Implementation of Multiple Relaxation Time, Viscosity Counteraction, Dynamic Grid Refinement, and Generalized Bounce-Back for the Lattice Boltzmann Method

Final Project

Robert Lee

ME 7751, Spring 2016

Included Files:

C++ LBM solver code inside ‘code’. It is also available at: https://github.com/rlee32/lbmpp

MATLAB R2015b code for post-processing inside ‘code/val’.

How to Use Code:

To compile, go inside the ‘code’ directory using a terminal on a Linux environment that has GNU Make or a similar compilation tool. Then choose one of the following compilation options (the second option likely requires installation of additional dependencies):

1. Compilation without visualization (in parallel with 4 threads): ‘make NOVIS=1 -j4’
2. Compilation with visualization (in parallel with 4 threads): ‘make -j4’

After compiling, run ‘./lbmpp’. Input parameters are defined in ‘settings’ (the executable reads this file).

To reproduce the MATLAB plots, run the corresponding script in the ‘code/val’ directory (change to the directory before running). To generate centerline plots, run ‘centerline_<Re>.m’. To generate spatial error convergence plot, run ‘accuracy.m’. To generate streamline plots, run ‘streamlines_<Re>.m’.

Other Details

To compile the C++ code, I used g++ 4.8.4 on Ubuntu 14.04 LTS (Linux), as well as the on-campus cluster deepthought. My code uses OpenMP for parallelized computation. I use CImg for visualization, so you may need to install additional standard dependencies if you compile with visualization. C++ was used for computational efficiency and object oriented capability (useful for unstructured grids).

Contents

This report is organized as follows:

Glossary of Abbreviations

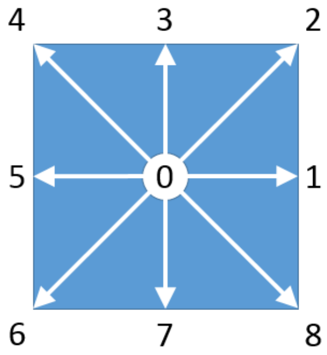
Abbreviation	Definition
BC	Boundary condition
BGK	Bhatnagar-Gross-Krook
DGR	Dynamic Grid Refinement
GBB	Generalized bounce-back
HRE	High Reynolds number
LBM	Lattice-Boltzmann method
LDC	Lid-driven cavity
MRT	Multiple relaxation time
Re	Reynolds number
SRT	Single relaxation time
VC	Viscosity Counteraction

Introduction and Motivation

LBM enables features that are not possible or more difficult in traditional finite-volume methods, such as arbitrary embedded geometries and solution-adaptive DGR that would be useful to a variety of HRE problems. However, the common SRT implementation has poor stability, especially with respect to HRE. When the problem facilitates modulating the viscosity by increasing the characteristic length L, such as in LDC, we can obtain better stability simply by increasing the grid resolution. This comes from the fact that (given that $Re = UL/\nu$) $L \propto \nu$ for constant Re and $Re \propto \nu^{-1}$ when other variables are held constant. However, there is a practical limit to this technique, due to computational resource constraints, and the fact that the range of desirable Re is usually much larger than the range of computationally feasible grid resolutions and determined by the physics of the problem of interest. MRT has been used to dramatically increase the stability of the LBM and the range of Re that can be simulated accurately. An MRT implementation is validated in the present work. To try and further increase stability, viscosity counteraction (VC) is also implemented and tested. A dynamic grid refinement (DGR) scheme is also implemented and validated. Finally, a generalized bounce-back (GBB) scheme for representation of arbitrary geometries is presented, but not validated.

LBM with BGK Collision Operator

Throughout this report, it is assumed that $\Delta t = \Delta x = 1$ and that the following lattice numbering is adopted:



The LBM equation:

$$f_i(\boldsymbol{x} + \boldsymbol{c_i}, t + 1) - f_i(\boldsymbol{x}, t) = \Omega_i$$

SRT BGK:

$$\Omega_i = -\frac{1}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\right)$$

Symbol	Description
$\boldsymbol{x}$	Position vector
i	Particle distribution index
$\boldsymbol{c_i}$	Lattice velocity
Ω_i	Collision operator
τ	Relaxation time
ν	Viscosity
f_i^{eq}	Equilibrium distribution
c_s	Lattice speed of sound
ρ	Macroscopic density
$\boldsymbol{u}$	Macroscopic velocity vector
ω_i	Lattice link weight

The relaxation time can be computed by:

$$\tau = 3\nu + 0.5$$

The equilibrium distribution can be computed by:

$$f_i^{eq} = \omega_i \rho \left[1 + \frac{\boldsymbol{c_i} \cdot \boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{c_i} \cdot \boldsymbol{u})^2}{2c_s^4} - \frac{\|\boldsymbol{u}\|^2}{2c_s^2}\right]$$

For our assumption of $\Delta t = \Delta x = 1$:

$$c_s = \frac{1}{\sqrt{3}}$$

$$f_i^{eq} = \omega_i \rho [1 + 3(\boldsymbol{c_i} \cdot \boldsymbol{u}) + 4.5(\boldsymbol{c_i} \cdot \boldsymbol{u})^2 - 1.5(\|\boldsymbol{u}\|^2)]$$

The macroscopic variables are computed from the distribution functions as follows:

$$\rho = \sum_k f_k$$

$$\rho \boldsymbol{u} = \sum_k f_k \boldsymbol{c_k}$$

The weights ω_i for D2Q9 are given by:

Component	Value
0	4/9
1	1/9
2	1/36
3	1/9
4	1/36
5	1/9
6	1/36
7	1/9
8	1/36

The lattice velocity components are given by:

Component	x-component	y-component
0	0	0
1	1	0
2	1	1
3	0	1
4	-1	1
5	-1	0
6	-1	-1
7	0	-1
8	1	-1

Distribution Update

The spatial and temporal change in the new distribution can be calculated separately and in sequence to arrive at the common SRT explicit algorithm:

1. Collision step: $f_i(\boldsymbol{x}, t+1) = f_i(\boldsymbol{x}, t) - \frac{1}{\tau}\left(f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\right)$
2. Streaming step: $f_i(\boldsymbol{x} + \boldsymbol{c_i}, t+1) = f_i(\boldsymbol{x}, t+1)$

The first step is a computation purely local to each lattice site and the second step transfer distributions to neighboring cells. This scheme is second-order accurate in space and first-order accurate in time.

BCs

Dirichlet boundary conditions are enforced by bounce-back schemes.

Stationary Wall ($u = v = 0$):

> Simple bounce-back schemes are implemented for the stationary walls. Bounce-back is simply implemented as setting interior-facing distributions to its corresponding (negative direction) wall-incident distributions. Components tangent to the wall do not change. Lattice edges (as opposed to lattice centers) reside on physical boundaries.
>
> Care must be taken to apply the bounce-back correctly. It is useful to keep in mind the mechanics of bounce-back: a distribution represented by a lattice link travels a distance of one cell width. Bounce-back distributions should be buffered so as not to interfere (i.e. overwrite) other distributions. Of course, there is a way to implement bounce-back without buffering, but it is not very conducive to parallelization and/or unstructured grids.

Moving Wall:

> For the top lid, $v = 0$ and u is computed according to the desired Mach number. Enforcing this is essentially a bounce-back scheme with modifications on the interior-facing distributions to enforce the prescribed velocity components. For the north wall:

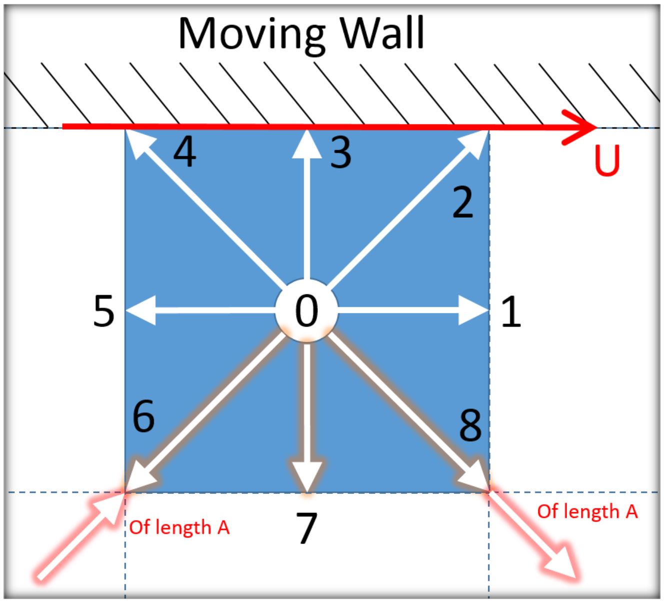
$$f_7 = f_3$$

$$f_8 = f_4 + A$$

$$f_6 = f_2 - A$$

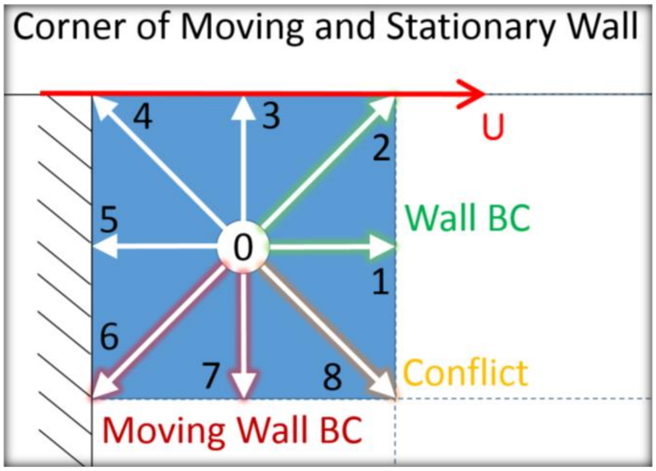
$$A = \frac{\rho^* U}{6}$$

$$\rho^* = f_1 + f_0 + f_5 + 2(f_2 + f_3 + f_4)$$



Corners of Moving and Stationary Wall:

It was found that the corners should be treated carefully. Applying the moving wall BC and overwriting the Conflict lattice link (as shown below) resulted in instability. However, overwriting any of the moving wall lattice links would result in violation of mass conservation. To see this, consider the moving wall diagram above. The addition and subtraction of same-length lattice link vectors results in zero net mass addition. Removing one of these components would result in overall addition or subtraction of mass. Therefore, stationary wall BC was applied to both sides of corner cells in the present implementation.

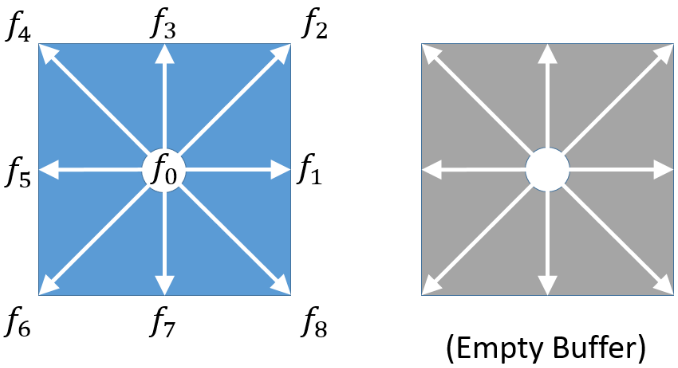


Implementation of Bounce-Back

It is easy to overlook when and how components are streamed, but it is very important for ensuring that no components are overwritten, or streamed more than once per time step. The present implementation uses a buffer to hold streamed and bounced-back distribution values for correctness, as well as to facilitate parallel computation.

Suppose BCs and streaming updates the vector of particle distribution values from $\boldsymbol{f}$ to $\boldsymbol{f}$', and that neighboring distributions at the same time level as $\boldsymbol{f}$ are denoted by $\boldsymbol{f^n}$, where n is the lattice direction pointing towards the neighbor. Let us consider a cell adjacent to a northern wall. Just after collide step, we have:

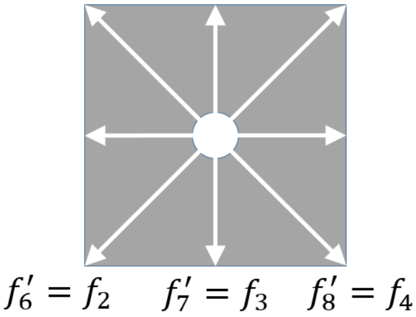
After Collision:



(Empty Buffer)

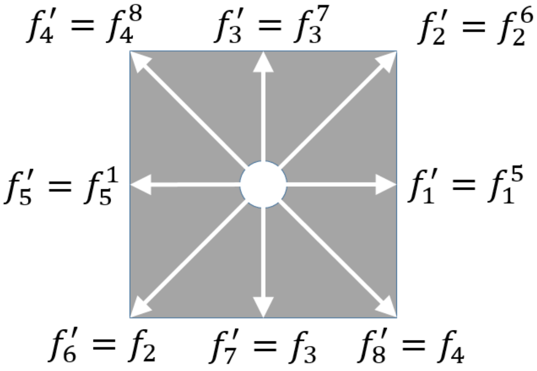
Now the values of $\boldsymbol{f}$ are not modified until a final update from the buffer at the end of the stream step. After wall bounce-back:

Buffer after bounce-back (north wall)



Notice that if bounce-back components had not been buffered and thereby inactivated in regard to the streaming process, they would have been incorrectly streamed again. After neighbor streaming:

Buffer after streaming:



Notice that had we not used a buffer, values could be overwritten. In a regular, structured arrangement of lattice cells, we can perform the streaming step even in parallel without a buffer. However, the present implementation takes into account an unstructured arrangement of lattice cells of different sizes, so a buffer seems necessary to facilitate parallel streaming.

Finally, the values from the buffer are simply copied over to promote $\boldsymbol{f}$ to $\boldsymbol{f}'$**.**

General Algorithm

1. Initialization
   a. Initialize particle distributions from equilibrium distributions computed from macroscopic values
2. Solver loop
   a. Collide
   b. Apply BC (buffer bounce-backs so they do not get overwritten)
   c. Stream

Parameterization

There are four main parameters to completely specify each simulation:

1. Grid resolution
2. Mach number
3. Physical Length
4. Desired Reynolds number

The Mach number is an important parameter because too high of a Mach number leads to a breakdown of the incompressible assumption, so it must be kept low. With other parameters fixed, the Mach number has a positive linear relationship with the relaxation time. Physical length must be consistent with the expected physics and therefore must be specified by the user. For example, the lid-driven cavity problem takes the length to be the dimension of the domain. However, an airfoil simulation would take the chord of the airfoil to be the characteristic length, which is much smaller than the domain dimension. For the lid-driven cavity, since $\Delta x = 1$, the physical length is tied to the grid resolution. Finally, the desired Reynolds number allows us to compute the proper viscosity.

MRT

The LBM with BGK operator and single relaxation time (SRT) is a common implementation due to its simplicity, but suffers from stability problems. MRT is commonly used to greatly enhance the stability of LBM, as shall be demonstrated. In MRT, the collision step happens in the moment space (with macroscopic variables) instead of with the distribution values. This entails transformation of the distribution values into the moment space, then extracting back new distribution values. MRT modifies only the collision step of the traditional SRT algorithm.

(SRT)	$f_i(\boldsymbol{x}+\boldsymbol{c_i},\ t+1) = f_i(\boldsymbol{x},t) - \frac{1}{\tau}\left(f_i(\boldsymbol{x},t) - f_i^{eq}(\boldsymbol{x},t)\right)$
(MRT)	$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{c_i},\ t+1) = \boldsymbol{f}(\boldsymbol{x},t) - M^{-1}\hat{S}\left(\boldsymbol{m}(\boldsymbol{x},t) - \boldsymbol{m}^{\boldsymbol{eq}}(\boldsymbol{x},t)\right)$

MRT collision step may be described generally as:

1. Transform particle distributions to moment space: $M\boldsymbol{f} = \boldsymbol{m}$
2. Collide in moment space: $\boldsymbol{m}(\boldsymbol{x},t) - \boldsymbol{m}^{\boldsymbol{eq}}(\boldsymbol{x},t)$
3. Extract back out the particle distributions via: $M^{-1}\hat{S}$

Notice that compared to SRT, the relaxation frequency $1/\tau$ is replaced by $M^{-1}\hat{S}$ and multiplies the equilibrium difference in moment m_i instead of particle distribution function f_i. The matrix M transforms particle distribution functions into moment space ($M\boldsymbol{f} = \boldsymbol{m}$). For the lattice link indexing defined before, the moment transformation matrix M is:

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -4 & -1 & 2 & -1 & 2 & -1 & 2 & -1 & 2 \\ 4 & -2 & 1 & -2 & 1 & -2 & 1 & -2 & 1 \\ 0 & 1 & 1 & 0 & -1 & -1 & -1 & 0 & 1 \\ 0 & -2 & 1 & 0 & -1 & 2 & -1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 0 & 0 & 1 & -2 & 1 & 0 & -1 & 2 & -1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \end{bmatrix}$$

The relaxation transformation $\hat{S}$ is given by:

$$\hat{S} = diag(s_\rho, s_e, s_\epsilon, s_\chi, s_q, s_\chi, s_q, s_\nu, s_\nu)$$

The moment equilibrium distributions are given by:

$$m^{eq} = \rho \begin{bmatrix} 1 \\ -2 + 3u^2 \\ 1 - 3u^2 \\ u_x \\ -u_x \\ u_y \\ -u_y \\ u_x^2 - u_y^2 \\ u_x u_y \end{bmatrix}$$

There is some freedom in choosing some of these parameters. The values used in the present implementation were $s_\rho = s_\epsilon = s_\chi = 1.0; s_e = s_q = 1.2$. The s_ν values are simply the relaxation frequencies $1/\tau$.

MRT adds extra computation compared to SRT, however the overall speed of the present implementation was essentially unaffected. This is presumably due to the computational bottleneck being memory bandwidth instead of actual computation (due to the unstructured interrelationship of fluid cells in the present implementation). Also, the matrices were collapsed and reduced as much as possible to non-matrix equivalent forms.

VC

VC comes from a manipulation of the viscosity term in the Navier Stokes equations, along with the proposition that high-order truncation errors play a significant role in numerical stability relating to the viscosity term. Let us introduce a term ν_c that represents a ‘buffer viscosity’.

$$\frac{\partial(\rho u_i)}{\partial t}+\frac{\partial(\rho u_i u_j)}{\partial x_j}=-\frac{\partial p}{\partial x_i}+(\nu+\nu_c)\frac{\partial}{\partial x_j}(2\rho S_{ij})-\nu_c\frac{\partial}{\partial x_j}(2\rho S_{ij})$$

One can see that the term $-\nu_c\frac{\partial}{\partial x_j}(2\rho S_{ij})$ can be combined with $(\nu+\nu_c)\frac{\partial}{\partial x_j}(2\rho S_{ij})$ to recover the viscous term in the original equation. However, treating $-\nu_c\frac{\partial}{\partial x_j}(2\rho S_{ij})$ in the discretization with high-order (for example 4th-order) schemes can improve stability without affecting accuracy, given a proper (not too large) ν_c value.

This new VC term takes the form of a body force that can easily be added to MRT as follows:

$$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{e_i}\Delta t,\ t+\Delta t)=\boldsymbol{f}(\boldsymbol{x},t)-M^{-1}S[\boldsymbol{m}(\boldsymbol{x},t)-\boldsymbol{m^{eq}}(\boldsymbol{x},t)]+\Delta t\boldsymbol{F}$$

Where

$$F_i=\frac{1}{2}[g_i(\boldsymbol{x},t)+g_i(\boldsymbol{x}+\boldsymbol{e_i}\Delta t,t+\Delta t)]$$

$$g_i=\omega_i\left\{\boldsymbol{B}\cdot\left[\frac{\boldsymbol{e_i}-\boldsymbol{u}}{c_s^2}+\frac{(\boldsymbol{e_i}\cdot\boldsymbol{u})\boldsymbol{e_i}}{c_s^4}\right]\right\}$$

$$B_i=-\nu_c\frac{\partial}{\partial x_j}(2\rho S_{ij})$$

Simplifying for D2Q9 and $\Delta t=\Delta x=1$:

$$g_i=\omega_i\{\boldsymbol{B}\cdot[3(\boldsymbol{e_i}-\boldsymbol{u})+9(\boldsymbol{e_i}\cdot\boldsymbol{u})\boldsymbol{e_i}]\}$$

$$g_i=3\omega_i\left\{\begin{bmatrix}B_x\\B_y\end{bmatrix}\cdot\left[\begin{bmatrix}e_{ix}-u\\e_{iy}-v\end{bmatrix}+3(e_{ix}u+e_{iy}v)\begin{bmatrix}e_{ix}\\e_{iy}\end{bmatrix}\right]\right\}$$

$$g_i=3\omega_i\begin{bmatrix}B_x\\B_y\end{bmatrix}\cdot\begin{bmatrix}e_{ix}-u+3(e_{ix}u+e_{iy}v)e_{ix}\\e_{iy}-v+3(e_{ix}u+e_{iy}v)e_{iy}\end{bmatrix}$$

$$g_i=3\omega_i\begin{bmatrix}B_x\\B_y\end{bmatrix}\cdot\begin{bmatrix}e_{ix}+3e_{ix}^2u+3e_{ix}e_{iy}v-u\\e_{iy}+3e_{ix}e_{iy}u+3e_{iy}^2v-v\end{bmatrix}$$

$$g_i=3\omega_i\left[B_x\left(e_{ix}+3e_{ix}^2u+3e_{ix}e_{iy}v-u\right)+B_y\left(e_{iy}+3e_{ix}e_{iy}u+3e_{iy}^2v-v\right)\right]$$

For the each of the D2Q9 components, we have:

$$g_0=-\frac{4}{3}\left[B_xu+B_yv\right]$$

$$g_1=\frac{1}{3}\left[B_x(1+2u)-B_yv\right]$$

$$g_2 = \frac{1}{12}\left[B_x(1+2u+3v)+B_y(1+3u+2v)\right]$$

$$g_3 = \frac{1}{3}\left[-B_x u + B_y(1+2v)\right]$$

$$g_4 = \frac{1}{12}\left[B_x(-1+2u-3v)+B_y(1-3u+2v)\right]$$

$$g_5 = \frac{1}{3}\left[B_x(-1+2u) - B_y v\right]$$

$$g_6 = \frac{1}{12}\left[B_x(-1+2u+3v)+B_y(-1+3u+2v)\right]$$

$$g_7 = \frac{1}{3}\left[-B_x u + B_y(-1+2v)\right]$$

$$g_8 = \frac{1}{12}\left[B_x(1+2u-3v)+B_y(-1-3u+2v)\right]$$

Notice in the F_i expression that the body force term at the next time step is needed. This is problematic, since it is an implicit term. In order to circumvent this, we can use a steady-flow approximation:

$$g_i(\boldsymbol{x}+\boldsymbol{e_i}\Delta t, t+\Delta t) \approx g_i(\boldsymbol{x}+\boldsymbol{e_i}\Delta t, t)$$

VC Discretization Schemes

4th-order discretization schemes were used whenever possible to compute the force term derivative $\frac{\partial}{\partial x_j}\left(2\rho S_{ij}\right)$. The strain rate tensor values $S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right) = \frac{3s_v}{2\rho}\Sigma_i\left[f_i - f_i^{eq}\right]e_{ix}e_{iy}$ were computed at lattice sites, then differenced. A 4-point centered 4th order accurate stencil was used for cells with at least two neighbors on both sides:

$$\frac{\partial \phi}{\partial x} = \frac{-\phi_{i+2}+8\phi_{i+1}-8\phi_{i-1}+\phi_{i-2}}{12\Delta x}$$

A 5-point side-biased 4th order accurate stencil was used for cells with only one neighbor on the left side (and similarly for only one on right side):

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+3}-6\phi_{i+2}+18\phi_{i+1}-10\phi_i-3\phi_{i-1}}{12\Delta x}$$

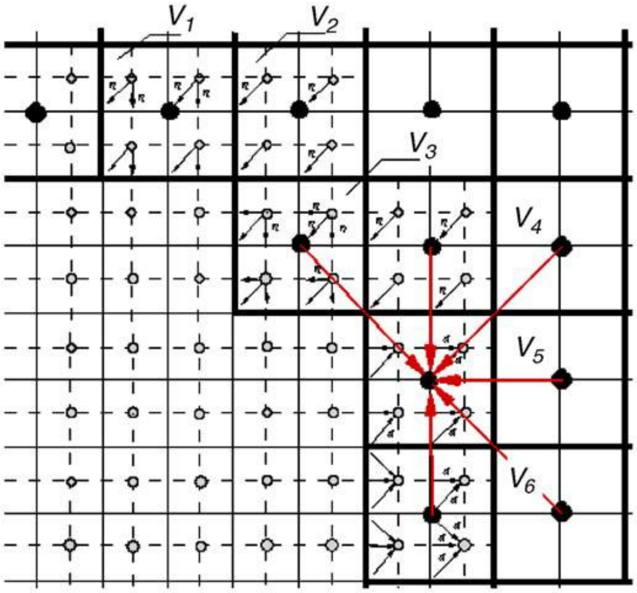
Finally, a 3-point one-sided 2nd order accurate stencil was used for cells with no neighbors on one side. Below is the formula for a cell with no left neighbors.

$$\frac{\partial \phi}{\partial x} = \frac{-3\phi_i+4\phi_{i+1}-\phi_{i+2}}{2\Delta x}$$

The expressions for different directions are straightforward to derive from the above expressions.

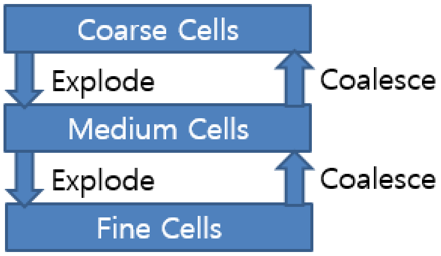
DGR

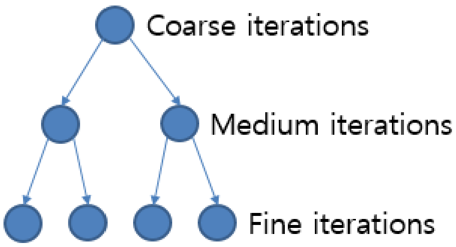
Cells of different sizes can be implemented in the same mesh using streaming interfaces between cells of different sizes. These interface cells are actually an overlap of coarse cells participating in the fluid simulation and 'ghost' fine cells that do not collide. The coarse interface layer maps its own values to the ghost fine cells. This procedure is called 'explode' in the literature. The coarse cells are streamed to by the fine cells by averaging relevant distribution directions, which are the directions streamed from fine cells to the ghost fine cells. The averaging of the fine-cell contributions to make the new coarse particle distribution is called 'coalesce' in the literature. The diagram below shows an example of an overlapping layer and lattice links that are streamed in each layer of the interface.



If the different lattice sizes are to be made consistent with each other, finer cells must be iterated multiple times for every coarse-cell iteration. Specifically every level of refinement in a quad tree grid requires twice as many iterations as the previous layer. To maintain consistency in the physics, the viscosity is doubled with every refinement level (and the relaxation parameters are of course computed accordingly).

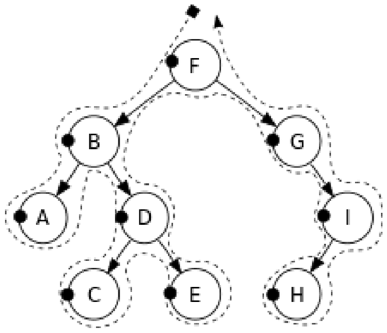
The order of accuracy of the interface depends on the explosion process. A uniform explosion, where all fine cells are homogeneously assigned the coarse cell's distribution values, results in a 1st-order accurate scheme. This scheme is used in the present implementation. A 2ndorder-accurate interface can be achieved by linear explosion, which is the assignment of values to fine cells according to some spatial gradient.

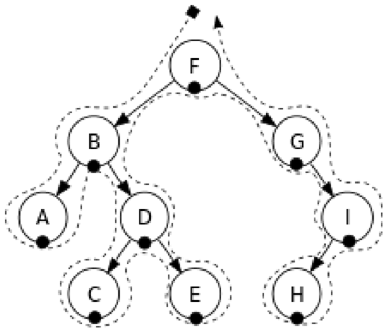


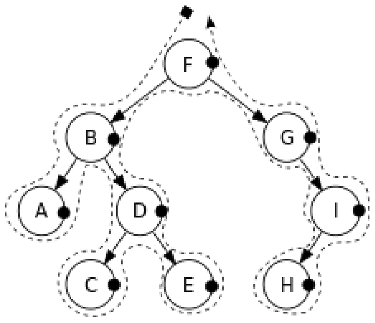


If the iterations are to be represented as a full binary tree as above, we can deduce the proper order of execution of every refinement level with respect to another. The table below summarizes the proper ordering. Different steps of the LBM iteration must be done either in pre-order or post-order.

Traversal	Diagram	Sample Code	LBM Steps

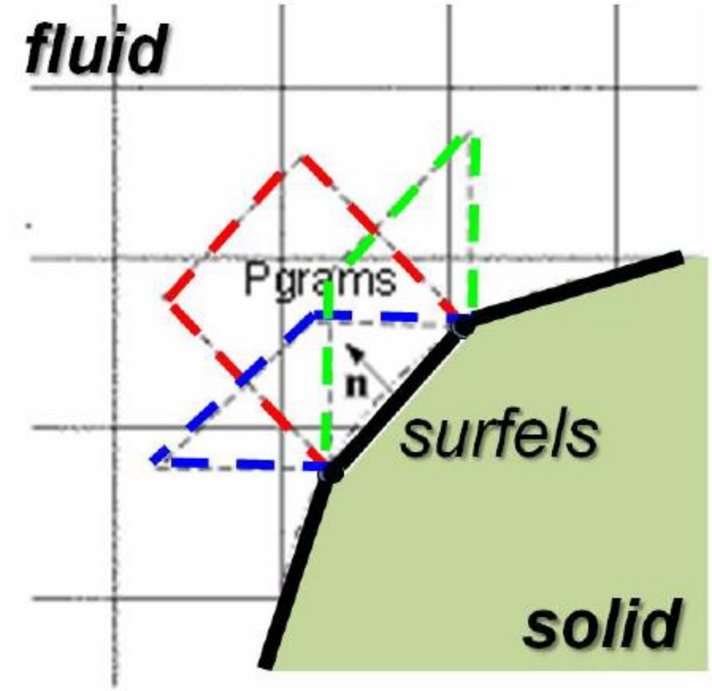
Pre-Order		void recurse(n)
		{
		...
		some_function(n);
		recurse(n);
		recurse(n);
		}
			Collide
			Explode

In-Order		void recurse(n)
		{
		...
		recurse(n);
		some_function(n);
		recurse(n);
		}	

Post-Order		void recurse(n)
		{
		...
		recurse(n);
		recurse(n);
		some_function(n);
		}
			Stream
			Coalesce

GBB

GBB is implemented via ‘surfels’ and ‘pgrams’. Surfels, as in surface elements, are simply the representation of the geometry; in 2D they are simply line segments. Pgrams, as in parallelograms, are extrusions of the surfels that represent the area traveled by wall-incident and bounced-back particles.



Correct interpretation and implementation of pgrams and their interactions with the fluid cells requires a volumetric approach. The different areas of cut cells must be taken into account when streaming. Cells can receive bounced-back particles as well as streamed particles for the same lattice direction. When particles are streamed into pgrams by an overlapped cell, a portion of the cell’s particles is transferred to the pgram (and other overlapping pgrams if present), and the other portion is streamed as normal to adjacent cells.

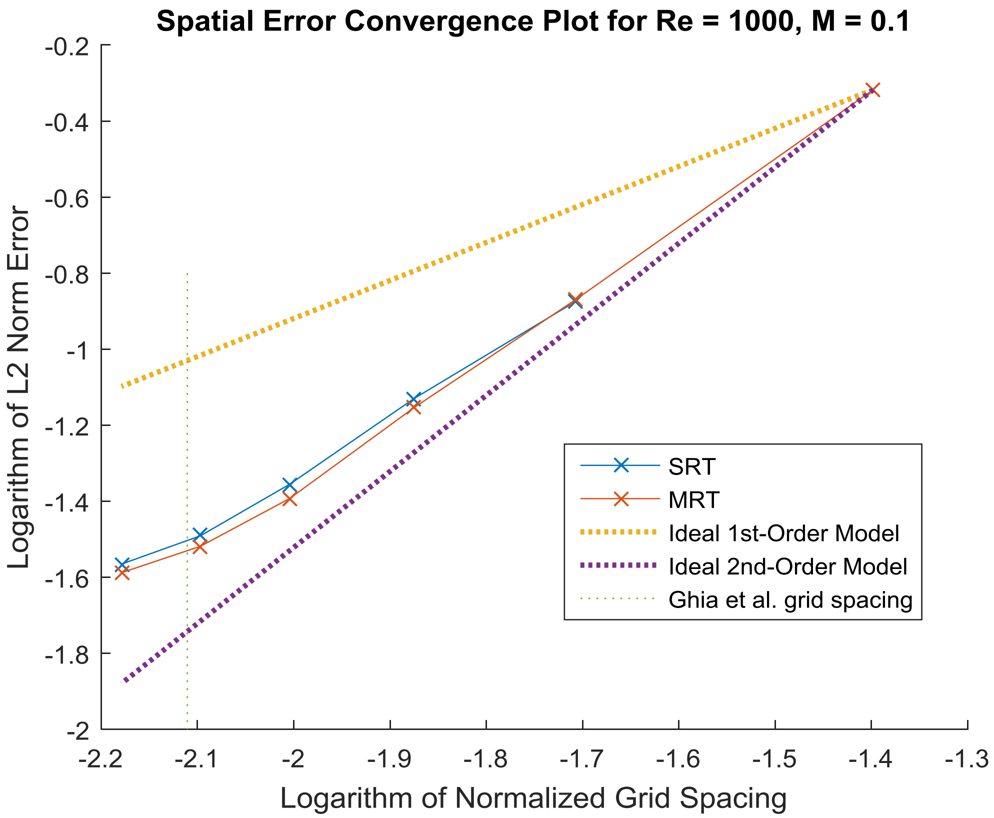
A pgram is generated for every lattice link direction that can impinge upon a surfel. Therefore there are 3 or 4 pgrams to every surfel (3 if the surfel is aligned with a lattice link, 4 otherwise). The extrusion of the pgram is of course in the direction opposite to the impinging lattice link. The length of the extrusion is equal to the magnitude of $\Delta t \boldsymbol{c_i}$. Pgrams reduce to the typical wall bounce-back BC when the surfel aligns with cell boundaries, and the size of the surfel is equal to the cell dimension.

GBB consists of 2 main steps for every pgram:

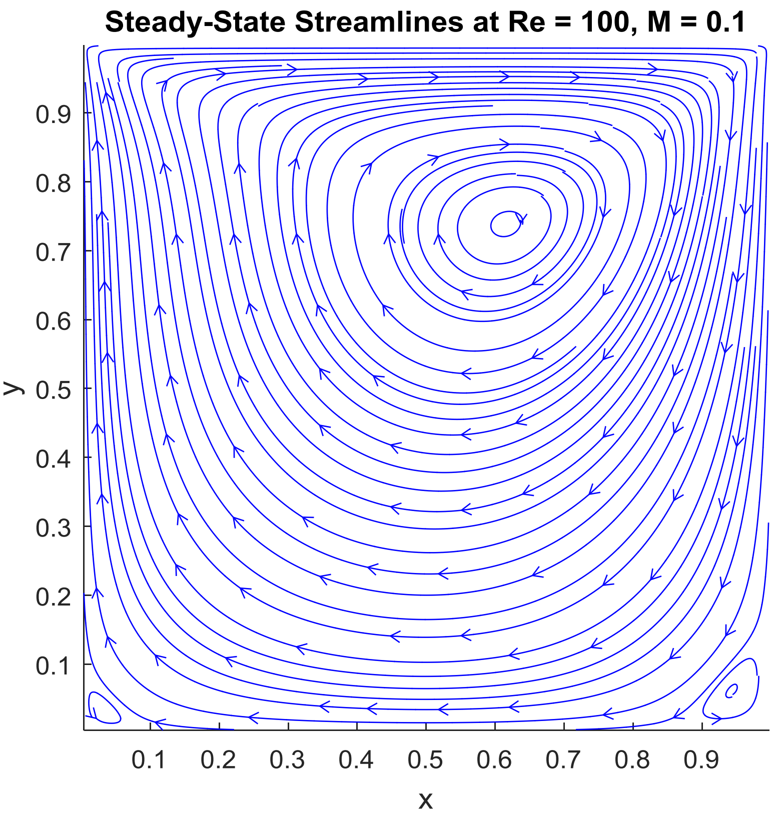
1. Gather: overlap-area-weighted average of incoming particle distributions.
2. Scatter: for 1st-order scheme, redistribute the particles homogeneously among overlapped cells, and in proportion to the overlap area.

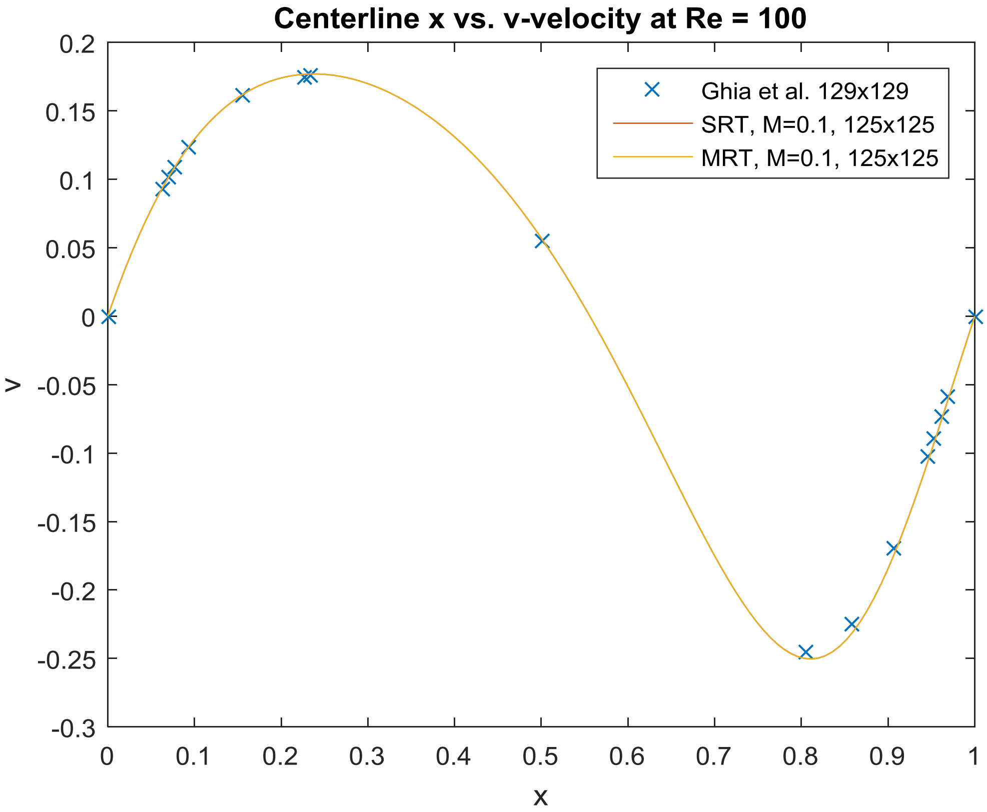
MRT Validation

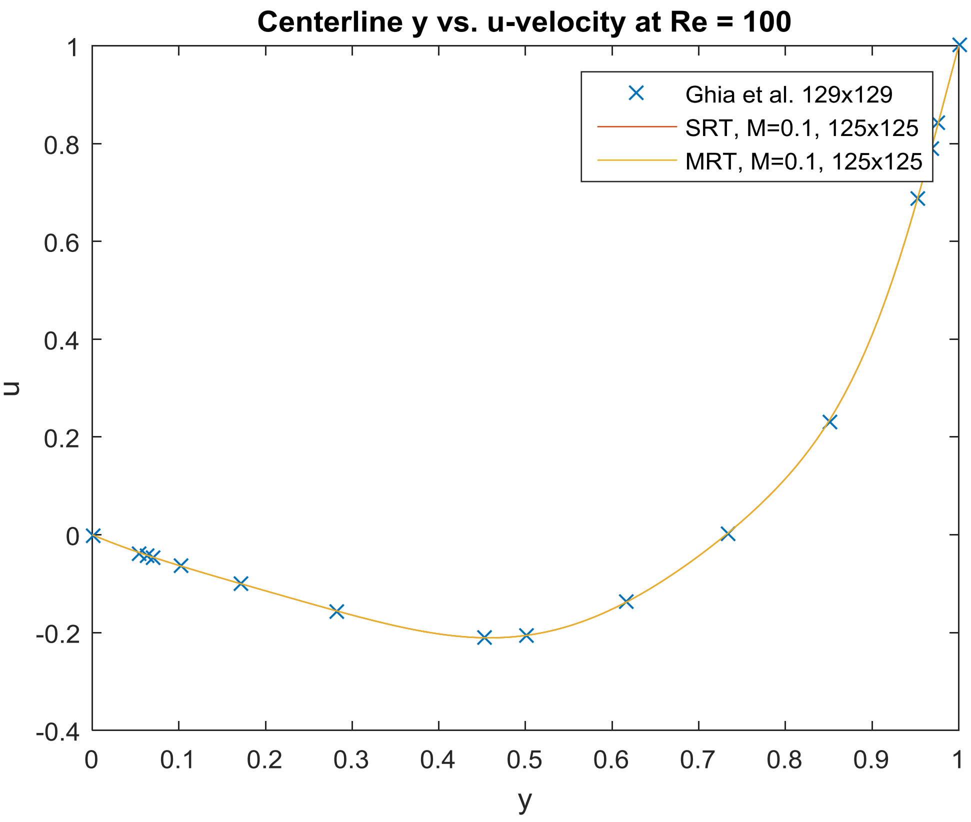
The present implementation results are compared with reference LDC data from Ghia et al. Before showing the streamlines and centerline velocity profiles, the order of accuracy is first verified as 2nd order (or nearby):

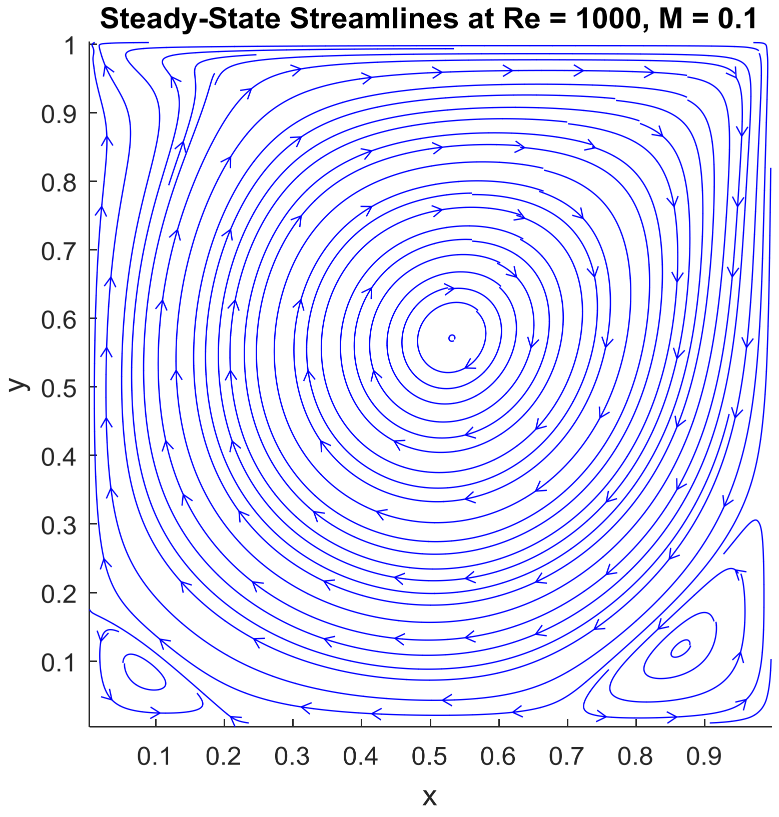


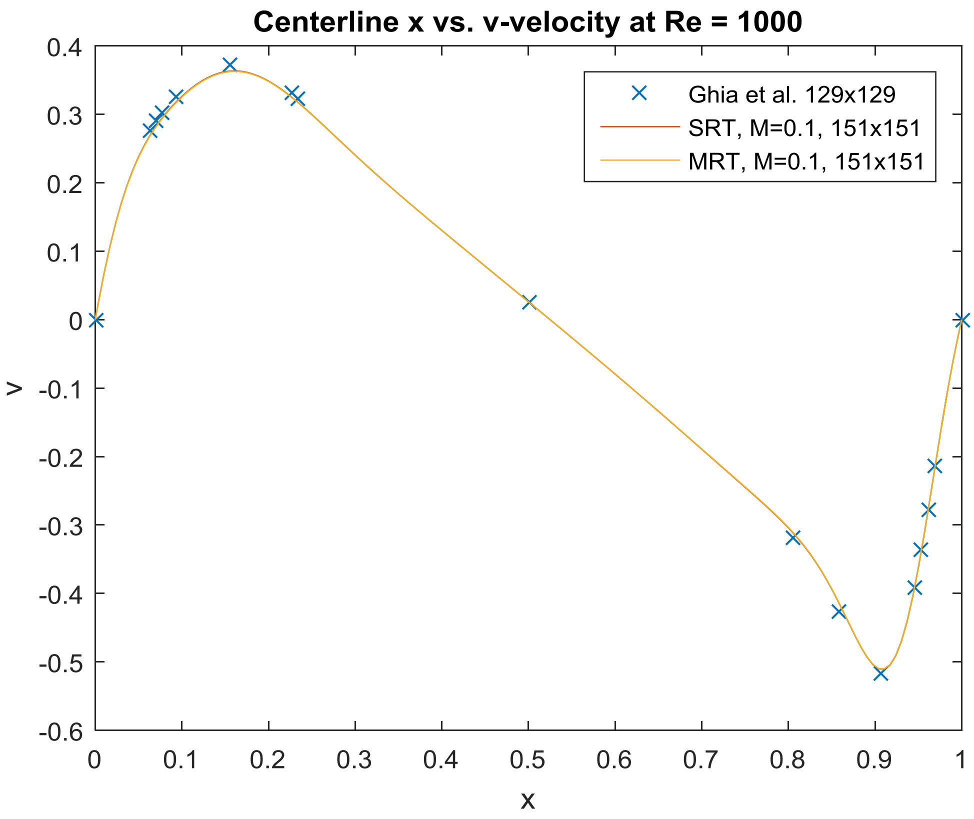
Now onto the streamline plots and centerline velocity profiles for Re values of 100, 1e3, 5e3, and 1e4. It was seen that SRT was able to simulate Re values of up to about 5e3, with proper grid refinement. MRT seemed to have no trouble handling the highest Re values tested in the Ghia reference. It seemed that the relaxation time values for the test cases of Re=1e4 and below were quite high compared to the minimum limit of MRT. Simulations at Re values of 1e5 and 1e6 will be shown, which tested the limits of MRT minimum stable relaxation time.

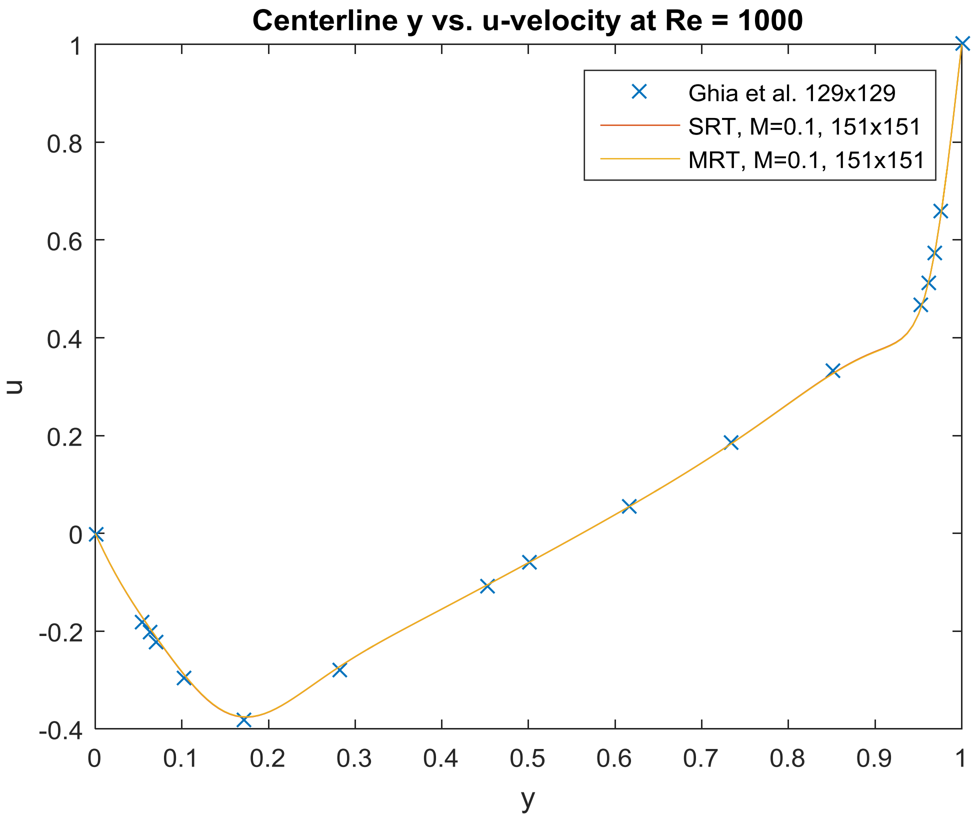

Steady-State Streamlines at Re = 100, M = 0.1
y
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
x
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

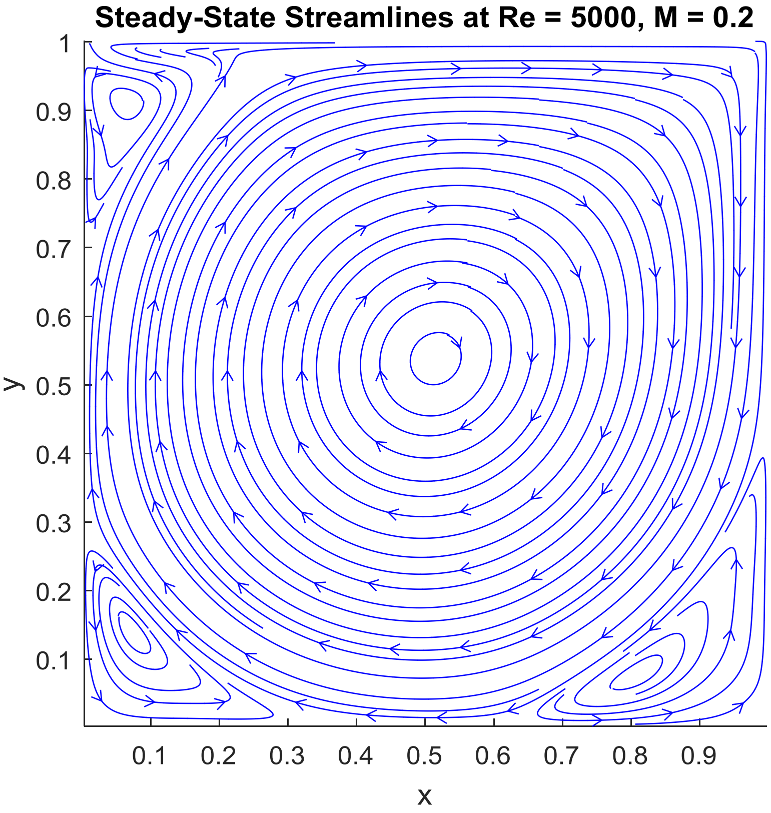

Centerline x vs. v-velocity at Re = 100
Ghia et al. 129x129
SRT, M=0.1, 125x125
MRT, M=0.1, 125x125
v
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
-0.3
x
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

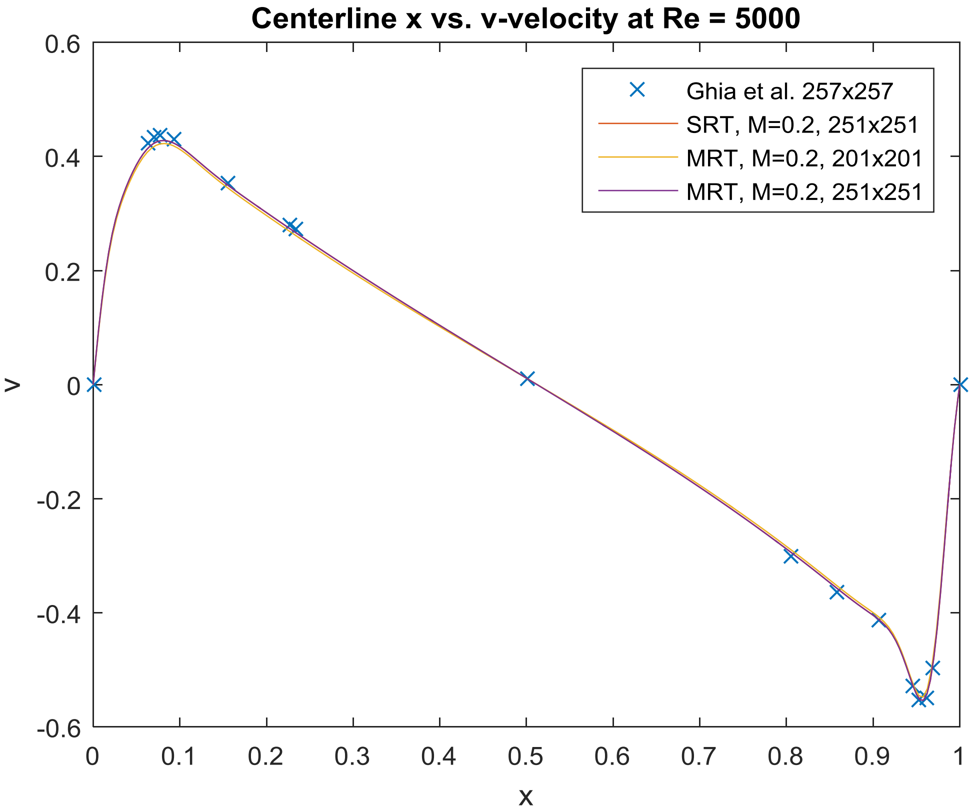

Centerline y vs. u-velocity at Re = 100
Ghia et al. 129x129
SRT, M=0.1, 125x125
MRT, M=0.1, 125x125
u
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
y

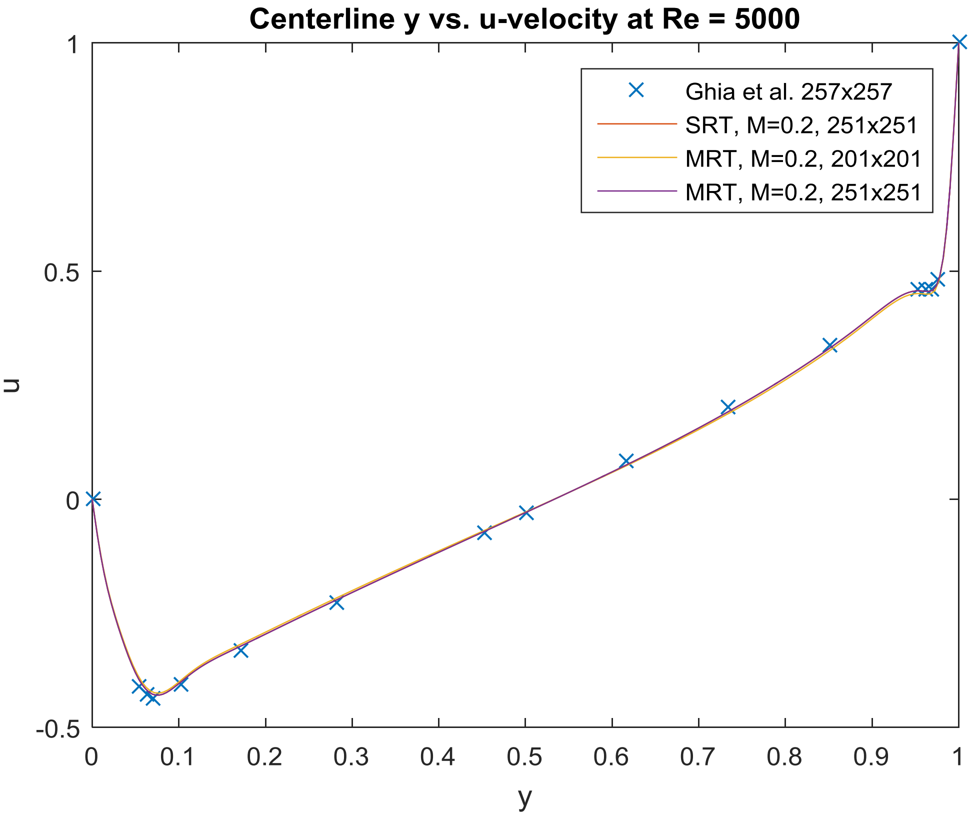

Steady-State Streamlines at Re = 1000, M = 0.1
y
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x

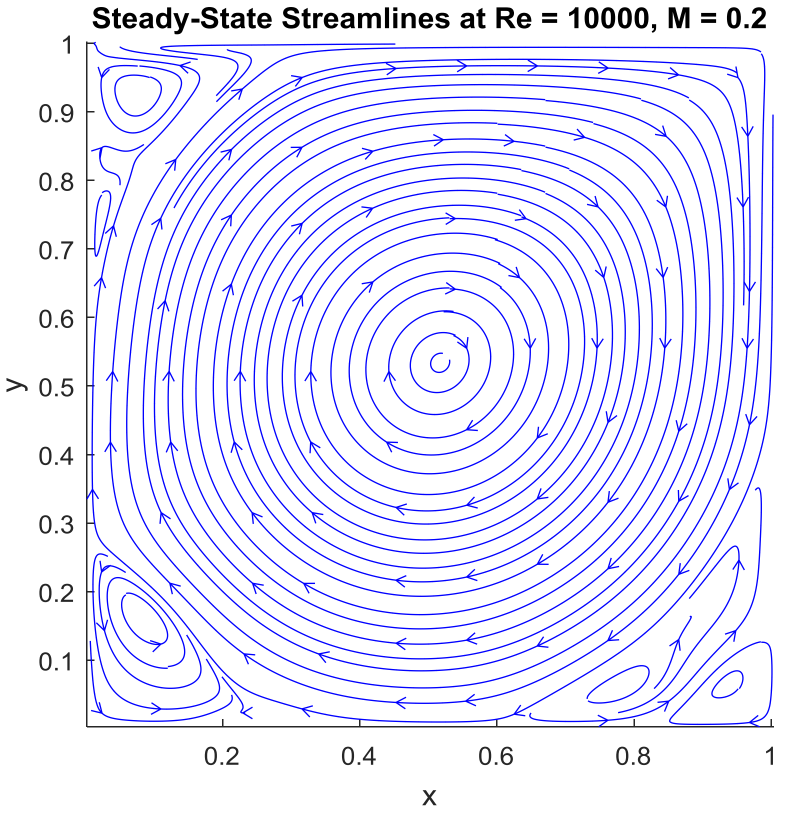

Centerline x vs. v-velocity at Re = 1000
Ghia et al. 129x129
SRT, M=0.1, 151x151
MRT, M=0.1, 151x151
v
x
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

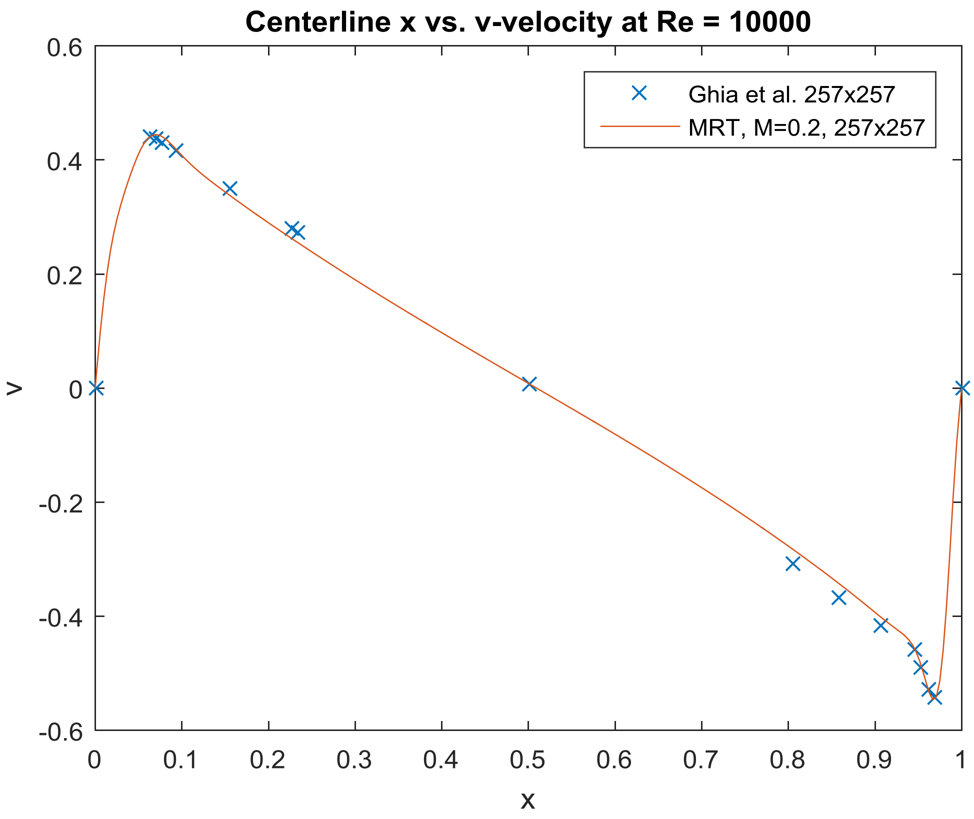

Centerline y vs. u-velocity at Re = 1000
Ghia et al. 129x129
SRT, M=0.1, 151x151
MRT, M=0.1, 151x151
u
y
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

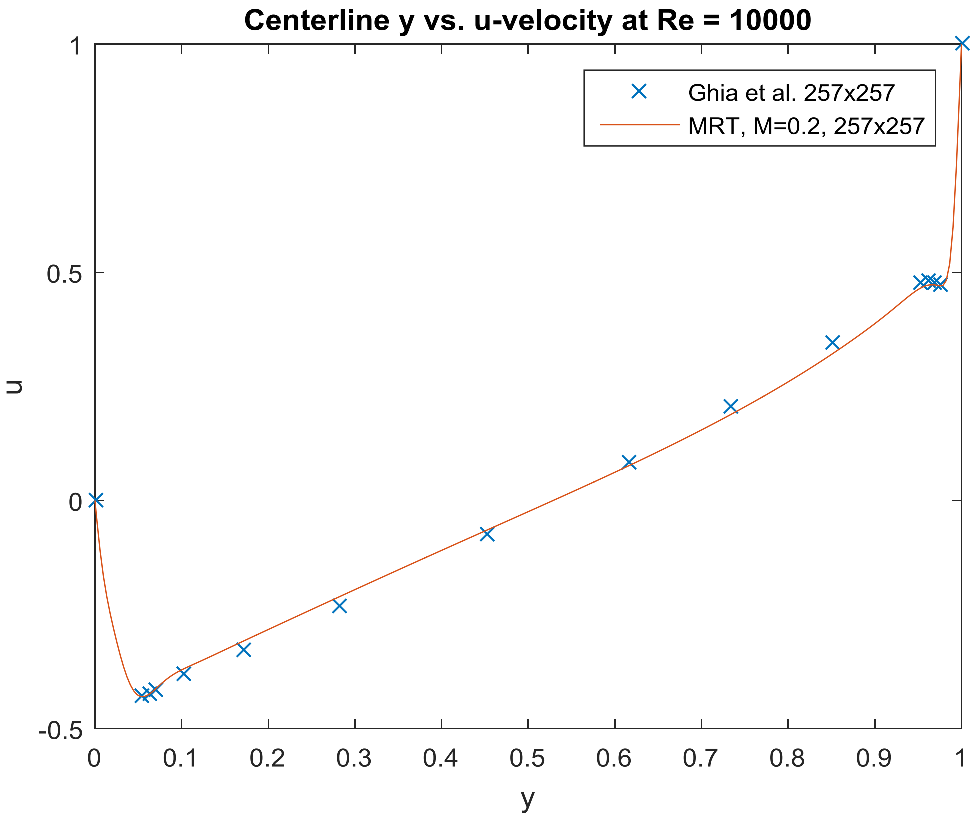

Steady-State Streamlines at Re = 5000, M = 0.2
y
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x


Centerline x vs. v-velocity at Re = 5000
Ghia et al. 257x257
SRT, M=0.2, 251x251
MRT, M=0.2, 201x201
MRT, M=0.2, 251x251
v
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x


Centerline y vs. u-velocity at Re = 5000
Ghia et al. 257x257
SRT, M=0.2, 251x251
MRT, M=0.2, 201x201
MRT, M=0.2, 251x251
u
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
y


Steady-State Streamlines at Re = 10000, M = 0.2
y
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.2
0.4
0.6
0.8
1
x

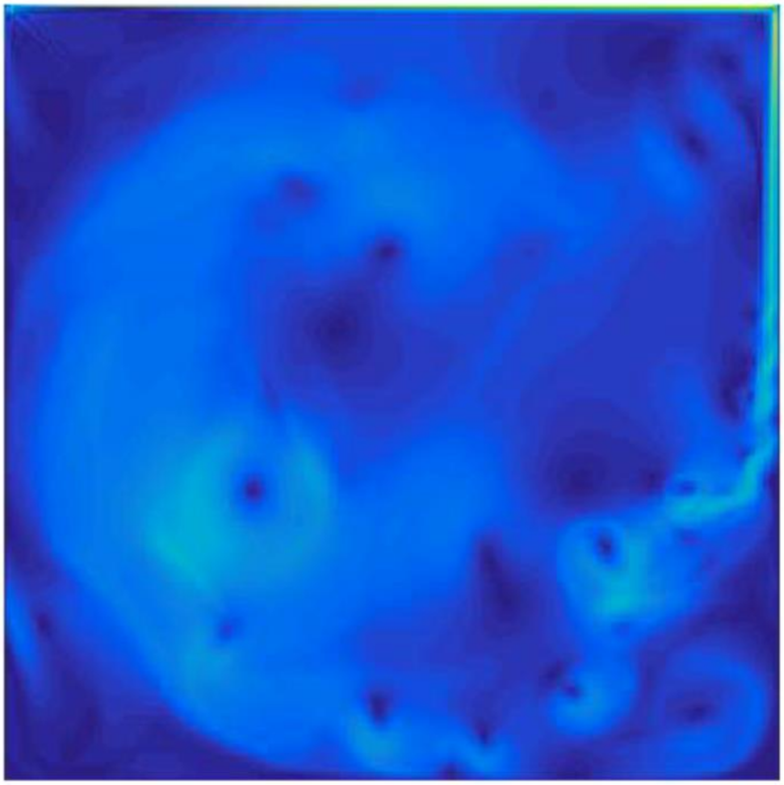

Centerline x vs. v-velocity at Re = 10000


Centerline y vs. u-velocity at Re = 10000

The stability of MRT was further tested by simulating at Re values of 1e5 and 1e6. MRT showed the capability of simulating flow at Re=1e6 at a grid resolution of 501x501. There are no validation data to compare for such HRE flows, as the flow is highly unsteady.

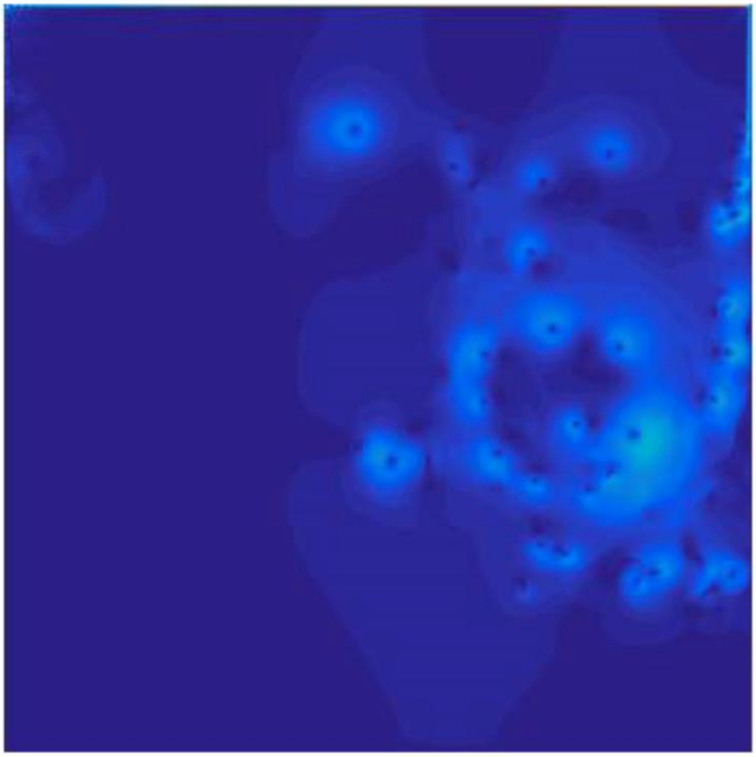
Re=100,000 (251x251)

$$\tau = 0.500869$$



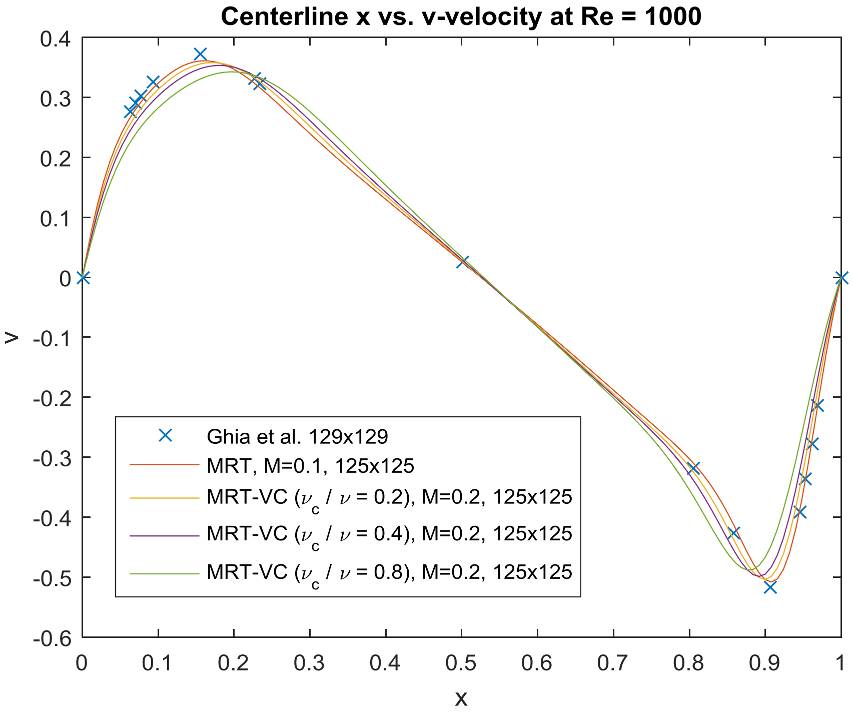
Re=1,000,000 (501x501)

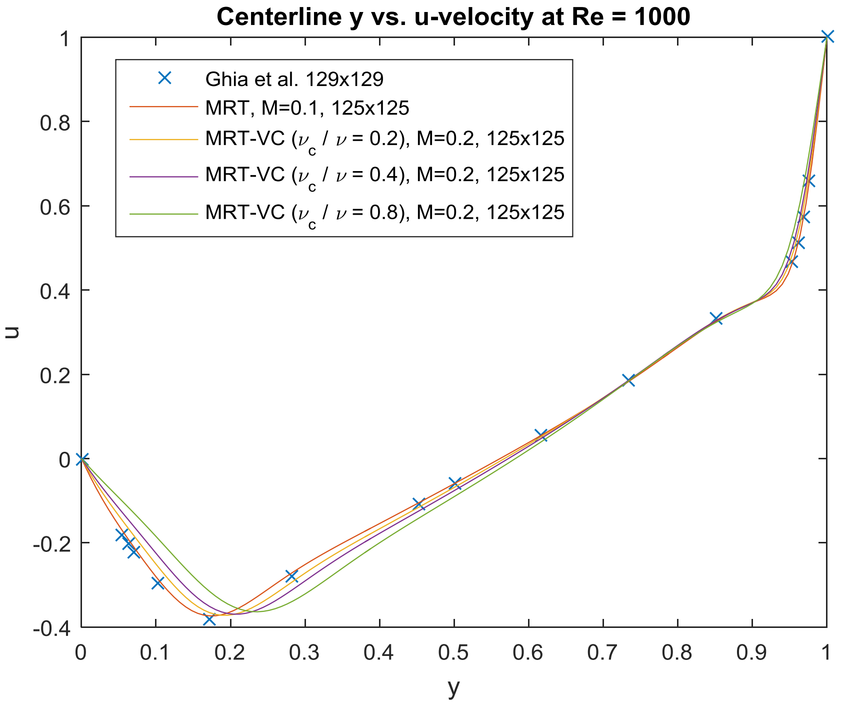
$\tau = 0.500260$



VC Results

VC was applied to a relatively low-Re case to test if the same, accurate results can be reproduced even when the advantage of VC is not of use.



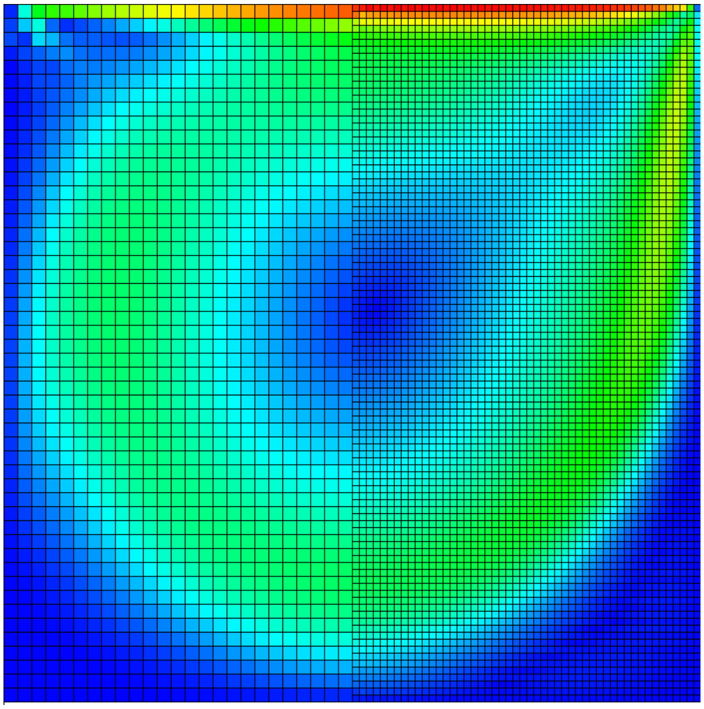


Even for small viscosity buffer ratio values, the results were affected in a negative way. Therefore VC was deemed unfit for further study. However, since VC uses very high-order stencils, it could be that VC works properly with sufficiently fine meshes. Grid dependency was not tested (125x125 is quite converged for the standard MRT at Re=1000) and a more thorough study can be conducted. Also, the possibility of implementation error cannot be ruled out completely.

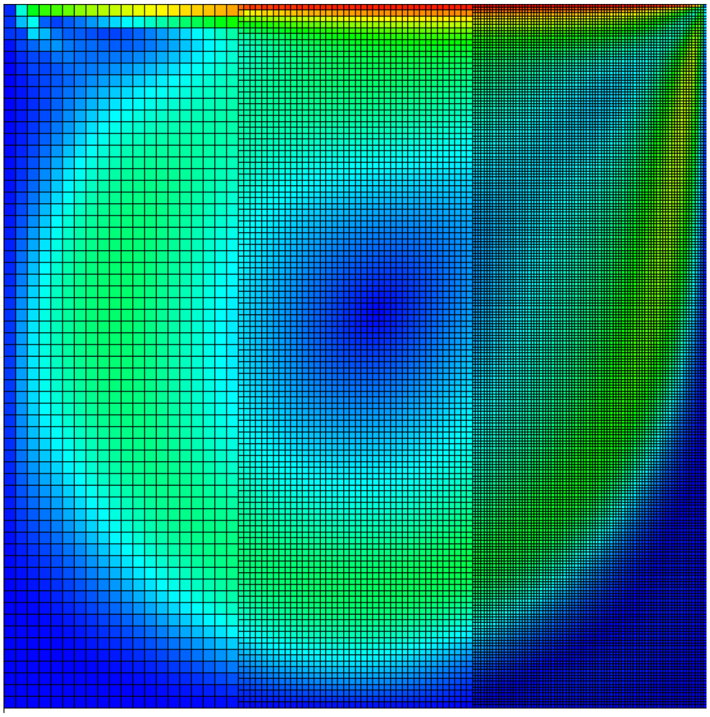
DGR Validation

DGR was validated by placing interfaces in validated standard MRT test cases to observe any negative effects on accuracy. Three test cases were used:

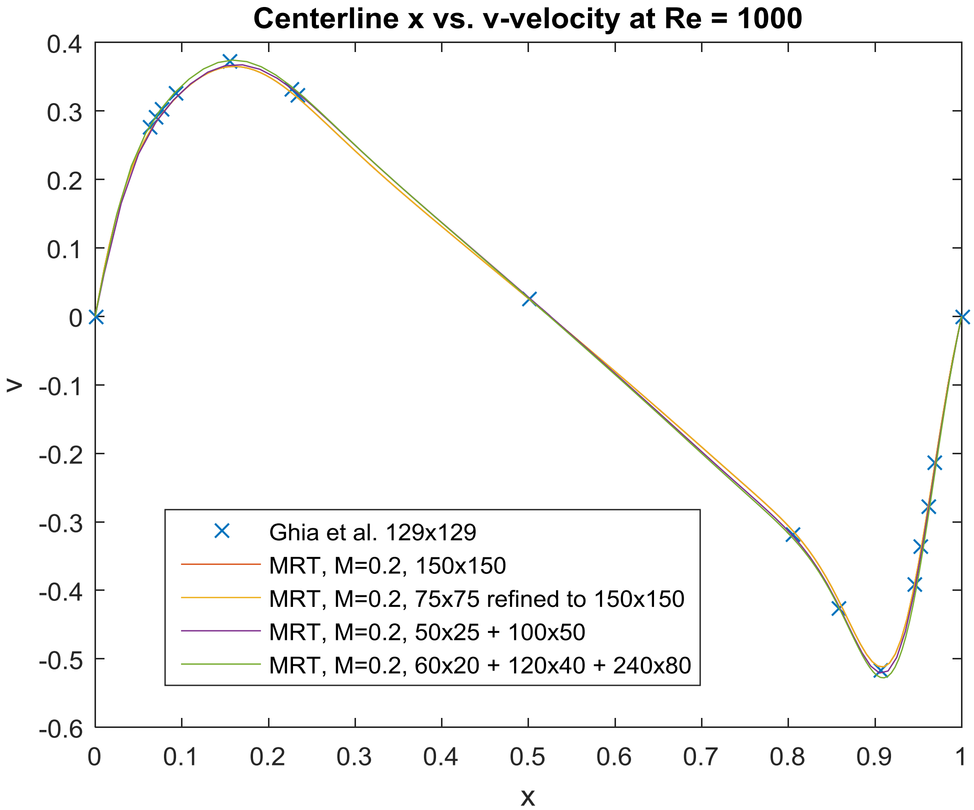
1. Full refinement to test the validity of viscosity scaling for finer cells.
2. Half-refinement to test the effect of the presence of an interface (shown below).

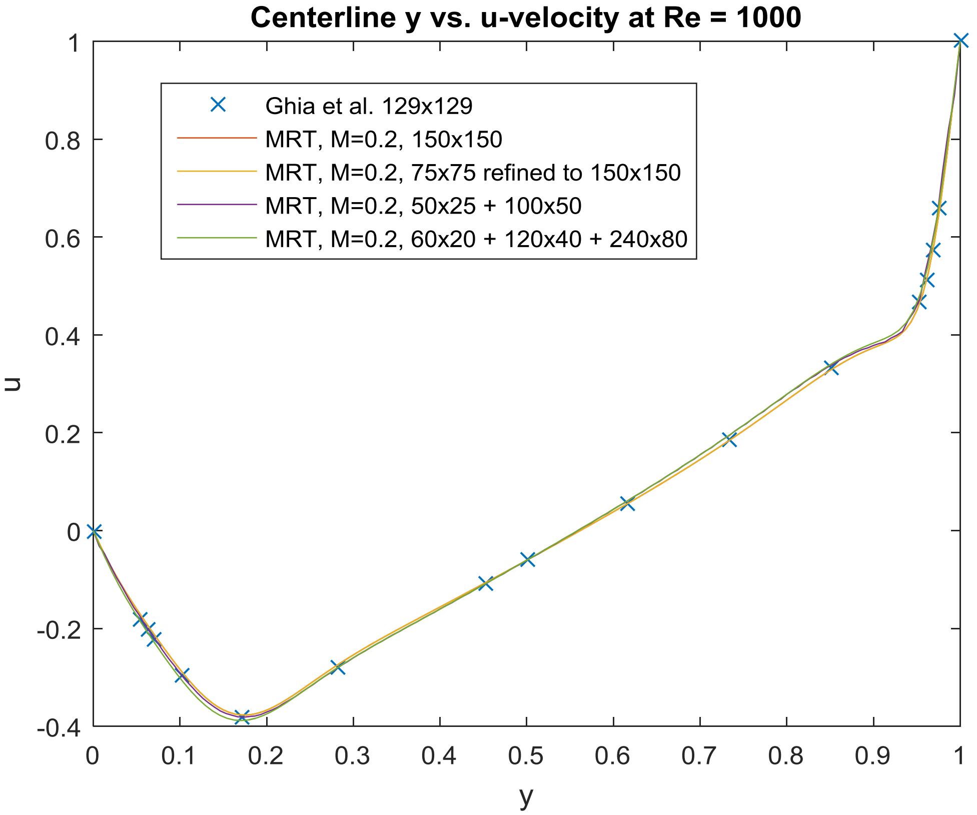


3. Three levels of refinement to test the effect of the presence of 2 interfaces and deeper iteration trees (3 levels deep in this case, shown below).



The results show that the half-refinement and full refinement cases almost negligibly affected the results. The 3-level case showed a bit more deviation, though still acceptable.



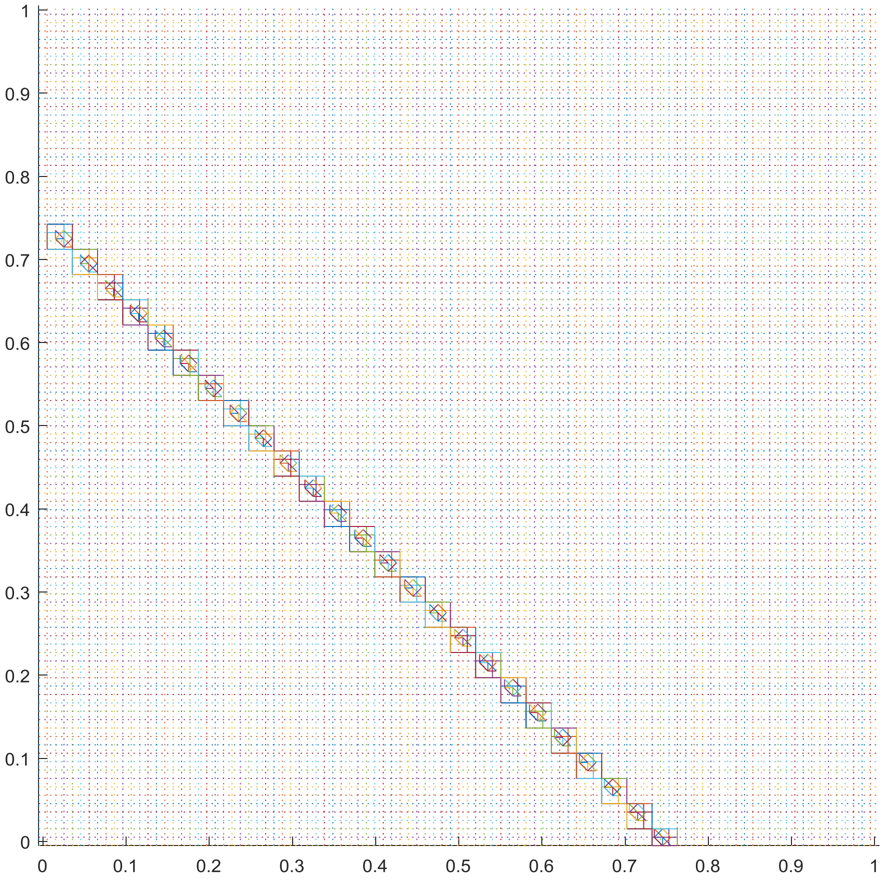


The 3-level refinement test case deviation could be due to:

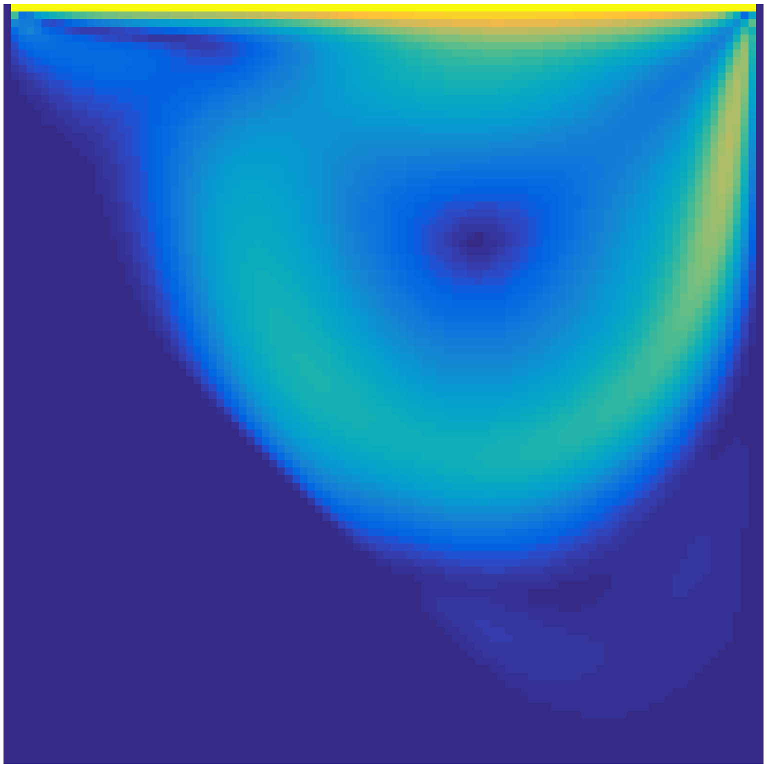
1. Introduction of an additional (1st-order-accurate) interface degrading accuracy.
2. Deleterious placement of coarse regions and/or interfaces.
3. Other effects due to the multi-level iteration scheme.
4. Implementation error.

GBB Results

GBB was implemented and tested in MATLAB, apart from the main C++ implementation, as a preliminary working model. The results look qualitatively promising. GBB was used to simulate a cut-corner LDC. The diagram below shows the surfels and pgrams (surrounded by bounding boxes) used to model the cut surface.



The following shows a simulation at Re=1000:



The effect of the cut wall can clearly be seen, and the flow appears to be shaped according to intuitive expectations. There does not appear to be flow penetration of the wall. Due to time constraints, additional results and validation was not generated.

Conclusions and Future Work

Four features were investigated for their potential in enhancing the stability, accuracy, and efficiency of LBM:

1. MRT
2. VC
3. DGR
4. GBB

Of these, MRT and DGR were validated by LDC reference data and shown to be promising. MRT greatly increased the maximum Re. MRT enabled a simulation at Re=1e6. DGR was shown to preserve good agreement when applied to validated test cases. VC was seen to be too diffusive, at least for the LDC at Re=1e3. Finally, a working preliminary implementation of GBB was shown, but not validated.

Future work will build upon the work presented here. Specifically, DGR will be developed towards a run-time solution-adaptive meshing system. Ideally, a course mesh and minimum cell size parameter would be specified by a user and the solver would refine or coarsen cells based on run-time gradients and/or other criteria. This hopefully would increase both accuracy and efficiency, while at least preserving stability. GBB will be further developed and validated. 2nd-order-accurate variants of DGR and GBB will be implemented and validated. The ultimate goal is to produce a validated 2D solution-adaptive solver for HRE external flows. Work on this project will likely continue until this goal is met. Up-to-date project source code can be accessed at: https://github.com/rlee32/lbmpp.

References

MRT and VC:

Chunze Zhang, Yongguang Cheng, Shan Huang and Jiayang Wu (2016). Improving the Stability of the Multiple-Relaxation-Time Lattice Boltzmann Method by a Viscosity Counteracting Approach. Advances in Applied Mathematics and Mechanics, 8, pp 37-51. doi:10.4208/aamm.2014.m512.

DGR:

Hudong Chen, et al. (2005). Grid Refinement in Lattice Boltzmann Methods Based on Volumetric Formulation.

GBB:

Li, Yanbing (2011). An improved volumetric LBM boundary approach and its extension for sliding mesh simulation. Graduate Theses and Dissertations. Paper 12380.