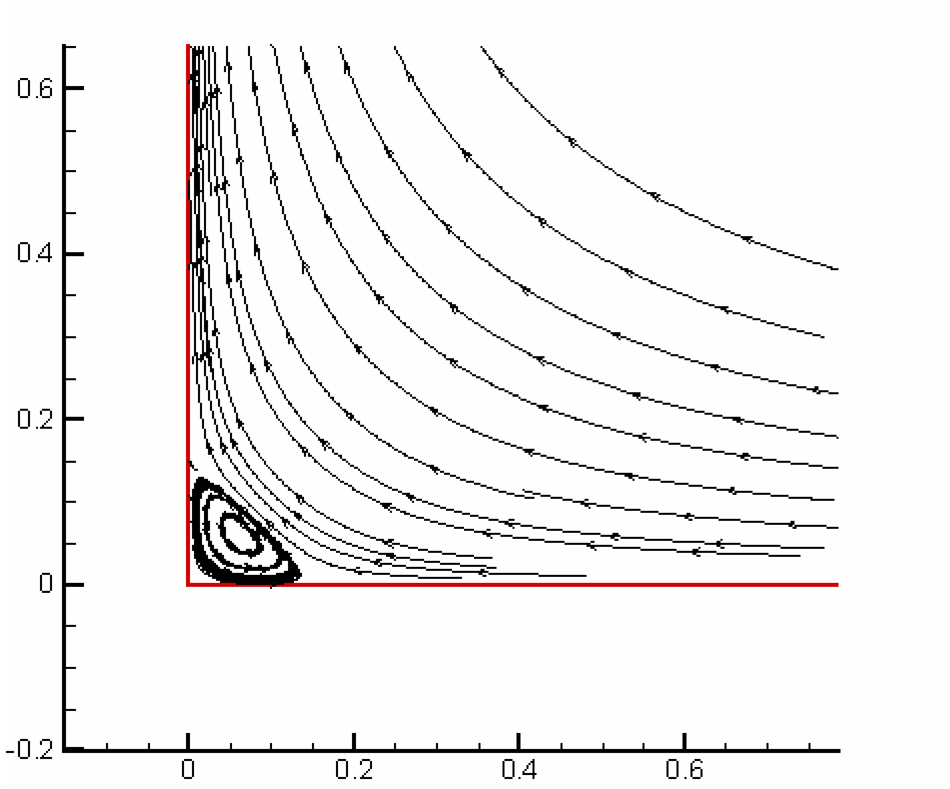


Figure 5.4. Tecplot diagram showing flow and secondary vortex at the bottom of the step for Reynolds number 500

5.4.1 Applying the Coupled Scheme

A high-resolution grid block with a grid spacing of $\Delta x = 0.01$ will cover the region near the step, and a coarse grid block with a grid spacing of $\Delta x = 0.05$ will cover the rest of the domain (schematic given in Figure5.5). This gives a grid refinement factor equal to 5. Section 5.3 shows that the LBM and the vorticity-stream function method possess the same time step for a grid spacing of $\Delta x = 0.01$. Therefore, the LBM will be applied on the fine grid block and the vorticity-stream function method will be applied on the coarse grid block since it can possess as big a time step as $\Delta t = 0.1$ for grid spacing

$\Delta x = 0.05$. To enable transfer of information between the two methods, the two grid blocks overlap at the interface.

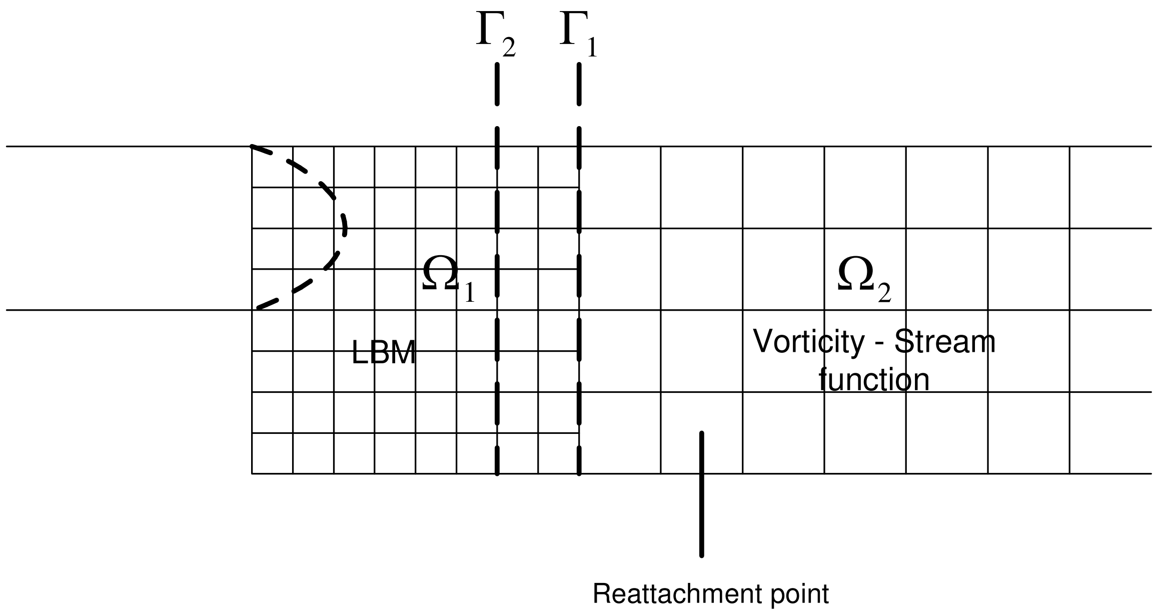


Figure 5.5. Schematic of multiblock grid used for the coupled solver

5.4.2 Coupling Procedure

The coupling procedure should enable information transfer between the adjacent grid blocks. Both the LBM and the vorticity-stream function method are completely separate non-primitive formulations. However, the velocity vector can be computed from both formulations. Hence, the velocity variable will be used to transfer information between both grid blocks. This leads to the following method for communicating information between the grid blocks:

1. The variables on the fine grid block are updated using the LBM. The distribution functions on the interface boundary of the fine grid are assigned the equilibrium distribution function values. To compute the equilibrium distributions (Equation 5.6) at the interface, the velocity vector at the interface is required. This velocity vector is computed from the stream function values in the neighboring coarse grid block using finite difference representations of Equation 5.15. The following is the computation of the velocity vector.

At interior points on the interface boundary Γ_1, the horizontal component of the velocity is given by:

$$u(k, j) = \frac{\psi(k, j+1) - \psi(k, j-1)}{2\Delta y} \tag{5.28}$$

At the base:

$$u(k,1) = \frac{-3\psi(k,1) + 4\psi(k,2) - \psi(k,3)}{2\Delta y} \tag{5.29}$$

At the top:

$$u(k,ny) = \frac{3\psi(k,ny) - 4\psi(k,ny-1) + \psi(k,ny-2)}{2\Delta y} \tag{5.30}$$

The vertical component of the velocity at the interface is given by:

$$v(k, j) = \frac{\psi(k+1, j) - \psi(k-1, j)}{2\Delta x} \tag{5.31}$$

In the above equations, k represents the horizontal direction grid index number of the interface while j represents the vertical direction grid index number. The grid resolution is different for both grid blocks and the fine grid's interface boundary has five grid points for every coarse grid point on that boundary. Since the grid spacing for both blocks is different, interpolation is performed to obtain the interface boundary conditions at the fine grid points that do not have a corresponding coarse grid point. The interface boundary is assumed to lie on a grid line of the neighboring grid block because the grid refinement factor is an integer. Two types of interpolation were performed and both gave the same final results:

Linear interpolation is shown here for the horizontal velocity component at a fine grid point lying between two coarse grid points given by k, j and $k, j+1$:

$$u_{lb}(k, jj) = u(k, j) + \bar{y}\frac{(u(k, j+1) - u(k, j))}{\Delta y} \tag{5.32}$$

$\bar{y}$ is the vertical distance between the fine grid point k, jj and the coarse grid point k, j. Δy is the coarse grid spacing and u_{lb} is the velocity value at the fine grid point.

Quadratic interpolation is shown here for the horizontal velocity component

$$u_{lb}(k, jj) = u(k, j) + \bar{y}\frac{(u(k, j+1) - u(k, j-1))}{2\Delta y} + \bar{y}^2\frac{(u(k, j+1) - 2u(k, j) + u(k, j-1))}{2(\Delta y)^2} \tag{5.33}$$

2. The vorticity-stream function solver computes the vorticity and the stream function variables in the coarse grid region. The interface vorticity and stream function boundary conditions are specified using the velocities computed in Step 1 at the corresponding fine grid locations. Equation 5.5 is used to compute the Γ_2 interface velocities (u, v) from the distribution functions. After the velocities are obtained, the stream function at the interface is obtained through numerical integration of the horizontal velocity component (u) along the vertical direction (y):

$$\begin{aligned}
&\psi(i,1) = 0 \\
&\psi(i,2) = \frac{1}{2}u(i,2)\cdot\Delta y \\
&\psi(i,j) = u(i,j-1)\cdot 2\Delta y + \psi(i,j-2) \qquad j > 2
\end{aligned} \tag{5.34}$$

The vorticity is obtained from the fine grid velocity components using the definition given by Equation 5.14 and applying finite differences.

As seen above, the coupling procedure essentially consists of solving two Dirichlet problems on overlapping subdomains. Steps 1 and 2 are executed alternately until the difference between the current solution and the solution from the previous iteration in the fine grid block is less than a given tolerance. This procedure occurs at very time step. To make full use of the cache-efficient nature of the LBM, Step1 is executed separately for a certain number of time steps ($tdiv$) keeping the same interface boundary condition. A high-level description of this strategy is given below.

Let Ω_1, Ω_2 represent the two overlapping grid blocks (or subdomains), and $\partial\Omega_1$, $\partial\Omega_2$ represent their boundaries. The part of $\partial\Omega_1$ lying in Ω_2 is represented with Γ_1 (artificial boundary or internal boundary or interface boundary of Ω_1). Likewise, Γ_2 represents the interface of Ω_2.

1. Initialize the distribution functions on Ω_1 and the vorticity and stream function on Ω_2
2. Loop for all time steps (until final time or until steady state conditions are reached in the whole domain)
    i. Distribution functions$|\Gamma_1$ are computed using $\psi|\Omega_2$ at Γ_1
    j. Loop for $tdiv$ number of time steps (for LBM cache optimization)

k. Update distribution functions on Ω_1 using the LBM (boundary conditions remain unchanged)

l. End loop for $tdiv$ time steps

m. Compute $\psi | \Gamma_2$ and vorticity$| \Gamma_2$ from the distribution functions$| \Omega_1$ at Γ_2

n. Perform the vorticity-stream function computations to update ψ and the vorticity on Ω_2 (boundary conditions remain unchanged)

3. End loop for all time steps when steady state solution is reached

5.4.3 Method for Comparison

To quantify the computational performance and accuracy of the coupled LBM/vorticity-stream function method, it should be compared with traditional finite difference methods that are applied over the whole domain. The vorticity-stream function method is applied over the whole domain using multiblock gridding. Here, the vorticity-stream function method is implemented separately on all grid blocks in the domain. Unlike the coupled LBM/vorticity-stream function method, there is no requirement to introduce a primitive variable such as velocity to transfer information between neighboring grid blocks. The coupling procedure for the multiblock vorticity-stream function method is shown below.

1. Initialize vorticity and stream function on both Ω_1 and Ω_2

2. Loop for all time steps (until final time or until steady state conditions are reached in the whole domain)

   a. Perform computations using vorticity-stream function solver on Ω_1 and update both vorticity and stream function for one time step

b. Compute the interface boundary values for Ω_2 from the variables belonging to Ω_1; i.e., $\psi|\Gamma_2$ and vorticity$|\Gamma_2$ are computed from $\psi|\Omega_1$ and vorticity$|\Omega_1$ near Γ_2
c. Perform computations using vorticity-stream function solver on Ω_2 and update both vorticity and stream function for one time step
d. Compute the interface boundary values for Ω_1 from the variables belonging to Ω_2; i.e., $\psi|\Gamma_1$ and vorticity$|\Gamma_1$ are computed from $\psi|\Omega_2$ and vorticity$|\Omega_2$ near Γ_1

3. End loop for all time steps when steady state solution is reached

It has been observed from computational experiments that both multiblock methods; i.e., the coupled LBM/vorticity-stream function solver and the multiblock vorticity-stream function solver, take the same number of iterations to converge to the steady state conditions. However, when the solver on the fine grid is repeated for a certain number of time steps before communicating with the neighboring solver, fewer overall iterations are required for convergence. To utilize the cache-optimization properties of the LBM, it is repeated for a certain number of time steps ($tdiv$) as shown in the high-level description. The same is not performed for the vorticity-stream function solver operating on the fine grid block, because the CPU time is more than that for the high-level strategy given above.

5.4.4 Results

The computations were performed on an SGI Onyx 2. The results were computed for Reynolds numbers 150 and 500. For case 1 (Re=150), the fine grid block covers the region from x = 0.0 to x = 7.5. The overlap region is from x = 6.5 to x = 7.5. The coarse grid block covers the region from x = 6.5 to x = 30.0. The grid spacing for the fine grid block is 0.01. Therefore, the size of this grid block will be 751×201. The grid refinement factor is 5. This results in a coarse grid block of size 471×41 corresponding to grid spacing 0.05. Local time stepping was performed by assigning a time step size of 0.003 to the fine grid block and a time step size of 0.1 to the coarse grid block. Table 5.5 shows the reattachment length, CPU time, and number of time iterations for both multiblock methods at Reynolds number 150. Both methods were executed until they reached steady state conditions.

For case 2, computations were performed for Reynolds number 500. Here, the fine grid block extends to x=10.0, so that the reattachment zone falls within the fine grid region. The overlap distance is taken as 2.0. Therefore, the coarse grid block covers the region from x= 8.5 to x= 30.0. Table 5.6 shows the reattachment length, CPU time, and total number of time iterations for the two methods at Reynolds number 500.

Table 5.5. Re = 150, reattachment length, CPU time, and total number of time iterations (Case 1)

	grid spacing	time step
fine grid block	0.01	0.004
coarse grid block	0.05	0.1
	coupled method	VSM
Xr	4.3	4
CPU time (seconds)	1465	4295
time iterations	2100	19000

Table 5.6. Re = 500, reattachment length, CPU time and total number of time iterations (Case 2)

	grid spacing	time step
fine grid block	0.01	0.003
coarse grid block	0.05	0.1
	coupled method	VSM
Xr	9.6	8.35
CPU time (seconds)	6155	17560
time iterations	7300	71000

For Reynolds number 150, the coupled LBM/vorticity-stream function method results in a reattachment length of 4.3, which is similar to the result obtained for the standalone LBM. The multiblock vorticity-stream function method results in a reattachment length of 4, which is the same as that obtained by the standalone vorticity-stream function method. The coupled LBM/vorticity-stream function solver was used without any cache optimization to check for the number of iterations. In this case, the coupled LBM/vorticity-stream function solver was able to resolve the secondary vortex at the bottom of the step for Reynolds number 150 after executing the same number of time iterations (19000) as the multiblock vorticity-stream function method. However, the cache-optimized version of the coupled solver takes only 2100 iterations. The number of iterations is reduced because the LBM performs 12 time steps within every computational cycle as mentioned in Section 5.4.2. The CPU time for the coupled LBM/vorticity-stream function method is less than the CPU time for the multiblock vorticity-stream function method by a factor 2.93.

For case 2 (Re = 500), the fine grid extends up to x=10.0, thus covering the reattachment zone. Here the accuracy of both methods is slightly better than the standalone versions. This is due to faster transfer of information across the domain due to part of the domain being covered with a coarse grid. The coupled LBM/vorticity-stream function method is about 2.85 times faster than the multiblock vorticity-stream function method in this case. Since the fine grid covers a greater region than in case 1, the multiblock vorticity-stream function method performs computations at a greater number of grid points and takes a comparatively higher number of iterations.

5.4.5 Conclusions

From the results for both multiblock methods, the coupled LBM/vorticity-stream function method is the clear winner in terms of both CPU time and accuracy. It is expected that an increase in the region represented by the fine grid block and the LBM will result in a greater factor for reducing CPU time. It is also expected that parallelizing the algorithms will result in a more efficient coupled LBM/vorticity-stream function solver relative to the multiblock vorticity-stream function solver.

5.5 Flow around Cylinder

The results of the multi-solver, multiblock technique for the backward facing step flow show its strength in solving laminar incompressible flows. To further validate the coupled solver, it is implemented for solving the flow around a cylinder. In this case, two dissimilar grids will be used to discretize the flow domain. As discussed in Chapter 2, using a body-fitted grid (a cylindrical grid in this case) near the surface of the body results in accurate and efficient geometry resolution as well as accurate specification of boundary conditions. The Cartesian grid is used to discretize the region away from the cylinder, since it is comparatively easy to generate, especially in flows containing complex geometries. Both grids overlap each other since it is not easy to blend them into each other at the interface. The LBM has demonstrated its superior computational performance on Cartesian grids. However, the LBM cannot be implemented efficiently on cylindrical grids since it loses its local nature due to interpolation [53]. Therefore, a traditional finite difference method (the vorticity-stream function approach) is implemented on the body-fitted grid.

As observed from this discussion and from the section on the flow across a step, a coupled LBM/vorticity-stream function method is efficient on two counts. One is for reasons of geometry and the other is for efficient time stepping (due to multiblock gridding) that improves numerical performance. The coupling procedure is similar to the one given in Section 5.4.2; i.e., the solution procedures on both grids are alternated until steady state conditions are achieved throughout the flow domain. The subsequent sections describe the solution procedures adopted in each subdomain and the interface boundary conditions for each subdomain. Figure 5.6 shows a representation of the decomposed flow domain around the cylinder. The cylinder is represented in black, while Ω_1 represents the subdomain assigned to the vorticity-stream function method, Γ_1, the outer boundary of this subdomain. The LBM solves on the Ω_2 subdomain whose inner boundary is represented by Γ_2. Figure 5.7 shows the gridding applied across the flow domain.

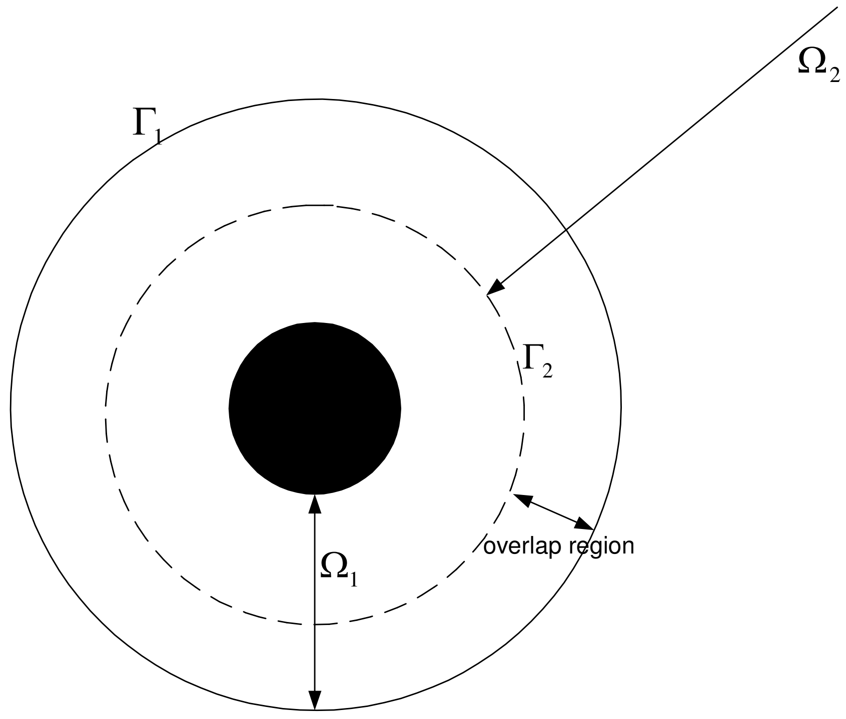


Figure 5.6 Domain decomposition around a cylinder

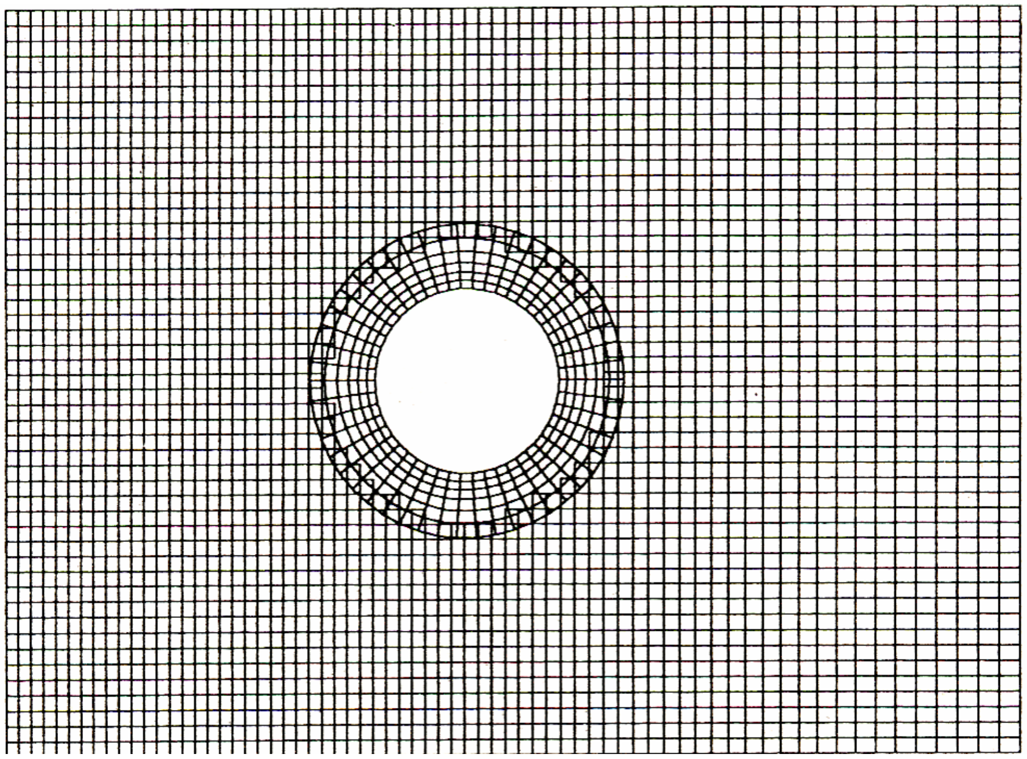


Figure 5.7 Multiblock gridding for flow around cylinder

5.5.1 Vorticity-Stream Function Solver

The vorticity transport and the stream function equations for an incompressible, two-dimensional, unsteady laminar flow around a cylinder can be written as

$$\frac{\partial\omega}{\partial t}+\frac{1}{r}\frac{\partial(V_\theta\omega)}{\partial\theta}+\frac{1}{r}\frac{\partial(rV_r\omega)}{\partial r}=\nu\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\omega}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2\omega}{\partial\theta^2}\right] \tag{5.35}$$

$$\frac{\partial^2\psi}{\partial r^2}+\frac{1}{r}\frac{\partial\psi}{\partial r}+\frac{1}{r^2}\frac{\partial^2\psi}{\partial\theta^2}=-\omega \tag{5.36}$$

In the above equations, ω is the vorticity, ψ is the stream function, and V_θ, V_r are the velocity components in the θ and r directions respectively and are defined as

$$\begin{aligned} V_r &= \frac{1}{r}\frac{\partial\psi}{\partial\theta} \\ V_\theta &= \frac{-\partial\psi}{\partial r} \end{aligned} \tag{5.37}$$

The vorticity is defined as

$$\omega=\frac{1}{r}\left[\frac{\partial(rV_\theta)}{\partial r}-\frac{\partial V_r}{\partial\theta}\right] \tag{5.38}$$

The following log-polar transformation simplifies the above equations and allows the use of a regular rectangular mesh for the numerical treatment of the equations.

$$\begin{aligned} r &= e^{az}; \qquad & z &= \frac{1}{a}\ln r \\ \theta &= ax; \qquad & x &= \frac{1}{a}\theta \end{aligned} \tag{5.39}$$

This transformation will allow a regular rectangular mesh with uniform spacing to represent a polar grid that is exponentially stretched in the radial direction (Figure 5.8). The transformed equations are

$$E_z \frac{\partial \omega}{\partial t} + \frac{\partial \omega}{\partial z}\frac{\partial \psi}{\partial x} - \frac{\partial \omega}{\partial x}\frac{\partial \psi}{\partial z} = \frac{2}{\mathrm{Re}}\left[\frac{\partial^2 \omega}{\partial z^2} + \frac{\partial^2 \omega}{\partial x^2}\right]$$
$$\frac{\partial^2 \psi}{\partial z^2} + \frac{\partial^2 \psi}{\partial x^2} = -E_z \omega \tag{5.40}$$

where $E_z = a^2 e^{2az}$.

Using the new transformation, the radial and tangential velocity components are computed as follows

$$V_r = E_z^{-1/2}\frac{\partial \psi}{\partial x}, \qquad V_\theta = -E_z^{-1/2}\frac{\partial \psi}{\partial z} \tag{5.41}$$

5.5.1.1 Numerical Discretization

To discretize the vorticity-transport equation on a rectangular mesh, second-order accurate central differences are applied to the diffusion and convection terms. The alternating direction implicit method will be used to perform time discretization. This is similar to the discretization used for the vorticity transport equation in the backward facing step flow case. The finite difference analog of the vorticity transport equation is given below

$$\text{Step 1: } E_z \frac{\omega_{i,j}^{n+1/2} - \omega_{i,j}^{n}}{\Delta t/2} - \left(\frac{\partial \psi}{\partial z}\right)_{i,j}^{n} \frac{\bar{\delta}_x \omega_{i,j}^{n+1/2}}{2\Delta x} + \left(\frac{\partial \psi}{\partial x}\right)_{i,j}^{n} \frac{\bar{\delta}_z \omega_{i,j}^{n}}{2\Delta z} = \frac{2}{\mathrm{Re}}\left(\frac{\delta_x^2 \omega_{i,j}^{n+1/2}}{(\Delta x)^2} + \frac{\delta_y^2 \omega_{i,j}^{n}}{(\Delta z)^2}\right)$$
$$\text{Step 2: } E_z \frac{\omega_{i,j}^{n+1} - \omega_{i,j}^{n+1/2}}{\Delta t/2} - \left(\frac{\partial \psi}{\partial z}\right)_{i,j}^{n} \frac{\bar{\delta}_x \omega_{i,j}^{n+1/2}}{2\Delta x} + \left(\frac{\partial \psi}{\partial x}\right)_{i,j}^{n} \frac{\bar{\delta}_z \omega_{i,j}^{n+1}}{2\Delta z} = \frac{2}{\mathrm{Re}}\left(\frac{\delta_x^2 \omega_{i,j}^{n+1/2}}{(\Delta x)^2} + \frac{\delta_y^2 \omega_{i,j}^{n+1}}{(\Delta z)^2}\right)$$
(5.42)

The operators $\bar{\delta}_x$, $\bar{\delta}_y$ represent central differencing for the first-order spatial derivatives and δ_x^2, δ_y^2 represent central differencing for the second-order derivatives in the diffusion term. They are defined as

$$\begin{aligned}
\bar{\delta}_x \omega_{i,j} &= \omega_{i+1,j} - \omega_{i-1,j} \\
\bar{\delta}_y \omega_{i,j} &= \omega_{i,j+1} - \omega_{i,j-1} \\
\delta_x^2 \omega_{i,j} &= \omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j} \\
\delta_y^2 \omega_{i,j} &= \omega_{i,j+1} - 2\omega_{i,j} + \omega_{i,j-1}
\end{aligned} \tag{5.43}$$

The matrices resulting from the application of the above discretization are solved using Gaussian elimination. Figure 5.9 shows a representation of the cylindrical mesh and its corresponding rectangular mesh with the finite difference indices and the boundary conditions specific to the vorticity-stream function method.

5.5.1.2 Initial and Boundary Conditions

The initial conditions are specified as a potential flow solution:

$$\psi_{i,j} = -\left[Ur \sin\theta \left(1 - \frac{1}{r^2} \right) \right]_{i,j} \tag{5.44}$$

where the free stream velocity $U = 1$. The vorticity $\omega_{i,j}$ is taken as zero throughout the subdomain.

The boundary conditions for the vorticity and the stream function will be specified for the wall boundary and the cylindrical grid interface boundary (Γ_1) that lies within the Cartesian grid subdomain. The wall or surface boundary conditions are given as follows:

$$\begin{aligned} \psi_{i,1} &= 0 \\ \omega_{i,1} &= -\frac{2\psi_{i,2}}{\Delta z^2} \end{aligned} \qquad (5.45)$$

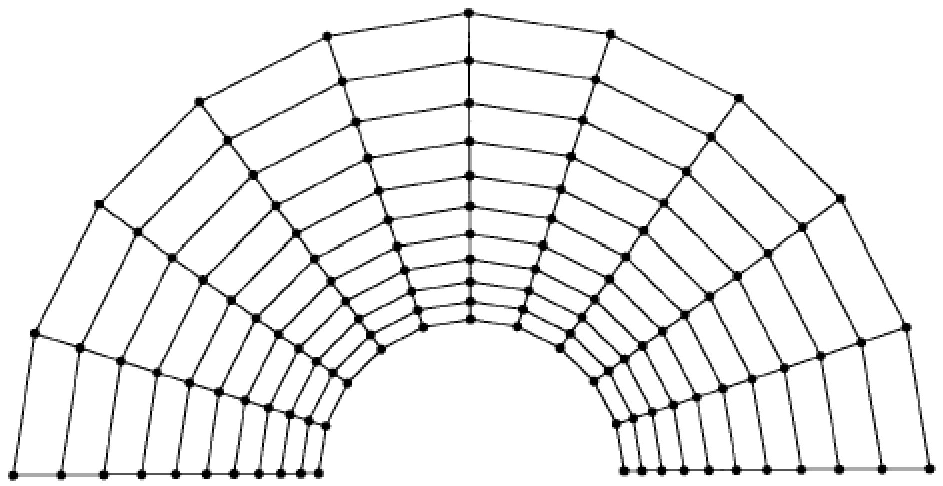
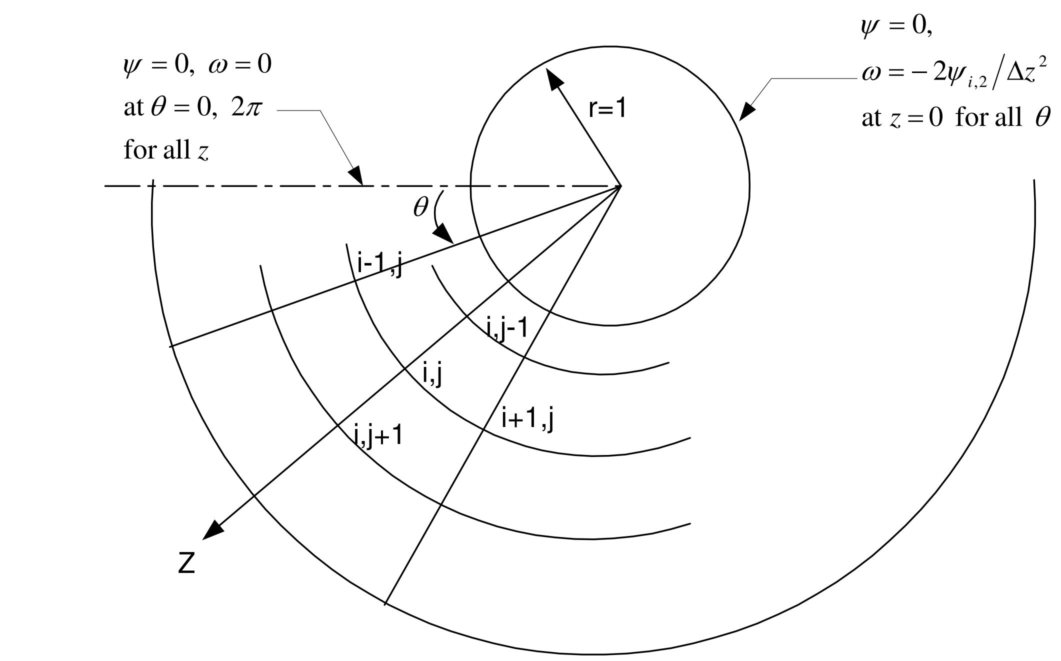


Figure 5.8 Polar grid



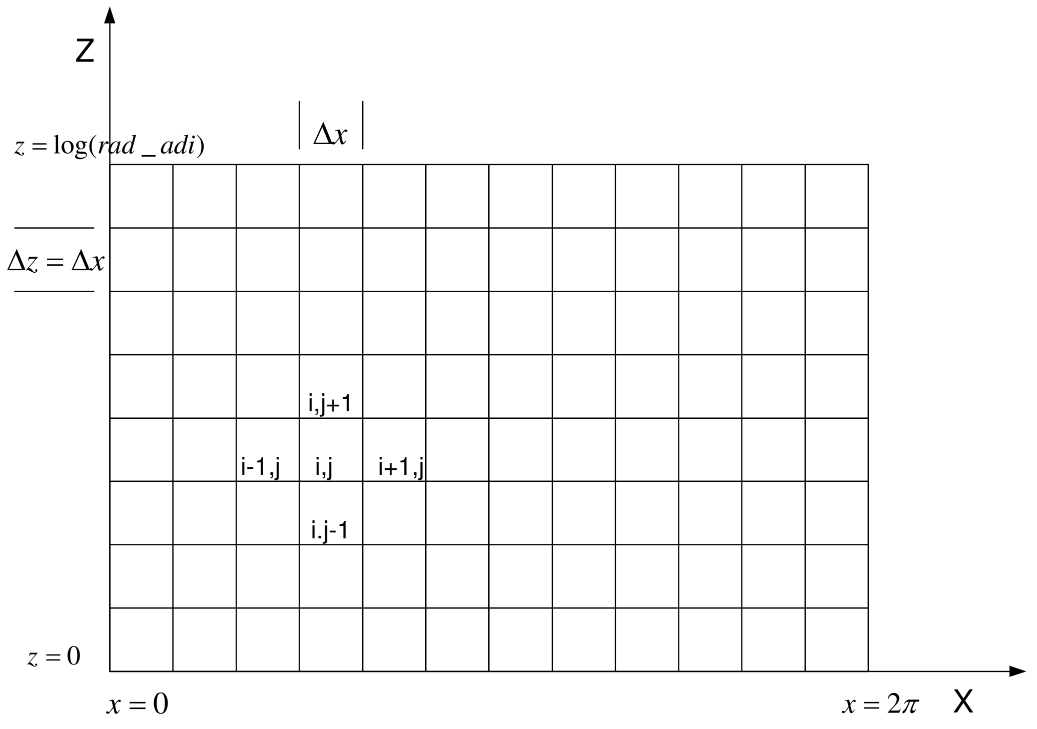


Figure 5.9 Cylindrical mesh and its corresponding rectangular mesh

The outer or interface boundary conditions are specified using values from the lattice Boltzmann computations in subdomain Ω_2. First the Cartesian velocity components at the outer boundary points on the cylindrical grid are computed from the Cartesian grid values using bilinear interpolation. To do this, the Cartesian grid cell containing the cylindrical grid boundary point is located and the velocity values at the four grid cell corners are utilised for interpolation (Figure 5.10).

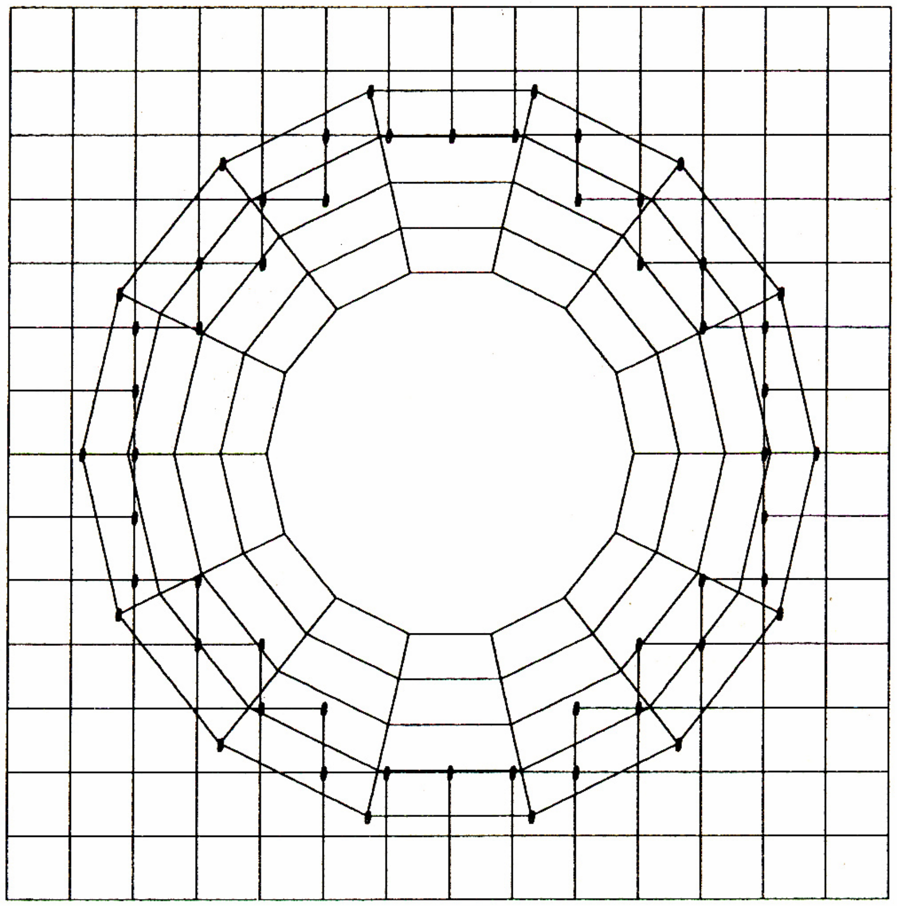


Figure 5.10 Interpolation points

The required radial and tangential velocity components are then computed from the interpolation-obtained Cartesian velocity components. The stream function at the outer boundary points is obtained through numerical integration of the just-computed radial velocity component in the θ direction (refer to Equation 5.34). Initially, the integration was applied from θ=0 to θ=360. However, this resulted in loss of symmetry. Since there is no matching of grid lines between the two grids, interpolation errors are introduced, affecting the numerical integration. Therefore, the numerical integration was performed separately for θ=0 to θ=180 and for θ=360 to θ=180. At θ=180, the result from the two previous integrations was averaged. To obtain the vorticity at the boundary, two different approaches were used. In the first approach, the vorticity was interpolated from the vorticity values on the Cartesian grid. These Cartesian grid vorticity values are obtained by using the velocity values at the Cartesian grid points (Equation 5.14). The second approach obtains the vorticity at the outer boundary points in terms of the cylindrical coordinate system using the following definition:

$$\omega = \frac{1}{r}\left[\frac{\partial(rV_\theta)}{\partial r} - \frac{\partial V_r}{\partial \theta}\right] \tag{5.46}$$

Translated into finite difference terms on the transformed mesh, this would appear as

$$\omega = \frac{(V_\theta)_{i,M+1} - (V_\theta)_{i,M-1}}{2E_z^{1/2}\Delta z} - \frac{(V_r)_{i+1,M} - (V_r)_{i-1,M}}{2E_z^{1/2}\Delta x} \tag{5.47}$$

Here $j = M$ represents the z direction index at the outer boundary. The values for $(V_r)_{i+1,M}$ and $(V_r)_{i-1,M}$ have already been computed as specified above for the numerical integration of the stream function. Both $(V_\theta)_{i,M+1}$ and $(V_\theta)_{i,M-1}$ are computed in a manner similar to the computation of the tangential velocity at the outer boundary, $(V_\theta)_{i,M}$; i.e., they are obtained from the interpolation of the velocity components on the corresponding Cartesian grid cells. Both approaches for computing vorticity were tested and whichever gave more accurate results was finally utilized.

5.5.2 Lattice Boltzmann Solver

The LBM applied to the flow around a cylinder is similar to that applied for the backward step problem. The difference between the two problem cases lies in the boundary conditions to be applied. In this case, since the LBM operates on an outer Cartesian grid, there is no requirement to apply wall boundary conditions. The Cartesian grid is uniformly spaced as shown in Figures 5.7 and 5.10. The inner boundary of the Cartesian grid approximates a circular contour (Γ_2) using the stair-stepped representation (Figure 5.10). The distribution functions at the inner boundary of the Cartesian grid are taken equal to the equilibrium distribution functions at those locations. The computation of the equilibrium distributions at the grid points lying on the inner or interface boundary (Figure 5.10) of the Cartesian grid requires velocity components at those grid points. These velocity components are computed using linear interpolation (in the θ direction) of the radial and tangential velocity components at the cylindrical grid points that are closest to the circular contour approximated by the inner Cartesian grid boundary. Computational experiments showed that using bilinear interpolation produced a greater error than simple linear interpolation along the θ direction on the approximated circular contour.

The far field boundary is specified at a radius of 90 from the center of the cylinder. The far field or outer boundary conditions for the Cartesian grid subdomain (Ω_2) are specified as free stream conditions; i.e., the horizontal velocity component u = 1 and the vertical component v = 0. The distribution functions are specified as equilibrium distributions, which are computed using Equation 5.6.

5.5.3 Results

The results portray the accuracy of the coupled method using the parameters shown in Figure 5.11. These parameters are the length of the recirculating eddy region behind the cylinder (L), the height of the eddy region (b) and the location of the center of the eddy (a). The results have been specified for two cases that differ in grid spacing and location of the interface boundaries for both grids. The results were computed for the cases shown in Table 5.9. The term rad_lb specifies the radius of the circular contour (Γ_2) that is approximated using the stair-stepped approximation of the inner boundary of the Cartesian grid. The term rad_adi specifies the radius of the outer boundary (Γ_1) of the cylindrical grid.

The computations were performed for Reynolds number 40, at which steady state results are obtained. Beyond this Reynolds number, the results are unsteady since vortex shedding will occur. For case 1, the inner boundary of the Cartesian grid lies within the recirculation zone (rad_lb = 4.0). The Cartesian grid spacing is 0.25 throughout the domain. For case 2, the inner boundary of the Cartesian grid lies outside the recirculation zone (rad_lb = 8.0) and the grid spacing is 0.4. The experimental results from Coutanceau and Bouard [54] are represented as CB1977 in Table 5.10. The comparison between the present results and CB1977 shows the coupled method to be quite accurate for the two cases that were computed.

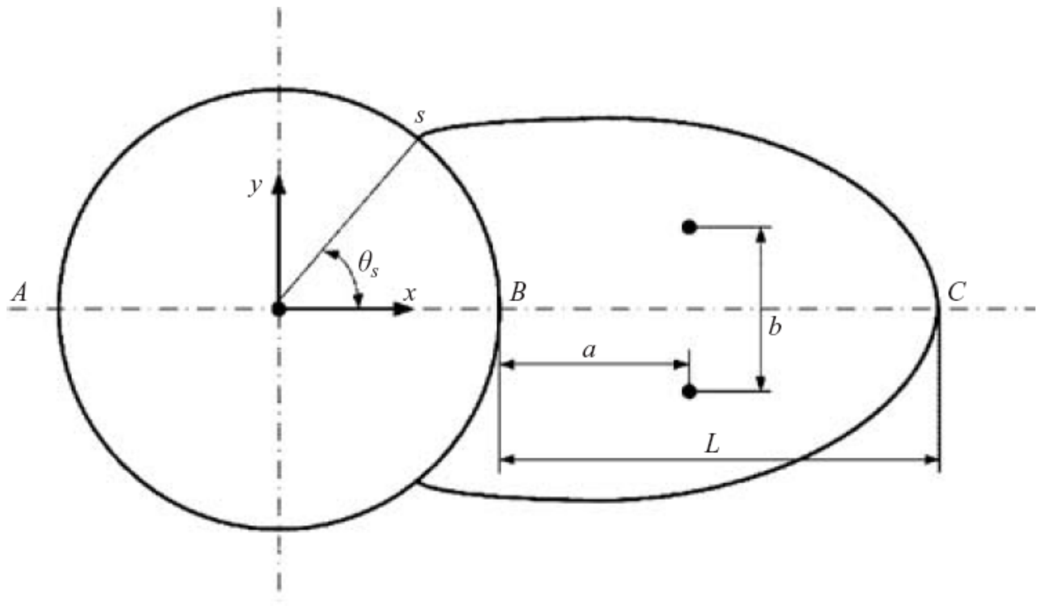


Figure 5.11 Parameters for flow around cylinder

Table 5.9 Parameters for case 1 and case 2

	dx_vort	dx_lb	rad_lb	rad_adi
case 1	Pi/50	0.25	4	9
case 2	Pi/70	0.4	8	12

Table 5.10 Comparison between CB1977 and case 1 and case 2

	L	a	b
CB1977	4.26	1.52	1.19
Present/case 1	4.5	1.45	1.2
Present/case 2	4.3	1.35	1.14

5.5.4 Conclusions for Flow around Cylinder

The interpolation errors arising from the transfer of information between the Cartesian grid and the cylindrical grid are greatly amplified through using velocity as the interpolating variable. This is because neither individual solver in the coupled method computes velocity as the primary variable. For cases where the outer boundary of the cylindrical grid lies within the recirculating zone for Reynolds number 40, inaccurate results are obtained on the cylindrical grid due to the vorticity information being inaccurately conveyed. Therefore, for both the cases shown here, the outer boundary of the cylindrical grid lies at a radius that is outside the recirculating region. Therefore, the computations for flow around the cylinder have been limited to Reynolds number 40, since shedding will occur beyond this vortex and accurate results cannot be expected due to interpolation errors.

CHAPTER 6.

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

Chapter 1 introduces the need for a faster high-fidelity solver for reducing the turnaround time for design using CFD. From the discussion in Chapter 2, it is clear that using Cartesian grids or hybrid Cartesian-prismatic (or Cartesian-cylindrical) grids aids in reducing the cost of grid generation corresponding to changes in design. Therefore, the conclusion is to have the fast high-fidelity solver operate on Cartesian or hybrid Cartesian-prismatic grids.

In Chapter 3, a cache-optimization algorithm was developed to extract the computational strength of the lattice Boltzmann method (LBM). This is possible due to LBM's locally based floating point operations. The comparison performed with standard finite difference methods such as the alternating direction implicit (ADI) method showed the LBM to be about eight times faster for solving the unsteady Burger's equation, while the ADI scheme outperformed the LBM for the steady Burger's equation. The time-explicit nature of the LBM limits the time step size and also speed of information transfer across the domain when solving steady state problems.

In Chapter 4, an improved version of the LBM called the ILBM that was developed to increase the time step size was described. The ILBM was developed by adopting a new spatial discretization for the convection term. The ILBM allowed a time step twice as big as the time step for the original LBM. The CPU time reduction factor was, however, only 1.7 due to the ILBM taking more floating point operations and losing the locality of the floating point operations. A different strategy was adopted to use the LBM for steady state problems, wherein it was coupled with the ADI scheme. The coupled LBM-ADI scheme solved the steady Burger's equation by discretizing the computational domain with two overlapping grid blocks of different resolution. On the coarse grid block, the LBM was able to take the same time step size as the ADI method on the fine grid block. The elliptical nature of the problem allowed the LBM to iterate separately over the coarse grid region, thereby utilizing its computational properties. A comparison between the LBM-ADI solver and the standalone ADI solver computing on the multiblock grid showed the former to be about 4.5 times faster.

Chapter 5 utilized the conclusions from Chapter 4 to build a coupled solver for two-dimensional incompressible fluid flow problems. The vorticity-stream function method was selected to represent the traditional CFD methods because there are only two partial differential equations (PDEs) that need to be solved for the vorticity-stream function formulation. Traditional methods based on the primitive form of the Navier-Stokes equations need to solve at least three PDEs and usually require staggered meshes to compute pressure correctly. The coupled LBM/vorticity-stream function solver was implemented for solving flow across a backward facing step. The coupled method reduced the computation time by a factor of 3 when compared to the standalone vorticity-stream function solver. Parallel computation is expected to significantly increase this factor due to the parallel efficiency of the LBM. The coupled solver was implemented for solving flow around a cylinder. However, interpolation errors that were introduced due to transferring information between the cylindrical and Cartesian grids were further amplified due to a lack of common variables between the methods operating on the two grids. Both the LBM and the vorticity-stream function formulation had to use the velocity vector to transfer information, but the velocity vector was not a dependent variable in either method and had to be computed separately from the given dependent variables.

6.2 Recommendations for Future Work

The vorticity-stream function formulation is useful for only two-dimensional incompressible flows. For three-dimensional computations of fluid flows, the primitive form of the Navier-Stokes equations would be more efficient. The flow-around-cylinder problem has shown that using completely separate formulations with no common dependent variable results in interpolation errors when using a hybrid grid. Therefore, using a primitive formulation of the Navier-Stokes equations as the traditional method in the coupled solver for problems involving hybrid grids in two-dimensional computations is suggested.

BIBLIOGRAPHY

1. Jameson A. Re-engineering the design process through computation. *AIAA Paper No. 97-0641* 1997.

2. Raj P. Aircraft design in the 21st century: implications for design methods. *AIAA Paper (Invited)* 1998.

3. Peace AJ. Maximising the impact of CFD in the design office: ARA's role. *The Aeronautical Journal* 2002; **106**:675-685.

4. Rubbert PE. On the pursuit of value with CFD. *Frontiers of Computational Fluid Dynamics* 1998; World Scientific, Singapore: 417-427.

5. Krajnovic S, Davidson L. Large-eddy simulation of the flow around simplified car model. *SAE Paper No. 2004-01-0227* 2004.

6. Berger M, LeVeque R. An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries. *AIAA Paper No. 89-1930-CP* 1989.

7. Charlton EF, Powell KG. An octree solution to conservation-laws over arbitrary regions (OSCAR). *AIAA Paper No. 97-0198* 1997.

8. Aftosmis MJ, Melton JE, Berger MJ. Adaptation and surface modeling for Cartesian mesh methods. *AIAA Paper No. 95-1725-CP* 1995.

9. Meakin RL. On adaptive refinement and overset structured grids. *AIAA Paper No. 97-1858* 1997.

10. Ham FE, Lien FS, Strong AB. A Cartesian grid method with transient anisotropic adaptation. *Journal of Computational Physics* 2002; **179**:469-494.

11. Chen HC, Yu NJ. Development of a general multiblock flow solver for complex configurations. *8th GAMM Conference on Numerical Methods in Fluid Mechanics* 1989.

12. Aftosmis MJ. Emerging CFD technologies and aerospace vehicle design. *NASA Workshop on Surface Modeling, Grid Generation and Related Issues in CFD* 1995.

13. Oaks W, Paoletti S. Polyhedral mesh generation. *13th International Meshing Roundtable – Virginia, USA* 2004.

14. Karman SL. SPLITFLOW: A 3D unstructured Cartesian/prismatic grid CFD code for complex geometries. *AIAA Paper No. 95-0343* 1995.
15. Melton JE, Pandya SA, Steger JL. 3D Euler solutions using unstructured Cartesian and prismatic grids. *AIAA Paper No. 93-0331* 1993.

16. Wang ZJ, Bayyuk SA. An automated, adaptive, unstructured, Cartesian-prism-based technique for moving-boundary simulations. *6th International Conference on Numerical Grid Generation in Computational Field Simulations* 1998.

17. McMorris H, Kallinderis Y. A hybrid mesh movement strategy for design optimization. *7th International Conference on Numerical Grid Generation in Computational Field Simulations* 2000.

18. Peterson J. The reduced basis method for incompressible flow calculations. *SIAM Journal on Scientific and Statistical Computing* 1989; **10**:777-786.

19. Kennedy MC, O'Hagan A. Predicting the output from a complex computer code when fast approximations are available. *Biometrika* 2000; **87**:1-13.

20. Van Leer B. Computational fluid dynamics: Science or toolbox. *15th AIAA Computational Fluid Dynamics Conference, AIAA-2001-2520* 2001.

21. Kim WT, Jhon MS, VanOsdol John. Vectorized flow network model. *29th International Technical Conference on Coal Utilization and Fuel Systems – Florida, USA* 2004.

22. Ma Z, Jeter SM, Abdel-Khalik SI. Flow network analysis in fuel cells. *Journal of Power Sources* 2002; **108**:106-112.

23. Michal T, Verhoff A. Hybrid computational fluid dynamic algorithms based on analytic and finite volume methods. *AIAA Paper No. 97-0645* 1997.

24. Verhoff A, Chen HH, Cebeci T, Michal T. An accurate and efficient interactive boundary layer method for analysis and design of airfoils. *AIAA Paper No. 96-0328* 1996.

25. Perng CY, Street RL. A coupled multigrid–domain-splitting technique for simulating incompressible flows in geometrically complex domains. *International Journal for Numerical Methods in Fluids* 1991; **13**:269-286.

26. Brakkee E, Wesseling P, Kassels CGM. Schwarz domain decomposition for the incompressible Navier-Stokes equations in general co-ordinates. *International Journal for Numerical Methods in Fluids* 2000; **32**:141-173.

27. Strikwerda JC, Scarbnick, CD. A domain decomposition method for incompressible viscous flow. *SIAM Journal on Scientific Computing* 1993; **14**:49-67.

28. Mendu LN, Parameswaran S. Multi block and multi model based computation of turbulent fluid flow and heat transfer. *American Society of Mechanical Engineers, Heat Transfer Division, (Publication) HTD*, v 318, *Heat Transfer in Turbulent Flows*, 1995, p 99-109.

29. Ikegawa M, Kaiho M, Kato C. FEM/FDM composite scheme for incompressible flow analysis around moving bodies. *American Society of Mechanical Engineers, Fluids Engineering Division (Publication) FED*, v 129, *Multidisciplinary Applications of Computational Fluid Dynamics*, 1991, p 83-89.

30. Nakahashi K, Obayashi S. FDM-FEM zonal approach for viscous flow computations over multiple bodies. *AIAA Paper No. 87-0604* 1987.

31. Ould-Salihi ML, Cottet GH, El Hamraoui M. Blending finite-difference and vortex methods for incompressible flow computations. *SIAM Journal on Scientific Computing* 2000; **22**:1655−1674.

32. Ling G, Wang Y, Ling G. Domain decomposition hybrid method combining finite difference and vortex methods for numerical simulation of bluff body flows. *2nd International Offshore Polar Engineering Conference* 1992; p 237-244.

33. Guermond JL, Huberson S, Shen W-Z. Simulation of 2D external viscous flows by means of a domain decomposition method. *Journal of Computational Physics* 1993; **108**:343-352.

34. Chen S, Doolen GD. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics* 1998; **30**:329-364.

35. Wolf-Gladrow DA. *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*. Springer: Berlin, 2000.

36. Tannehill JC, Anderson DA, Pletcher RH. *Computational Fluid Mechanics and Heat Transfer*. Taylor and Francis: Philadelphia, 1997.

37. Skordos PA. Initial and boundary conditions for the lattice Boltzmann method. *Physical Review E* 1993; **48**:4823-4842.

38. Palmer BJ, Rector DR. Lattice Boltzmann algorithm for simulating thermal flow in compressible fluids. *Journal of Computational Physics* 2000; **161**:1-20.

39. LeVeque RJ. Numerical Methods for Conservation Laws. Birkhauser-Verlag: Basel, 1990.

40. Velivelli A, Bryden KM. Parallel performance of lattice Boltzmann and implicit finite difference approaches to the approximation of two-dimensional diffusion equation. *IMECE Paper No. 2003-41280, 2003 ASME International Mechanical Engineering Congress* 2003.

41. Sterling JD, Chen S. Stability analysis of lattice Boltzmann methods. *Journal of Computational Physics* 1996; **123**:196-206.

42. Velivelli A, Bryden KM. A cache-efficient implementation of the lattice Boltzmann method for the two-dimensional diffusion equation. *Concurrency and Computation: Practice and Experience* 2004; **16**:1415-1432

43. Silicon Graphics, Inc. *Onyx2^{TM} Owners Guide*. SGI: Mountain View, CA, 1998.

44. Radwan SF. On the fourth-order accurate compact ADI scheme for solving the unsteady nonlinear coupled Burgers' equations. *Journal of Nonlinear Mathematical Physics* 1999; **6**:13-34.

45. J. Zhu, Solving Partial Differential Equations on Parallel Computers, World-Scientific, Singapore, 1994.

46. Ferziger JH, Peric M. *Computational Methods for Fluid Dynamics*. Springer: Berlin, 2002.

47. Zhou, M.H., Mascagni, M., and Qiao, A.Y., 1998, "Explicit finite difference schemes for the advection equation," *Preprints on Conservation Laws,* Norwegian University of Science and Technology, Trondheim, Norway.

48. Smith BF, Bjorstad PE, Gropp WD. *Domain Decomposition: Parallel Multilevel Methods For Elliptic Partial Differential Equations*. Cambridge University Press: New York, 1996.

49. Zou Q, Hou S, Chen S, Doolen GD. An improved incompressible lattice Boltzmann model for time-independent flows. *Journal of Statistical Physics*, 1995; **81**:35-48.

50. Roache PJ. *Fundamentals of Computational Fluid Dynamics*. Hermosa: Albuquerque, 1998.

51. Swarztrauber P, Sweet R. Efficient FORTRAN subprograms for the solution of elliptic partial differential equations. *Proceedings of the SIGNUM meeting on Software for partial differential equations* 1975; p 30.

52. Armaly BF, Durst F, Pereira JCF, Schonung B. Experimental and theoretical investigation of backward-facing step flow. *Journal of Fluid Mechanics* 1983; **127**:473-496.

53. He X, Doolen G. Lattice Boltzmann Method on Curvilinear Coordinates System: Flow around a Circular Cylinder. *Journal of Computational Physics* 1997; **134**:306-315.

54. Coutanceau M, Bouard R. Experimental determination of the main features of the viscous flow in the wake of a circular cylinder in uniform translation. Part 1. Steady flow. *Journal of Fluid Mechanics* 1977; **79**:231-256.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation towards my advisor Professor Mark Bryden for his constant guidance and support. I am grateful to him for having given me the opportunity to work with him and for providing the financial support for this work. I am greatly thankful to my committee members Professor Richard Pletcher, Professor James Oliver, Professor Tom I-P. Shih and Professor Richard Hindman for the knowledge I gained from the courses I took under them and for answering many of my research related queries. Special thanks to Dr. James Coyle and Professor Glenn Luecke for taking interest in my work and for helping me resolve some research problems.

I would like to thank all members of the Bryden research group, for taking interest in my work and for their constant encouragement. There are many other people who helped me during the course of this thesis, by providing their work, examples, images etc. Thanks to all of them.

Finally I would like to thank my mother and brother for their support and encouragement to complete my work faster.