Parallelization Of An Experimental Multiphase Flow Algorithm

Delft University of Technology

Ankit Mittal

August 12, 2016

Introduction

1. Multiphase flows are one of the most widely occurring flows in nature.
2. Have two or more immiscible fluids separated by an interface.
3. Of particular interest to the petroleum industry, such flows often occur in wells and pipelines during oil and gas production.
4. Working code to simulate multiphase flows in pipes available.

Aim

1. Understand the code.
2. Speed it up.

Part 1

Understanding the code...

Study the code

1. Physics of the problem and governing equations were studied
2. Discretization and time integration techniques.
3. Code implementation was thoroughly understood
4. Implemented solvers were studied.

Physics of the problem - Flow [1]

1. Consider a multiphase flow between the two fluids separated by a sharp interface.
2. Assume incompressible flow: the fluids on either side have different but constant densities and viscosity,
3. Also assume the flow to be isothermal and Newtonian.
4. The flow is governed by the 3-d unsteady incompressible Navier-Stokes equations [1].
5. Do not solve Energy equation.

Physics of the problem - Interface

1. Mark the two fluids as zero and one,
2. To separate the two fluid regimes, introduce a so-called color function χ defined as

$$\chi(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in \text{fluid } 0 \\ 1, & \mathbf{x} \in \text{fluid } 1 \end{cases}$$

3. subscript 0 and 1 indicate the respective fluids.

Physics of the problem - Interface

1. Volume Tracking methods - For numerical treatment of the interface.
2. Can be sub-categorized into the Level Set (LS) [2] method and the Volume of Fluid (VOF) [3] method.
3. LS not mass conserving. VOF is mass conserving but computationally very expensive.
4. We use MCLS method - combination of LS and VOF

Discretization & Linearization

1. Variables stored In Staggered formation [4] to avoid the so called Checkerboard modes

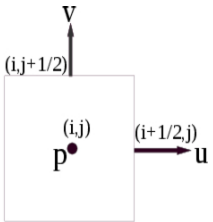


Figure : Arrangement of variables in Arakawa C grid.

2. 2nd Order space and time discretization (for the flow part) is used, for the interface 1st order.
3. Presently Newton's linearization [5] is used

Time Integration

1. The integration of flow and interface are staggered in time.
2. Interface is advected using MCLS method.
3. For flow integration two approaches can be used.
    1. Pressure velocity decoupled
    2. Fully coupled

Time Integration - Decoupled

1. Predictor

$$\frac{\hat{\mathbf{u}} - \mathbf{u}^n}{\Delta t} + \hat{F}\hat{\mathbf{u}} = -\frac{1}{\rho^{n-\frac{1}{2}}} B^T \mathbf{p}^{n-\frac{1}{2}} + \boldsymbol{\tau}^n + \left(\frac{1}{\rho}\mathbf{f}_s\right)^{n-\frac{1}{2}} + \mathbf{h}^{n-\frac{1}{2}}$$

2. Poisson

$$-B\frac{1}{\rho^{n+\frac{1}{2}}} B^T \mathbf{p}^{n+\frac{1}{2}} =$$

$$B\left(-\frac{1}{\Delta t}\hat{\mathbf{u}} - \left(\frac{1}{\rho}\mathbf{f}_s\right)^{n+\frac{1}{2}} - \frac{1}{\rho^{n-\frac{1}{2}}} B^T \mathbf{p}^{n-\frac{1}{2}} + \left(\frac{1}{\rho}\mathbf{f}_s\right)^{n-\frac{1}{2}} + \frac{1}{\Delta t}\mathbf{g}\right).$$

3. Corrector

$$\frac{\mathbf{u}^{n+1} - \hat{\mathbf{u}}}{\Delta t} = -\frac{1}{\rho^{n+\frac{1}{2}}} B^T \mathbf{p}^{n+\frac{1}{2}} + \left(\frac{1}{\rho}\mathbf{f}_s\right)^{n+\frac{1}{2}} + \frac{1}{\rho^{n-\frac{1}{2}}} B^T \mathbf{p}^{n-\frac{1}{2}} - \left(\frac{1}{\rho}\mathbf{f}_s\right)^{n-\frac{1}{2}}.$$

Time Integration - Decoupled

1. Robust and computationally cheap.
2. Used in the current code.
3. Does not preserve Kinetic energy of the fluid.
4. Preservation of Kinetic energy interesting for the turbulent flows.
5. If the Kinetic energy is not conserved, the size of eddies will not be accurately predicted.
6. To preserve Kinetic energy use Coupled solvers.

Description of the available code - Overall algorithm

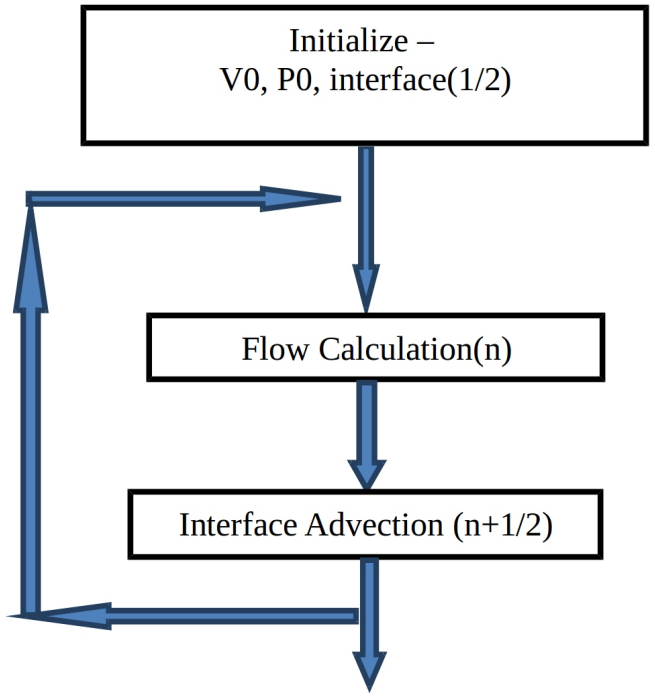


Figure : Overall flow chart of the algorithm.

Description of the available code - Flow Integration

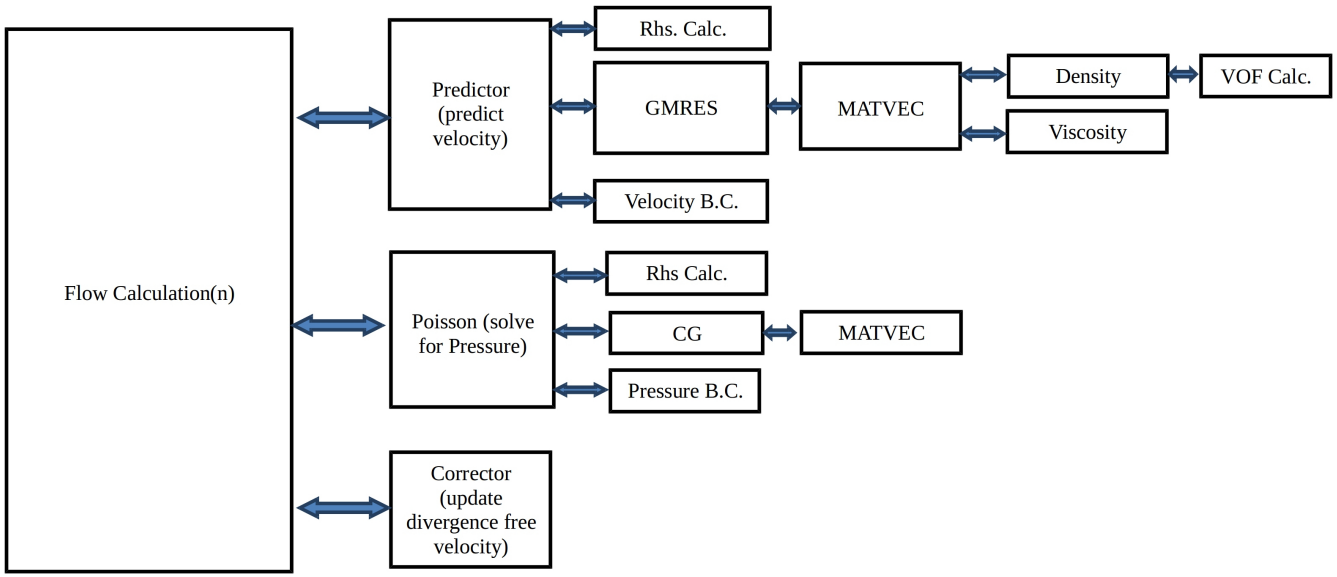


Figure : Brief flowchart of the flow algorithm.

Description of the available code - Interface Integration

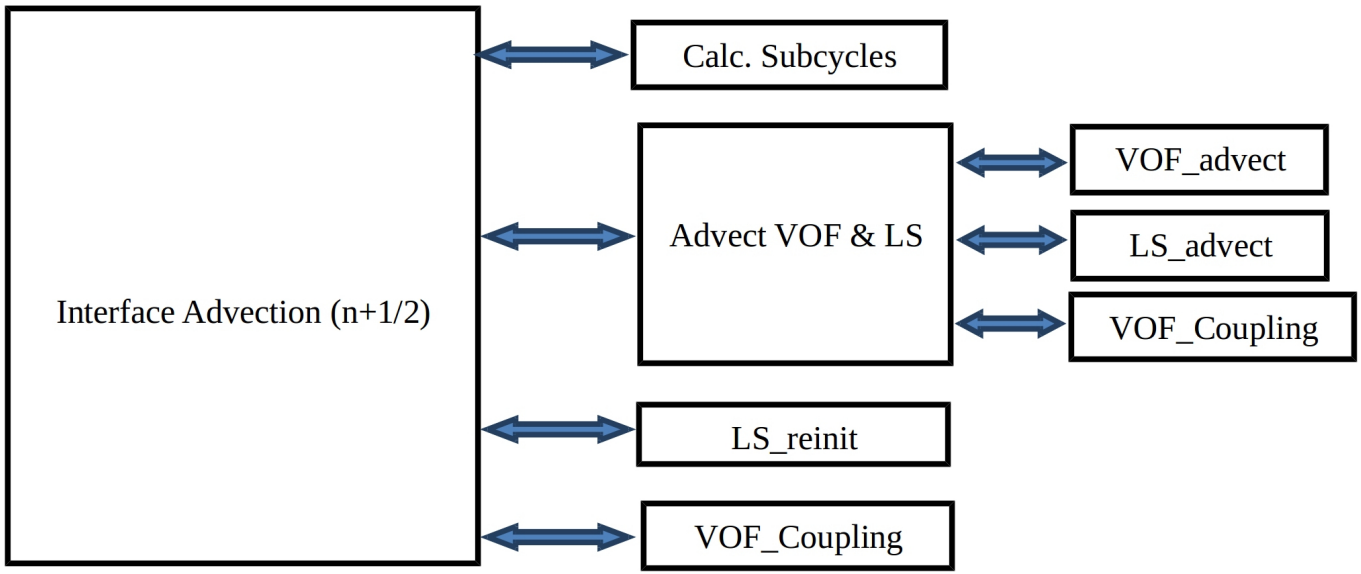


Figure : Brief flowchart of the interface advection algorithm.

Solvers Used

1. For solving the predictor, unpreconditioned restarted GMRES [6] is used.
2. Preconditioned CG [7] is used for symmetric pressure Poisson equation.
3. Incomplete Cholesky [8] is used as a preconditioner.

Part 2

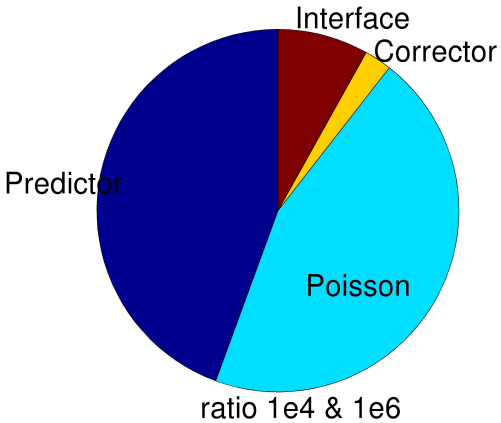
Improving the performance of the available code.

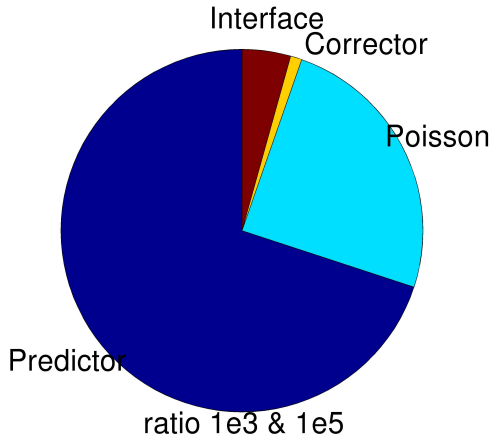
Improving the performance of the available code.

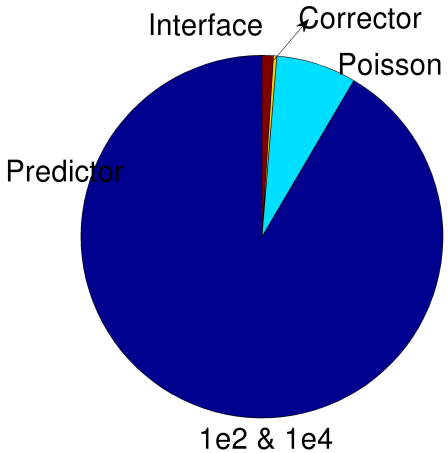
1. Based on the profiling results appropriate solvers and preconditioners were studied and implemented
2. Parallelization was implemented.
3. Deflation was implemented to improve the performance further.

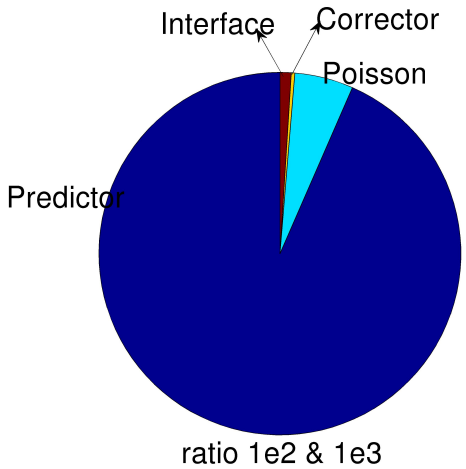
Profiling of the code

for different density to viscosity ratios









Profiling

1. The predictor part takes the longest time followed by the interface calculation part.
2. The Poisson solver took much less time.
3. We must focus our attention on the Predictor part.

Solvers Proposed

1. IDR(s) [1] in place of restarted GMRES to reduce the overhead of GMRES.
2. The convergence of IDR(s) against the restarted GMRES for the problem at hand will have to be tested.
3. The reduction of overhead in IDR(s) should not come at a cost of stalled convergence.
4. Hence a comparison study has is performed.

Preconditioners Proposed - Jacobi

1. Multiphase flows have a jump in diffusion coefficients across the interface.
2. This jump slows down the convergence of the iterative solvers.
3. Diagonal scaling for the predictor step.
4. Suitable for problem having discontinuous coefficients [2].
5. Lends itself well for parallelization.
6. Already implemented, gave 1.5-2 times speed up.
7. Moreover, the structure of the code was changed.
8. Instead of forming matrix in each GMRES iteration, we form it once and store it.

Parallelization

1. Parallelization by decomposing the domain axially.
2. For clusters (using MPI).
3. Deflation to improve the convergence on parallel system.

Preconditioners Proposed - Deflation

1. The convergence behavior of the preconditioner deteriorates as the domain is split into high number of sub-domains [3].
2. Hence, higher number of iterations need to be performed.
3. The loss of convergence is attributed to small eigenvalues arising from the domain decomposition.
4. This motivates the use for deflation [3, 4] for improving the convergence of preconditioned CG method in the Poisson equation.
5. Deflation can also be used as a preconditioner for coarse grid correction (predictor equation).

Steps taken for achieving a speedup

1. The structure of the code was changed.
2. Jacobi preconditioning applied.
3. Parallelization was implemented to reduce the computational time further.
4. IDR(s) and Deflation were also implemented.

Results and Discussion

1. Academic test cases
   1. To prove the accuracy of the code,
   2. Derive speedup characteristics of parallelization
   3. Speedup results for different solvers and preconditioners.
2. Practical test case
   1. To demonstrate the capabilities of the new code.

Academic Cases : Accuracy

1. To prove the accuracy of the code Rise velocities were compared for two cases.
2. Normalized mass of one fluid was also compared.

Accuracy : Rising Bubble

1. We initially have an axisymmetric bubble of one fluid in the other fluid.
2. As the time progresses, the bubble rises due to the buoyancy effect.
3. Initially the bubble behaves fine, but after some time the bubble is not axisymmetric anymore.
4. Due to the numerical inaccuracies and insufficient grid size.
5. These numerical inaccuracies give rise to instabilities, which causes different codes to behave differently.

Accuracy : Rising Bubble

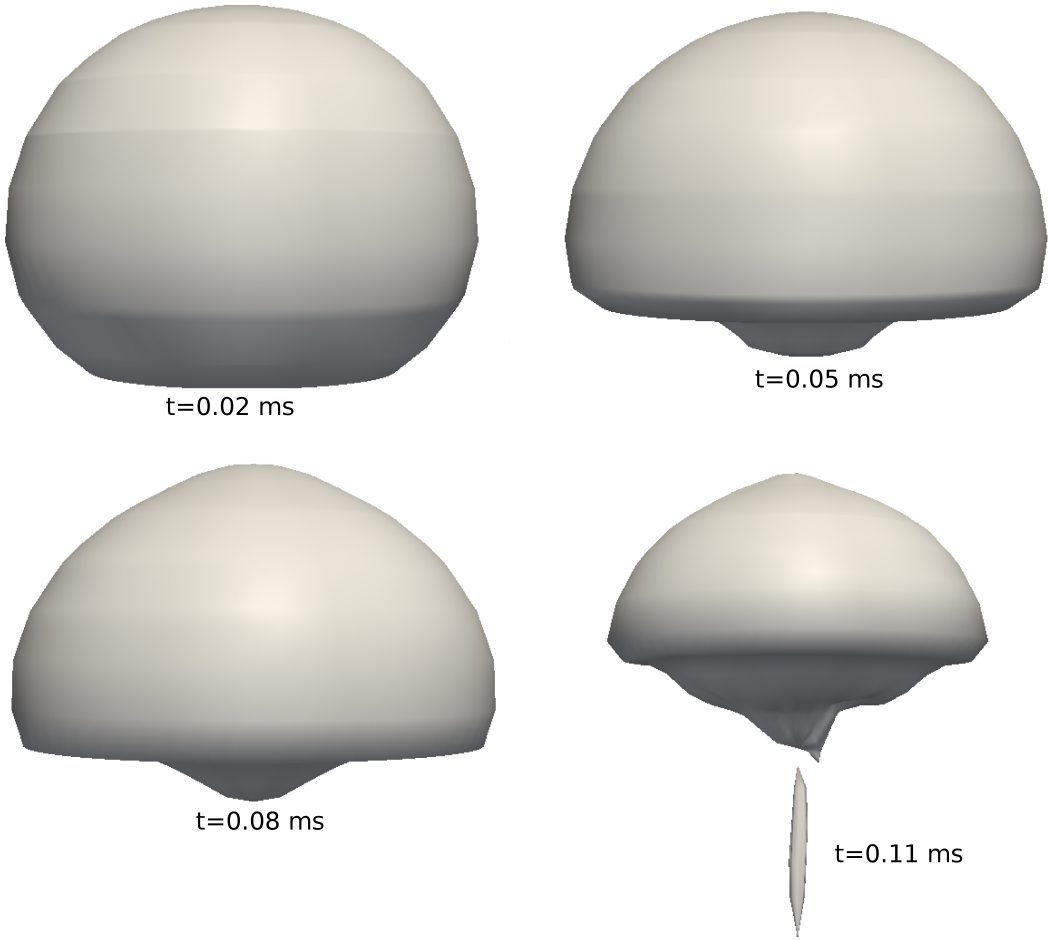
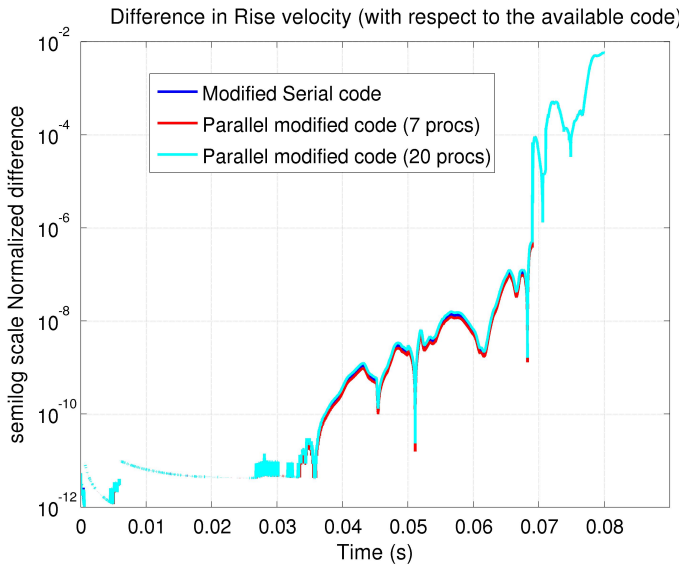
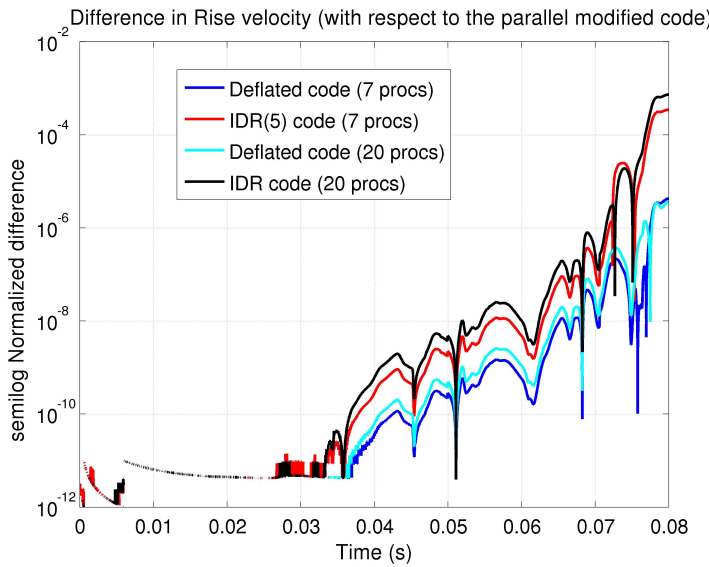


Figure : Movement of the bubble.

Accuracy : Rising Bubble



(a) Difference in Rise velocity obtained by modified serial and parallel codes.



(b) Difference in Rise velocity obtained by using deflation and IDR(s) solver.

Figure : Difference in Rise velocity for the rising bubble.

Accuracy

1. Normalized mass given by $\frac{mass(t)-mass(0)}{mass(0)}$ of one fluid given by all the codes is also compared.
2. Due to the mass conservation the normalized mass should ideally be 0.
3. Mass is below the specified tolerance of $1x10^{-8}$.

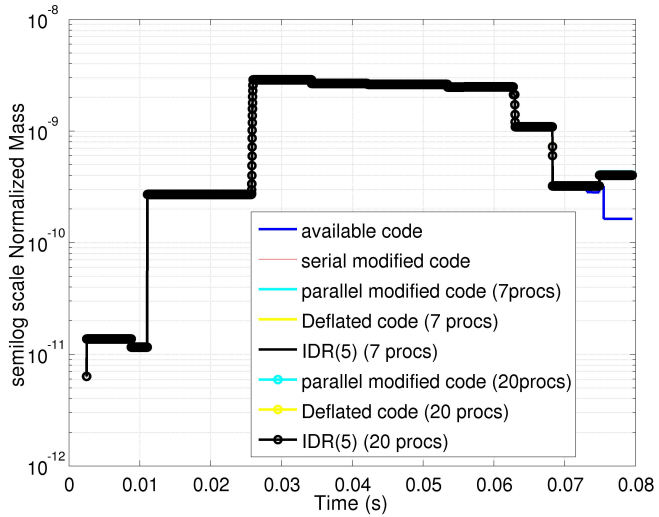


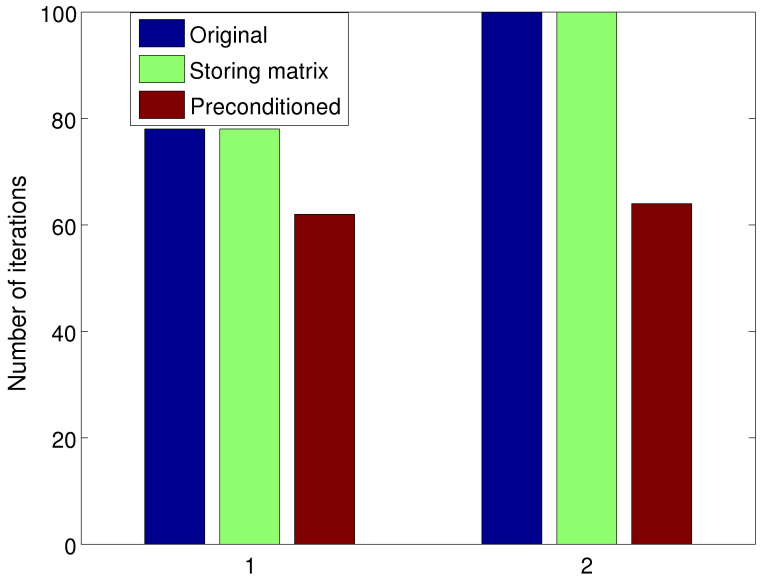
Figure : Normalized Mass obtained by different codes.

Speedup : Serial code modification and Jacobi diagonalization

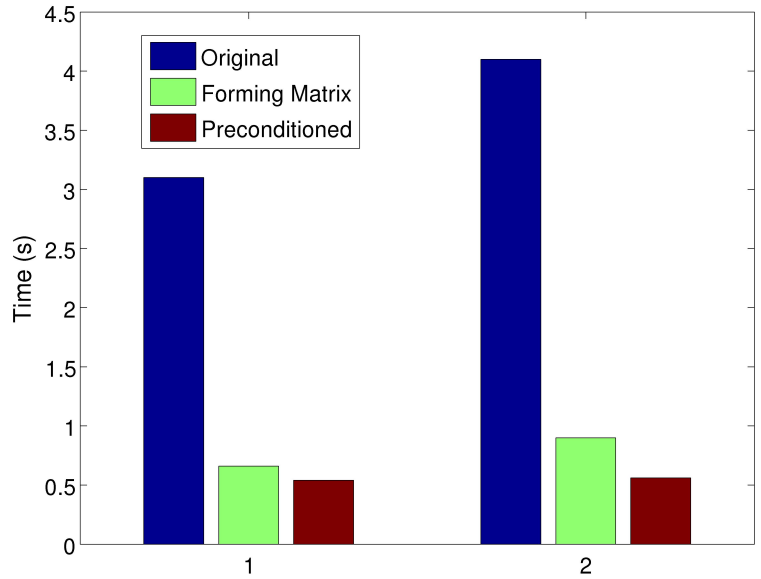
1. Restarted GMRES is used to solve the predictor part.
2. In the available code, the calculations of matrix entries are performed in each iteration.
3. We saved the matrix in diagonal form.
4. A speedup of 4 times was achieved.
5. The Jacobi preconditioner was used to speed up the predictor module.
6. It was chosen because it is easy to implement, delivers considerable speedups for our case [2]

Speedup : Serial code modification and Jacobi diagonalization

Density to viscosity ratio for the two fluids 1 - 1e2 & 1e4
2 - 1e2 & 1e3



(a) No. of iterations



(b) Time [s]

Table : Number of iterations and time taken by GMRES method to solve a single predictor step.

	Rising bubble			
Density to viscosity ratio	Original code		Storing Matrix	
	# Iter.	Time [s]	# Iter.	Time [s]
1e2 & 1e4	78	3.1	78	0.66
1e2 & 1e3	100	4.1	100	0.9

Table : Number of iterations and time taken by the improved GMRES method to solve a single predictor step with and without preconditioning.

	Rising bubble			
Density to viscosity ratio	No Precond.		Precond.	
	# Iter.	Time [s]	# Iter.	Time [s]
1e2 & 1e4	78	0.66	62	0.54
1e2 & 1e3	100	0.9	64	0.56

Speedup : Serial code modification and Jacobi diagonalization

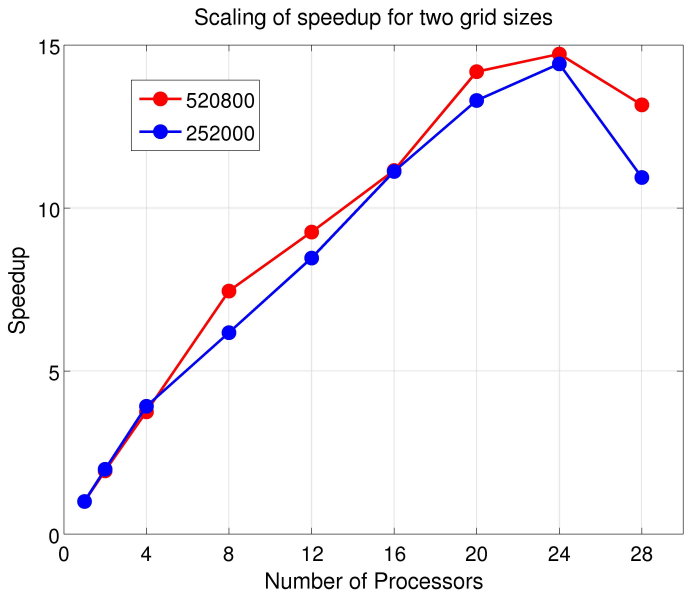
1. Restructuring gave a speedup of 4 times irrespective of the Reynolds number.
2. The Jacobi preconditioner gave a speedup of 1.5-2 time for low Reynolds number flow.
3. Target applications have low Reynolds number.
4. In total a speed up of 7-9 times was obtained for low Reynolds number flows.

Speedup : Parallelization (Rising Bubble)

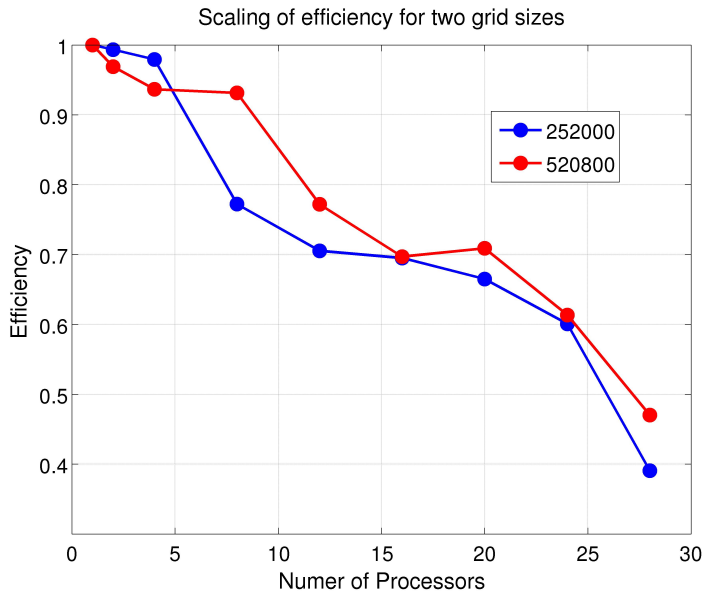
Table : Total computational time taken to solve Rising bubble for 2 grids on different number of processors.

Grid size	Time [s]									
	avail. code	mod. serial	parallel modified (# cores)							
			2	4	8	12	16	20	24	28
50x60x84	13500	1186.6	597.3	302.9	192.1	140.2	106.7	89.3	82.4	108.5
60x70x124	27940	2606.1	1344.7	695.6	349.7	281.4	233.7	183.8	177.1	198.2

Speedup : Parallelization (Rising Bubble)



(a) Scaling of Speedup



(b) Scaling of Efficiency

Figure : Scaling for the rising bubble problem.

Speedup : Parallelization (Rising Bubble)

1. A maximum speedup of around 15 times is obtained on 24 processors
2. Scaling does not improve much as the problem size increases.
3. As the grid is refined, the problem behavior changes due to which the solvers behave differently.
4. non-monotonicity in the parallelization efficiency comes
    1. Multiple levels of memory (cache)
    2. Load on the cluster is different at different times.

Speedup : Deflation (Predictor step)

1. No speedup gained.
2. Deflation not required.

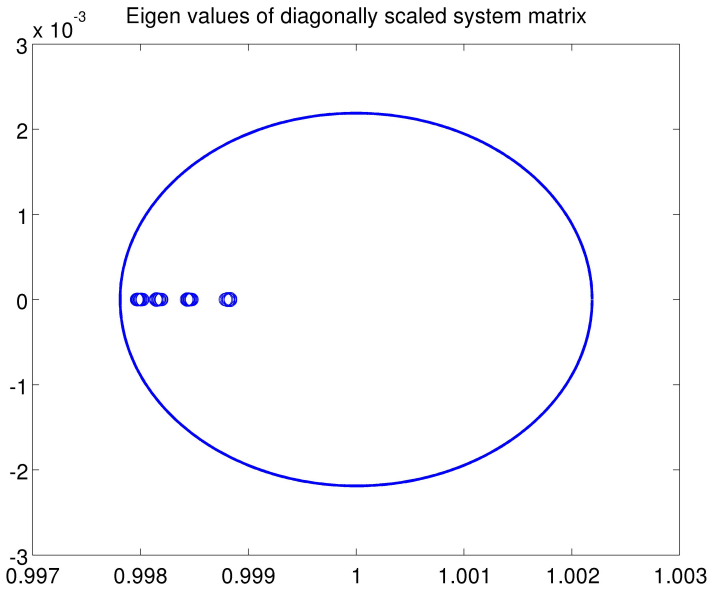


Figure : Smallest 50 eigenvalues of a typical diagonally scaled system matrix.

Speedup : Deflation (Poisson step)

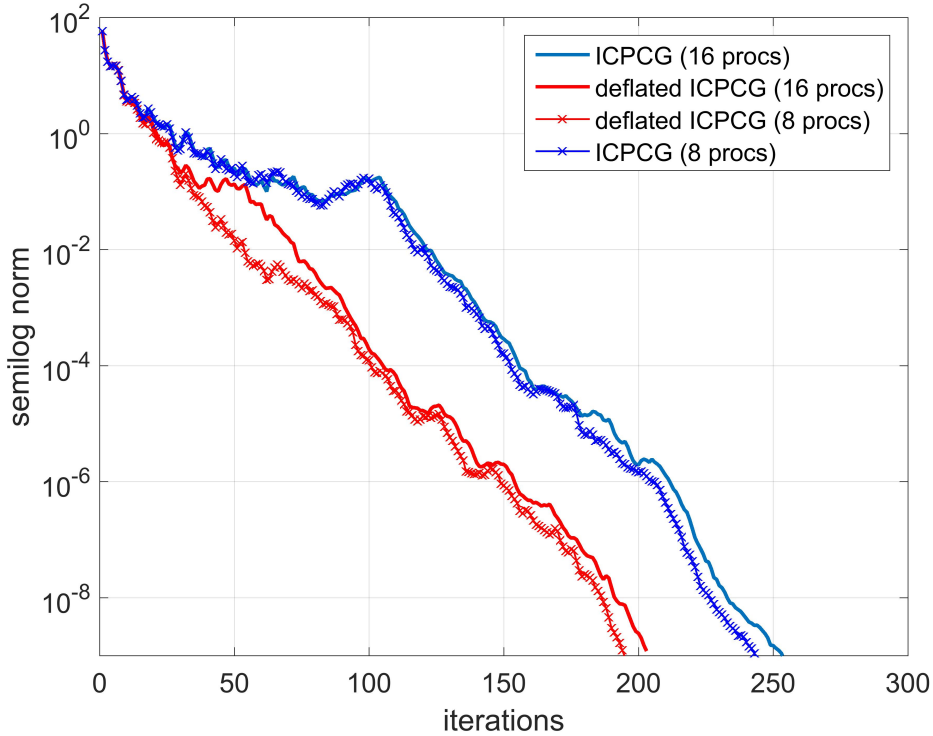


Figure : Convergence history of ICCG and deflated ICCG.

Speedup : Deflation (Poisson step)

1. Useful in improving the convergence properties of the incomplete Cholesky preconditioner.

Table : Time and iterations taken by the Poisson solver for a single integration step (Rising bubble).

Rising bubble					
		# cores			
		8	12	16	20
ICCG	# iterations	274	278	285	295
Deflated ICCG	# iterations	225	229	233	237
ICCG	Time(s)	0.73	0.478	0.37	0.34
Deflated ICCG	Time(s)	0.76	0.57	0.5	0.48

Speedup : Deflation (Poisson step)

1. But does not help in saving time.
2. ICCG does not perform particularly bad as number or processors increases.
3. Hence, deflation does not improve the convergence by much.
4. Deflation takes more time than it saves.

Speedup : IDR(s)

1. For Rising bubble problem we get speedup
2. GMRES takes more time to converge.
3. If for some case GMRES converges faster, IDR(s) may not be useful.

Speedup : IDR(s)

Table : Restarted GMRES versus IDR(s) for solving one predictor step on different number of processors (Rising bubble).

Timestep 0.01ms			
	No. of cores	Fastest Restarted GMRES	Fastest IDR(s)
# iterations (time(s))	4	17 (0.85)	18 (0.82)
	8	17 (0.45)	18 (0.41)
	16	17 (0.35)	18 (0.32)
Timestep 0.5ms			
	No. of cores	Fastest Restarted GMRES	Fastest IDR(s)
# iterations (time(s))	4	125 (7.6)	122 (4.2)
	8	125 (4.1)	117 (2.4)
	16	125 (2.7)	110 (1.6)

Speedup : IDR(s)

For Rising bubble (0.5ms) case

1. The fastest IDR solver is nearly 2 times faster than the fastest restarted GMRES solver (per predictor equation solve)
2. In terms of overall speedup a factor of nearly 1.5 is obtained

Practical Capabilities of The Modified Code

1. We simulate the flow field inside a pipe as indicated.
2. Initially, the pipe is assumed to be filled with stagnant oil
3. At t=0, water is provided at the inlet to flush the oil out.
4. After water reaches the bend, it instead of rising further at the same speed, creeps horizontally displacing the oil.

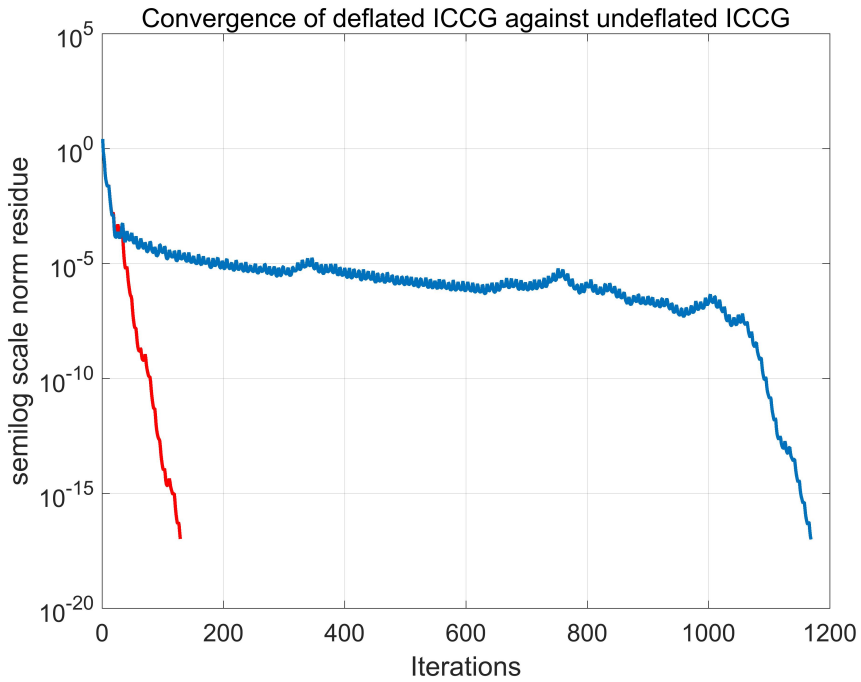
Figure : Geometry and configuration of the simulated pipe.

Practical Capabilities of The Modified Code

1. The resulting interface is difficult to capture numerically.
2. Previous simulations suggests that a no-slip boundary condition gives an unphysical interface shape [5]
    1. An oil film is formed between water and the wall
3. While a slip boundary condition over-predicts the speed of the interface [5].
4. In our simulation the bend is approximated by fixing an appropriate gravity vector.

Practical Capabilities of The Modified Code : Full scale Model

1. We saw that 40 processors gives the best performance for this case.
2. Deflation improved the convergence.



Practical Capabilities of The Modified Code : Full scale Model

1. Deflation gave a speedup of nearly 2 times.
2. For bigger problems, bigger gains from deflation could be expected.

Table : No. of iterations and time taken by (deflated) ICCG to solve the Poisson equation.

ICCG		deflated ICCG	
Number of iterations	Time [s]	Number of iterations	Time [s]
1167	3.3	129	1.7

Practical Capabilities of The Modified Code : Full scale Model

1. A speedup of 75 times was achieved by running the deflated code on 40 processors.
2. Available code takes 2818.51s for 10 iterations, modified code takes 37.29s.

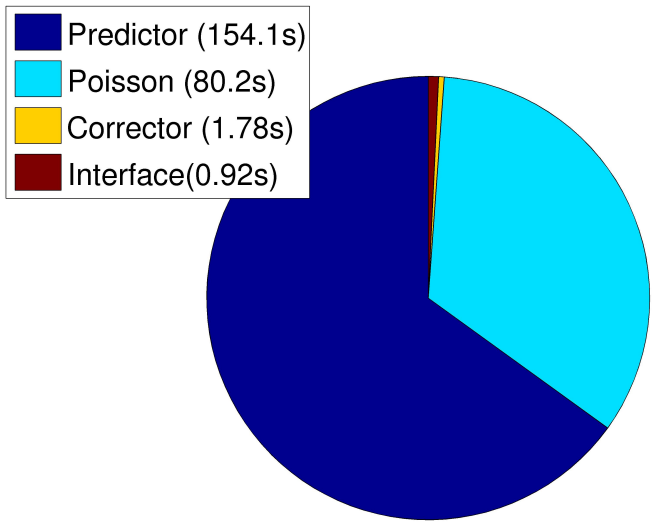
Table : Total time taken by available code and deflated ICCG code on 40 processors to integrate 10 time steps.

	available code	deflated ICCG on 40 procs
Time [s]	2818.51	37.29

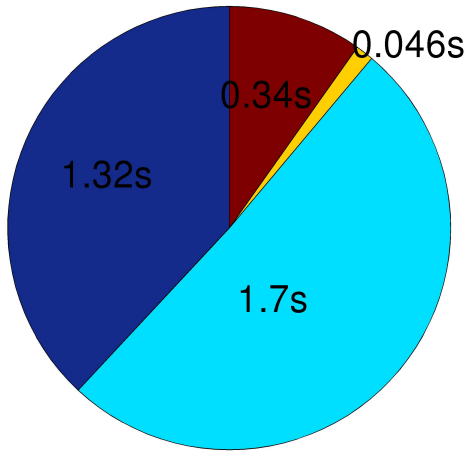
Practical Capabilities of The Modified Code : Full scale Model

Profiling the modified parallel code on 40 processors.

1. Poisson step is the new bottleneck.



(a) Profiling : available code



(b) deflated ICCG (40 procs.)

Full scale model : Physical results

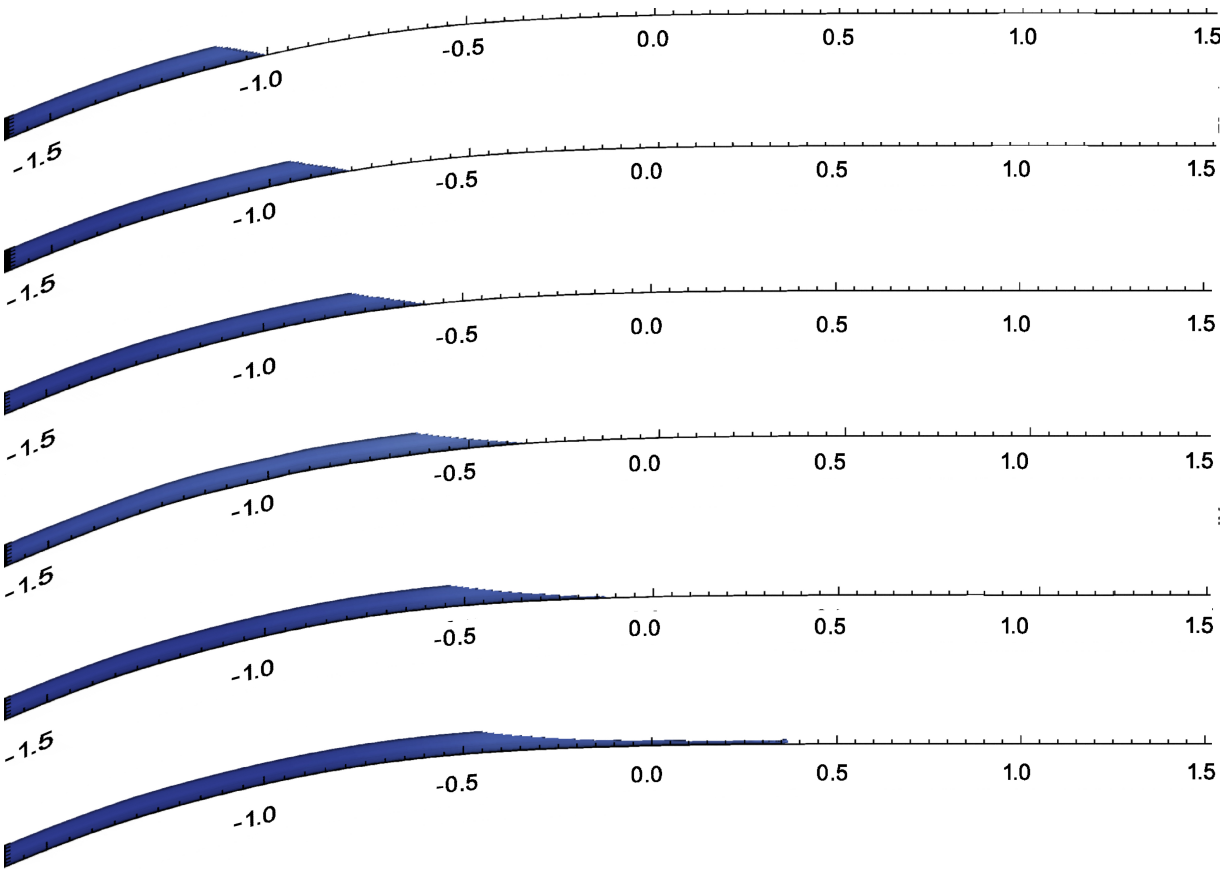


Figure : Movement of the interface with time.

Full scale model : Physical results

Comparison of the physical results with the reported results.

1. The reported results are for 2-d channel flow, while we have a 3-d pipe flow.
2. The reported results are obtained by implementing an adaptive grid refinement, while uniform grid is used in the current endeavor.
3. The initial conditions are different for our case.
4. The interface is captured using the VOF method in [5], while in the present study we use the MCLS method.
5. The length of the horizontal part for the current test case is 3.2 m, while in [5] it was 15 m.

Full scale model : Physical results

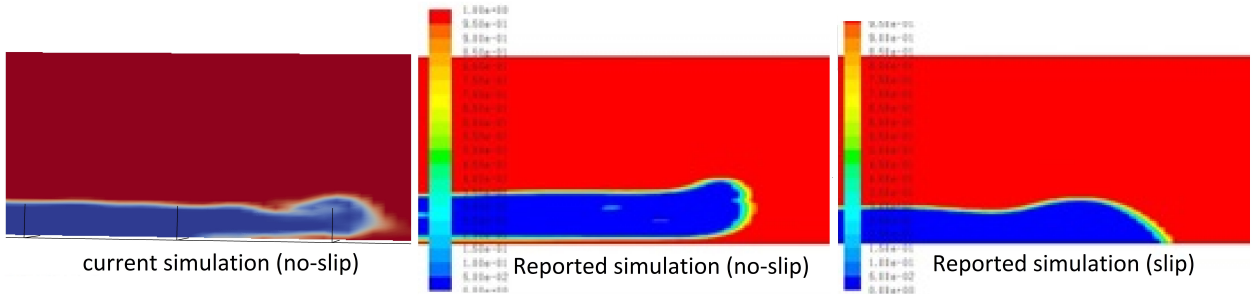


Figure : Shape of the interface head captured by the current simulation and as reported in [5].

Table : Interface velocity obtained from experiments and simulations.

	experiments	current (no-slip)	reported (no-slip)	reported (slip)
velocity $[m/s]$	0.143	0.15	0.11	0.165

Conclusion

1. Aim.
2. Profiling results revealed that predictor step is the heaviest.
3. The diagonal scaling of the predictor matrix gave a speedup of nearly 1.5-2 times.
4. Storing the matrix in the predictor step, further gave a speedup of 4 times.
5. In total 4.5-9 times speedup was obtained in the predictor step, due to the above modifications!
6. Parallelization was reduced the computational time drastically.
7. A speedup of nearly 15 times was achieved on 24 processors for the rising bubble problem.

Conclusion

1. Deflation did not help for the predictor equation owing to the presence of $\frac{1}{timestep}$ term on the diagonal of the system matrix.
2. Deflation improved the convergence for the predictor step, but time saving depends on the case.
3. IDR(s) was quite helpful in reducing the computational time if the matrix had slightly poor spectral properties.
4. For TNO test cases,
    1. Deflation reduced the computational time of the Poisson equation substantially.
    2. The IDR(s) method was of no advantage for these cases since the GMRES method converged quickly.
    3. 6 days on 40 cores - full scale.
    4. An overall speedup of nearly 75 times was obtained.

Conclusion

1. The speedup gained by using different solvers, preconditioners and parallelization makes it possible to simulate the real world problems in a reasonable amount of time.
2. This code can definitely help a researcher carry out the numerical simulations for pipe flows in much less time.

Future Recommendations

1. In this endeavor we achieved considerable speedups, and improved the performance of the original code many folds.
2. Reduce the computational time of Poisson equation (the new bottleneck).
    1. Different solvers/preconditioners.
    2. The structure of the Poisson solver could be modified to reduce the computational time.
3. Second future research direction - coupled solver to have a more accurate kinetic energy conserving solution.

References


S. van der Pijl. *Computation of bubbly flows with a Mass-Conserving Level-Set Method.* PhD Thesis, TU-Delft (2005).


S. Osher, J.A. Sethian. *Fronts propagating with curvature-dependent speed: algorithms based on HamiltonJacobi formulations.* J. Comput. Phys. 79, (1988) pp. 1249.


D. Gueyffier, J. Li, A. Nadim, S. Scardovelli, S. Zaleski. *Volume of Fluid interface tracking with smoothed surface stress methods for three-dimensional flows.* J. Comput. Phys. 152, (1999) pp. 423456


Y. Morinishi, O.V. Vasilyev, Takeshi Ogi. *Fully Conservative finite difference scheme in cylindrical coordinates for incompressible flow simulations.* Journal of Computational Physics 197, (2004) pp. 686-710.


T.W.H Sheu, R.K. Lin. *Newton linearization of the incompressible NavierStokes equations.* Int. J. Numer. Meth. Fluids 44, (2004) pp. 297-312.


Y. Saad, M.H. Schultz. *GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems.* SIAM J. Sci. Stat. Comput., 7, (1986) pp. 856-869.


Magnus R. Hestenes, Eduard Stiefel. *Methods of Conjugate Gradients for Solving Linear Systems.* Journal of Research of the National Bureau of Standards. 49, (1952) pp. 409436.


J.A. Meijerink, H.A. van der Vorst. *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix.* Math. Comp., 31, (1977) pp. 148162.

References


P. Sonneveld, M.B. van Gijzen. *IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Stsrems of Linear Equations.* SIAM J. Sci. Comput., 31 (2), (2008) pp. 1035-1062.


A. J. Wathen. *Preconditioning.* Acta Numerica, 24, (2015) pp. 329-376.


T.B. Jonsthovel, M.B. van Gijzen, C. Vuik, A. Scarpas. *On The Use Of Rigid Body Modes In The Deflated Preconditioned Conjugate Gradient Method.* SIAM J. Sci. Comput., 35 (1), (2012) pp. B207-B225.


J. Frank, C. Vuik, A. Segal. *On The Construction of Deflation-Based Preconditioners.* SIAM J. Sci. Comput., 23 (2), (2001) pp. 442-462.


B. de Jong. *Contact Line Dynamics in Oil Water Simulations.* Internship Report, TNO (2015).