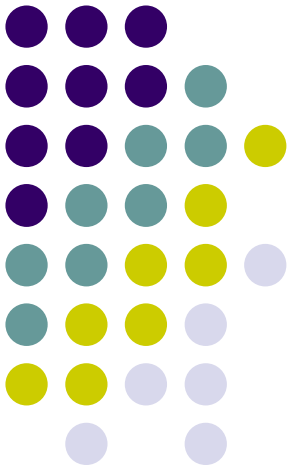
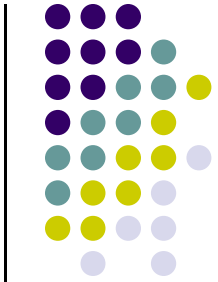
Flow Science Inc. Interview

MPI Developer Position

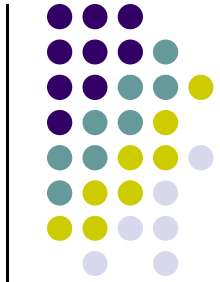
Gandharv Kashinath





Content

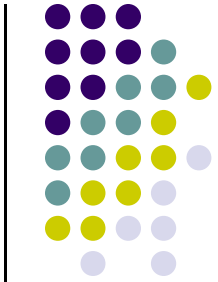
- Project 1: KIVA – Parallel Patch
- Project 2: Serial Lid-Driven Cavity
- Project 3: Parallel Shock Boundary Layer Interaction



KIVA-Parallel Patch

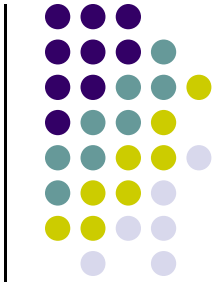
IC Engine Code

Background



- KIVA – Internal Combustion Engine Physics Solver
  - Transient, 3-D, multiphase, multi-component serial code written in FORTRAN 77.
  - Analysis of chemically reacting flows with sprays.
  - Two main components: Flow Solver (Navier-Stokes) and CHEMKIN (Finite Rate Chemistry).
  - Used by Auto/Aero Research Group at University of Michigan.
- Motivation for Parallelization
  - Research Group seeking for any speed-up in short time frame.
  - Serial code limited by the chemistry modeling segment.
  - Chemistry subroutine is highly decoupled from the entire system.
  - Reaction Kinetics is modeled as a chemical source term in the governing equations (N-S).
  - Project work for Grad Level - Introductory Parallel Computing course at University of Michigan.

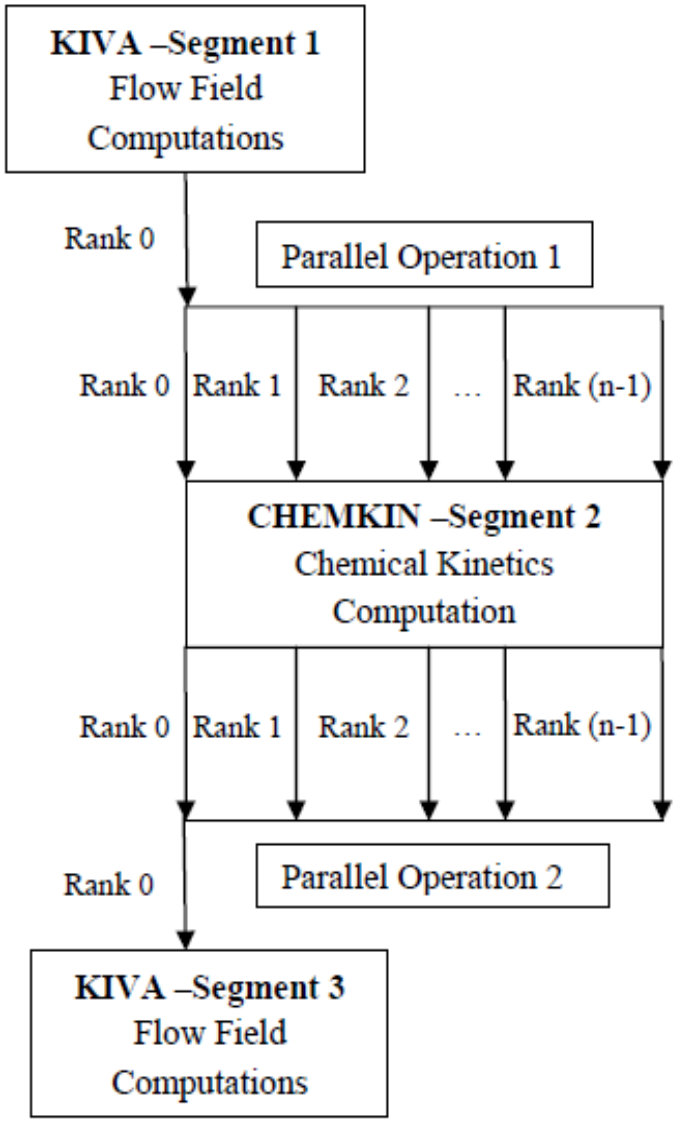
Parallel Algorithm Design and Development
Master-Slave Approach



- Parallelization focused only on the limiting segment - CHEMKIN call.
- A Master Slave Approach considered and implemented

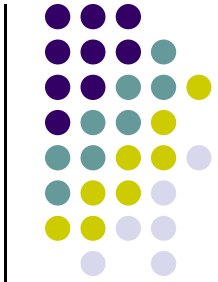
- Functions of Master processor
  - Run Initial segments of the Flow Solver
  - Initiate and perform Domain Decomposition
  - Assign work to slave processors
  - Participate in CHEMKIN solve
  - Gather Solution from CHEMKIN
  - Continue with Flow Solver
- Functions of Slave processors
  - Perform finite rate chemistry computations using CHEMKIN ode solve
  - Communicate switches to other all processors to activate and deactivate relevant models
  - Keep a check on global parameters and report any violations

Parallel Algorithm Design
Master-Slave Approach



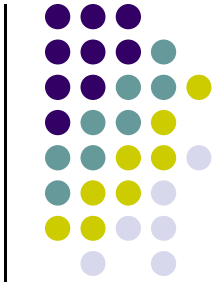
Parallel Algorithm Design and Development

KIVA – Influence and Constraints



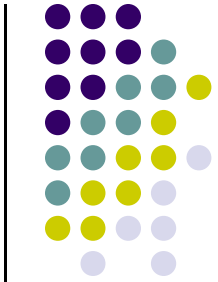
- All flow variables 1-D vectors on an unstructured grid – KIVA grid notation retained for convenience.
- Adaptive grids meant the domain decomposition needed to be dynamic.
- Grid features identifying fluid and solid domain actively used as CHEMKIN was only applied on fluid domain.
- Grid points activate and deactivate during computation but grid fixed before CHEMKIN call (changes only with each time advance).
- Several parameters like crank angle, rpm, hydrodynamic pressure needed to be continuously monitored across the entire fluid domain.
- Other switches updated coming out of CHEMKIN so that the flow solver could use them for further computation.

Parallel Algorithm Implementation
Domain Decomposition



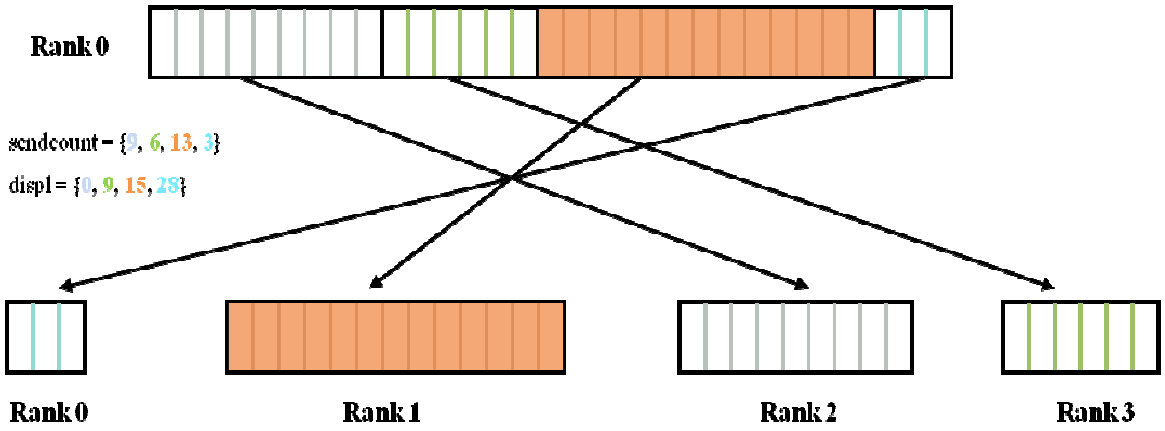
- CHEMKIN - An ODE Solver
  - Each node point computation is dependent on itself.
  - No boundary conditions on spatial domain required.
  - Boundary conditions are imposed in the time domain at each local time-step.
  - This is automatically imposed by Master processor before entering the CHEMKIN subroutine.
- Data Distribution
  - Number of cells in fluid domain computed at each time step.
  - Depending on combinations of odd and even; processors and cells, distribution kept as uniform as possible among all processors.
  - Flow variables like density, temperature, velocity, internal energy and species density are scattered as evenly as possible among all processors for CHEMKIN computation.

Parallel Algorithm Implementation
Domain Decomposition

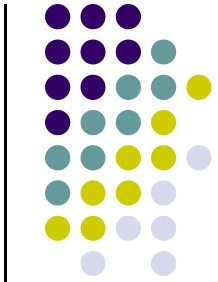


- Parallel Operation 1
  - Split primitives among all the processors including the master.
  - Collective Communication initiated by the master node using MPI_SCATTERV.
  - The send count and displacement vectors dynamically computed by the master node.
  - Local variable of each of the global primitive variable being decomposed - recvbuf.

```
MPI_SCATTERV(sendbuf, sendcount*, displ*,   sendtype,
             recvbuf, recvcount,  recvtype, root,
             comm,    ierror)
```



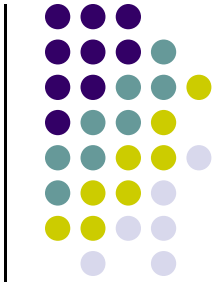
Parallel Algorithm Implementation



- CHEMKIN Computation
  - CHEMKIN – ODE Solver implemented on local primitives within each sub-domain simultaneously.
  - MPI_BCAST used to communicate dt, switch values for chemistry models.
  - MPI_REDUCE used to compute and communicate maximum and minimum values of T, P, ρ and other variables used as monitors in the fluid domain.
- Parallel Operation 2 (Inverse of Parallel Operation 1)
  - Integrate local primitives from all processors and reconstruct in Master.
  - Collective Communication initiated by the master node using MPI_GATHERV.
  - Global primitives reconstructed in the same locations as before scatter operation.

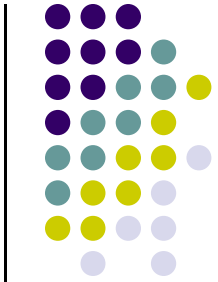
```
MPI_GATHERV(sendbuf,     sendcount, sendtype, recvbuf,
            recvcount*, displ*,    recvtype, root,
            comm,        ierror)
```

Parallel Algorithm Implementation
Miscellaneous Notes



- Processor Idling
  - Master executes serial while Slave processors remain idle.
  - MPI_BARRIER used to make sure the Slaves are in sync with the Master.
  - Essential since Slaves need to be at the same part of the program when the Master processor is ready to initiate the domain decomposition during each iteration.
- Legacy Code (FORTRAN 77)
  - Code developed in FORTRAN 77 - Dynamic arrays not used to compute the size of the local arrays.
  - Instead a common block was defined where the local variables were initialized using certain arbitrary sizes keeping in mind the size of the domain.
- Code Timing
  - MPI_WTIME subroutine was extensively employed at various portions of the code to analyze performance.

Parallel Performance Evaluation



- Overall performance evaluation not significant since more than 65% of the code is done in serial.
- Performance of only the CHEMKIN solve can be evaluated and its scalability can be studied.

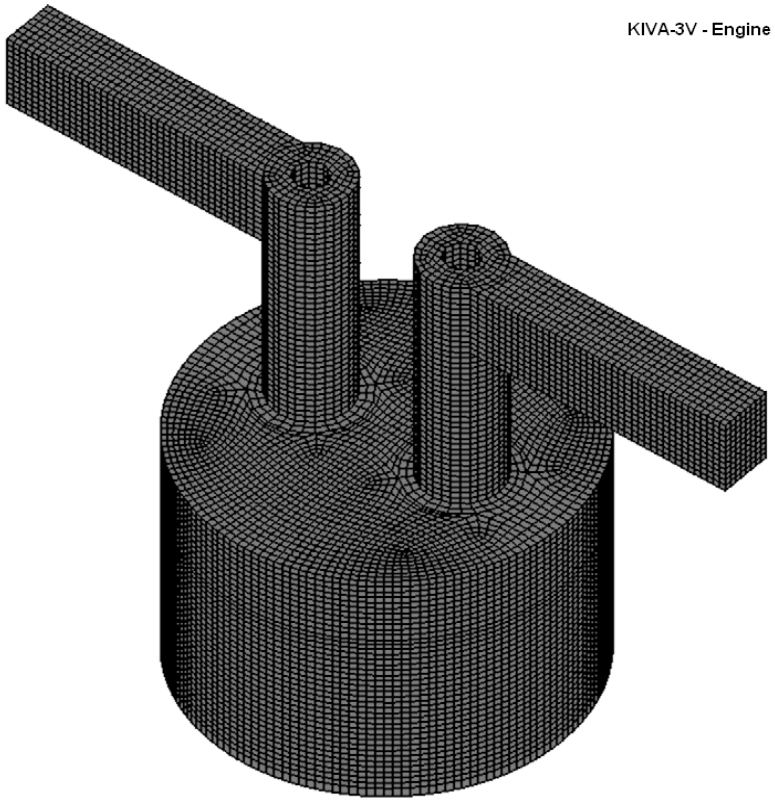
- Gustafson's Law

$$S(P) = P - \alpha(P - 1)$$

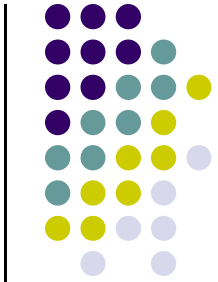
  - (S) Speed-up, (P) # of procs., (α) Non-parallelizable part of process.
  - Gustafson's law is similar to Amdahl's law, where the non-parallelizable section is considered the portions of the code that have been parallelized but may not be solved in parallel
  - This model can be extrapolated to the current problem since most segments of the code are executed in serial.
  - In order to compute the constant α the timer in the program was setup to calculate the times from each segment of the code.

Parallel Performance Evaluation
Simulation Studied

- 2-V Internal Combustion Engine.
- Operating conditions
  - Crank Angle varying from 180-360°.
  - Mesh of 128,000 grid points.
  - H_2, O_2, OH, N_2, H_2O, HO_2, H_2O_2, CO and CO_2 species with Shock and Detonation mechanism prescribed in CHEMKIN input files and similarly the thermodynamics data correspondingly chosen in CHEMKIN.



Parallel Performance Evaluation
Results



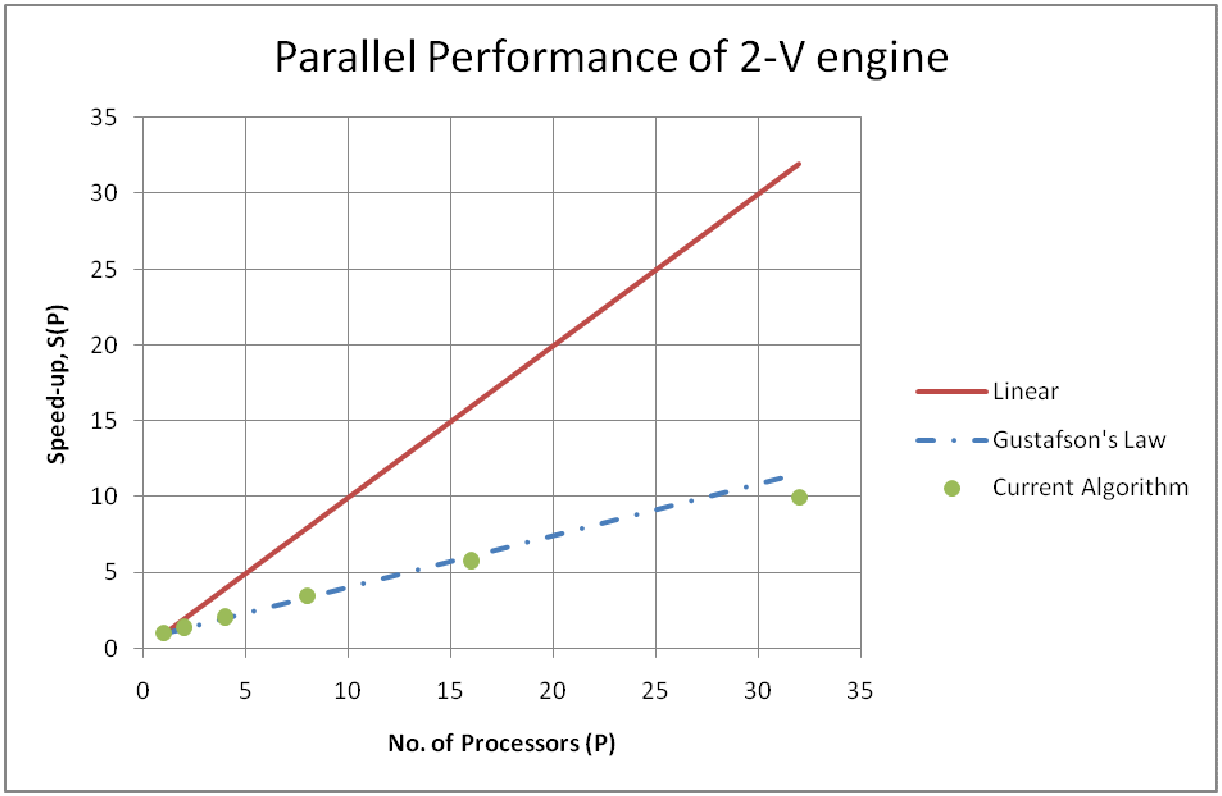
- Simulation setup and run on CAC-nyx cluster with the nodes that have dual-dual core Opteron CPU's at the University of Michigan, Ann Arbor.
- Reference Solution in serial was used to compute α=0.66.
- Needed to construct the Gustafson's Law Curve.
- Summary of time distribution

1-processor	Time (hr)	Percent (%)
KIVA - Segment 1	1.68	27.768595
CHEMKIN	2.01	33.2231405
KIVA - Segment 2	2.36	39.0082645
Total	6.05	

- Speed-up and Efficiency

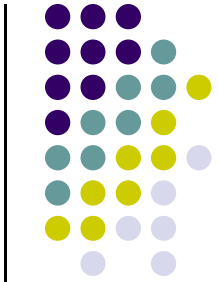
Number of Processors	Time (hrs)	Speed-up	Efficiency (%)
1	6.05	1	100
2	4.46	1.35650224	67.82511211
4	2.95	2.05084746	51.27118644
8	1.77	3.4180791	42.7259887
16	1.05	5.76190476	36.01190476
32	0.45	9.91111111	30.97222222

Parallel Performance Evaluation
Results

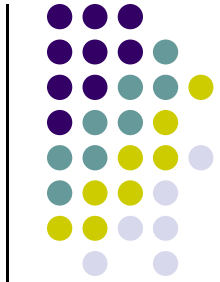


- Scalable Parallelism (Strong Scaling) obtained upto 16 processors.
- An improvement in performance was achieved in computing the reaction kinetics of the KIVA engine code.
- With increasing processors (P = 32) the speed-up diverges from Gustafson's Law.

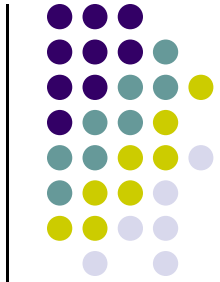
KIVA Parallel Patch
Summary



- Parallelization successfully incorporated and speed-up achieved.
- Parallelization not very well scaled for P≥32
  - Due to localized parallelization, α=0.66.
  - Could be resolved by considering an extensive parallelization.
- Code Needs to be tested for larger number of processors.
- Experience with parallelizing segments of a larger code.
- First hand experience with domain decomposition, synchronization and collective communication in parallel computing.
- Effective use of standard MPI libraries with Fortran 77.



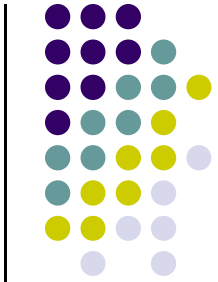
Questions?



Serial Lid-Driven Cavity Flow

Code-Development Experience

Background



- 2-D Lid-Driven Flow
  - Standard Validation Solution for several Incompressible CFD codes.
  - Excellent comparison data available for comparison.
  - Test bed for various numerical practices to be developed and tested for CFD codes.

- Motivation
  - CFD Programmer – Skill Set Development.
  - Exposure to various techniques in Numerical Methods.
  - Programming experience with Navier-Stokes Equation, applicable in almost all fluid flow phenomena.
  - Final Project of Advanced CFD Course at University of Michigan.

Lid-Driven Cavity Flow
Problem Formulation

- Flow described by the Incompressible (ρ = 1) Navier-Stokes equation.

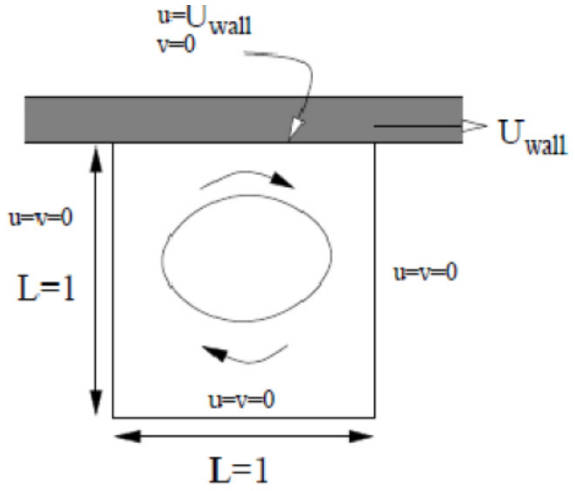
$$u_t + (F + p)_x + H_y^x = 0$$

$$v_t + H_x^y + (G + p)_y = 0$$

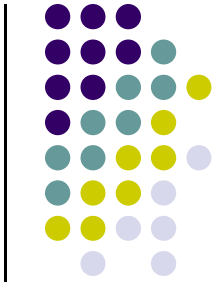
- The Fluxes are given by

$$F = u^2 - \nu u_x, \quad G = v^2 - \nu v_y, \quad H^x = uv - \nu u_y, \quad H^y = uv - \nu v_x$$

- The flow domain was given by

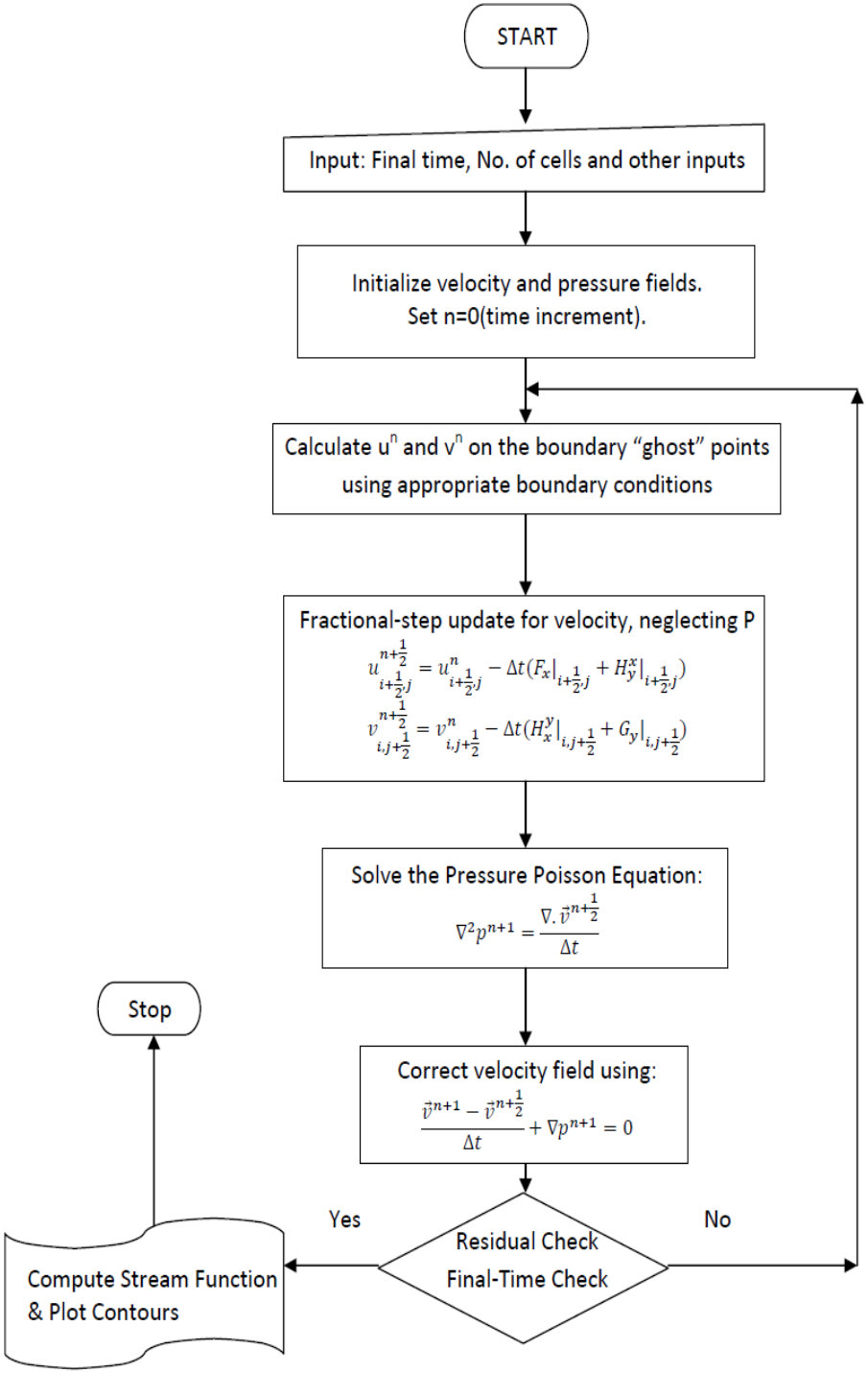


Lid-Driven Cavity Flow
Numerical Methods



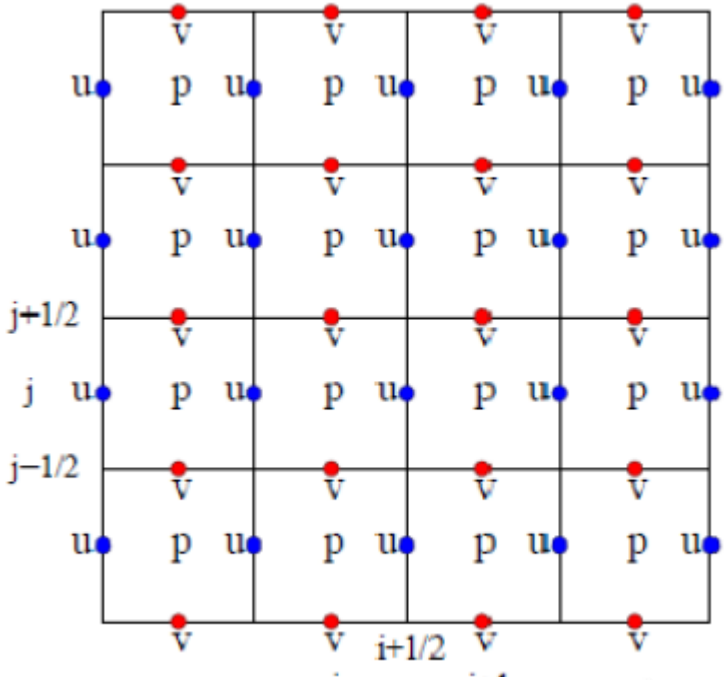
- Primitive Variable Approach – Pressure Correction Method
- All finite-difference schemes developed were second order accurate (higher order with limiters) in space and first order accurate in time.
- QUICK scheme was used to compute Fluxes.
- SMART limiter was also incorporated for high Re number cases.
- A staggered storage on a regular, orthogonal grid was used.
- Pressure Poisson Equation solved using an iterative Successive Over Relaxation based on the Gauss-Seidel method was used.

Lid-Driven Cavity Flow
Solution Algorithm

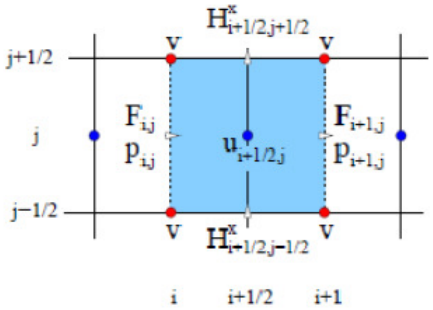


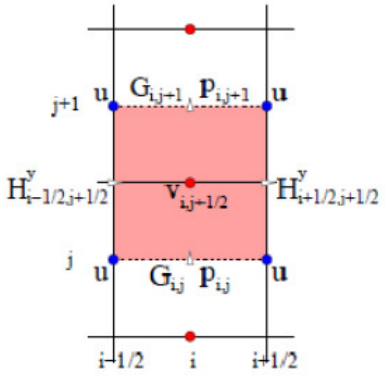
Lid-Driven Cavity Flow
Staggered Storage

- Primitives

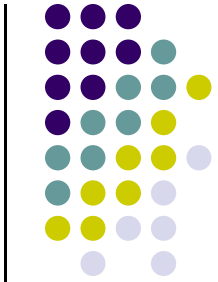


- Fluxes





Lid-Driven Cavity Flow
Pressure Poisson Equation Solver



- The Pressure Correction step was used to make the velocity field at n+1 divergence free.
- Pressure Equation at (i, j) was discretized using a central difference scheme.

$$\delta_x^2|_{i,j}p^{n+1} + \delta_y^2|_{i,j}p^{n+1} = \frac{h}{\Delta t}\left(\delta_x|_{i,j}u^{n+\frac{1}{2}} + \delta_y|_{i,j}v^{n+\frac{1}{2}}\right)$$

where,

$$\delta_x^2|_{i,j}p^{n+1} = p_{i-1,j}^{n+1} - 2p_{i,j}^{n+1} + p_{i+1,j}^{n+1}$$

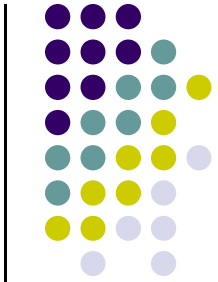
$$\delta_y^2|_{i,j}p^{n+1} = p_{i,j-1}^{n+1} - 2p_{i,j}^{n+1} + p_{i,j+1}^{n+1}$$

- A system of equation of the form [A] . {x} = {b} , was solved using Successive Over Relaxation iterative solver, a modification of the Gauss-Seidel approach.

$$\overline{P_{i,j}^{n+1}} = \frac{1}{4}\left(P_{i+1,j}^{n} + P_{i-1,j}^{n+1} + P_{i,j+1}^{n} + P_{i,j-1}^{n+1}\right) - \frac{h}{\Delta t}\left(\delta_x|_{i,j}u^{n+\frac{1}{2}} + \delta_y|_{i,j}v^{n+\frac{1}{2}}\right)$$

$$P_{i,j}^{n+1} = \omega\overline{P_{i,j}^{n+1}} + (1-\omega)P_{i,j}^{n}$$

Lid-Driven Cavity Flow
Flux Limiting



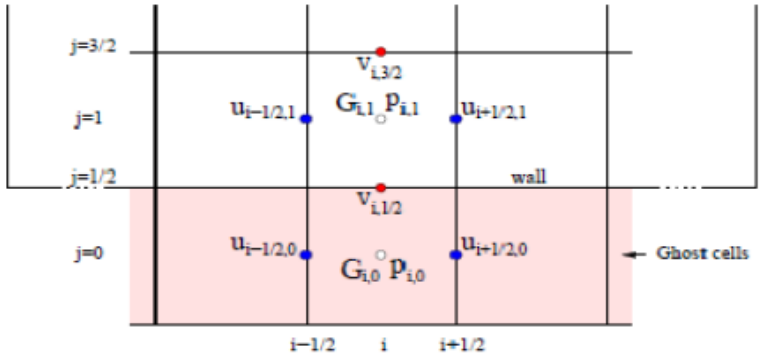
- Simple central differencing yielded a method similar to the FTCS scheme (Conditionally Stable).
- Unphysical oscillations for any reasonable Reynolds' number.

- Quadratic Upstream Interpolation for Convection Kinematics (QUICK)
  - FTCS type oscillations eliminated.
  - Flux = $q\varphi$, q-transported velocity, φ-transported quantity.
  - q computed using Central Difference, φ computed using upwind scheme depending on sign of q.
  - QUICK was a great improvement over simple upwinding, but still suffered from oscillations at higher Re.

- The Simple Monotone for Realistic Transport (SMART) limiter was used for High Re flows.

Lid-Driven Cavity Flow
Boundary Conditions

- At a wall,
  - No slip condition requires the tangential fluid velocity to match the wall velocity.
  - Zero-Flow through requires the normal fluid velocity to be zero.
- For staggered storage,
  - $H^x = uv - vu_y = 0$ on vertical walls.
  - $H^y = uv - vu_x = 0$ on horizontal wall.
- However, a pressure boundary condition was required which meant that H^x and H^y needed to be evaluated at the walls.
- Ghost point method employed, where fluid extends through the wall by satisfying the boundary condition.



At $y = 0, u_{i+\frac{1}{2},0} = -u_{i+\frac{1}{2},1}$ $(U_{wall} = 0)$

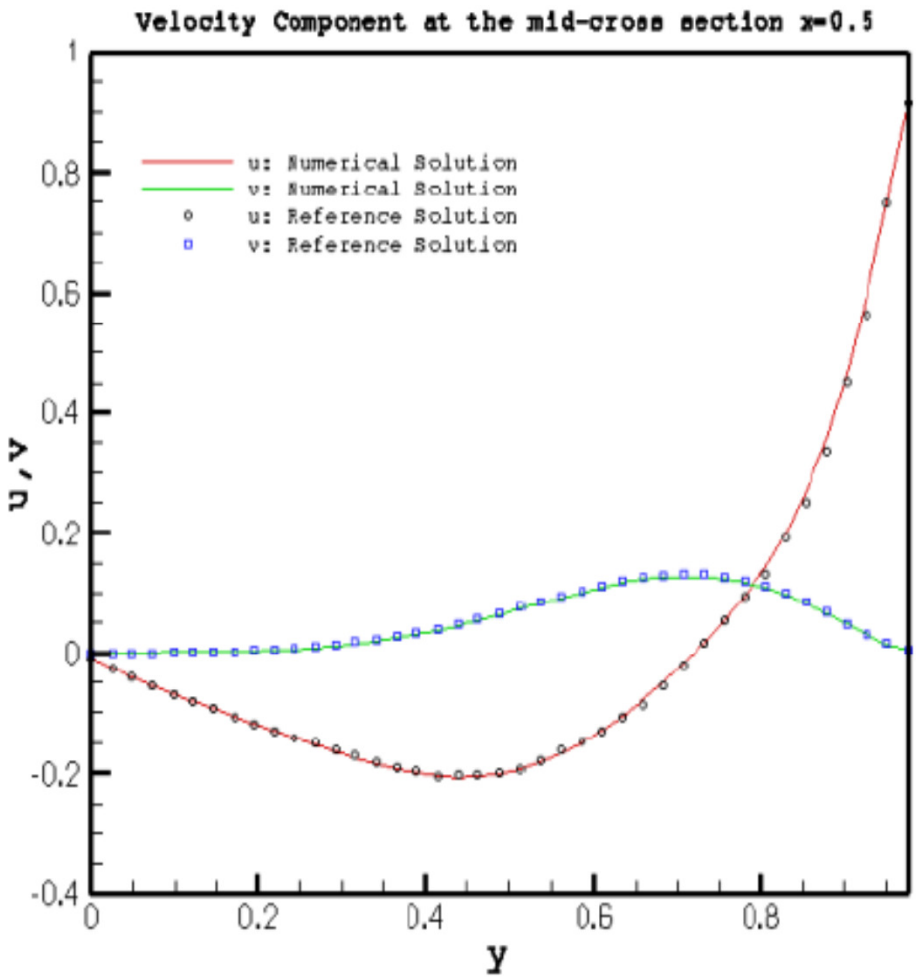
At $x = 0, v_{0,j+\frac{1}{2}} = -v_{1,j+\frac{1}{2}}$ $(V_{wall} = 0)$

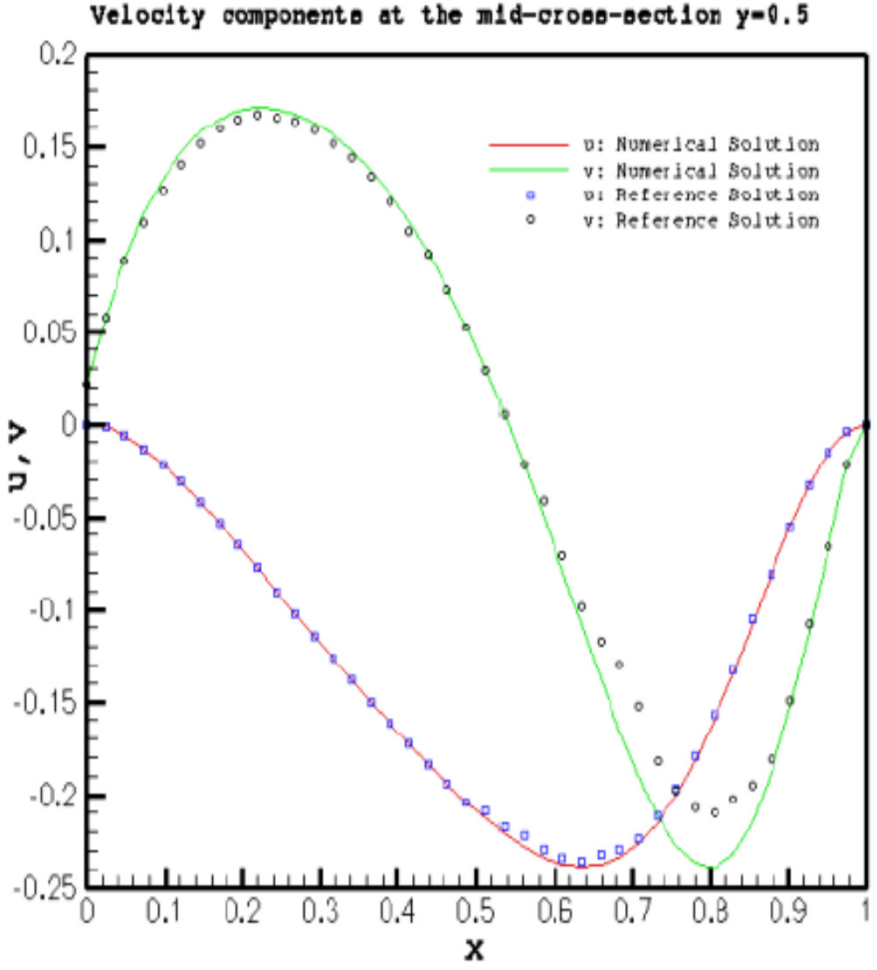
At $y = 1, u_{i+\frac{1}{2},N+1} = 2U_{wall} - u_{i+\frac{1}{2},N}$ $(U_{wall} = 1)$

At $x = 1, v_{N+1,j+\frac{1}{2}} = -v_{N,j+\frac{1}{2}}$ $(V_{wall} = 0)$

Lid-Driven Cavity Flow
Results

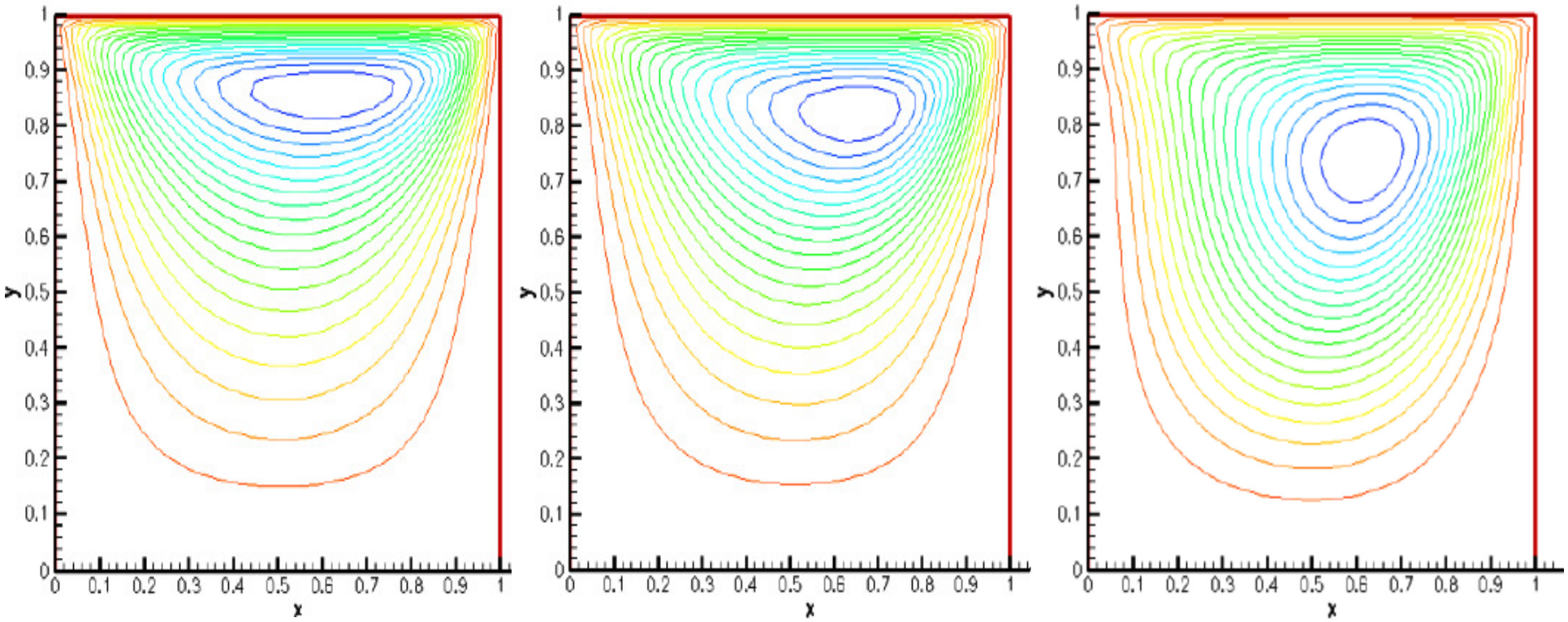
- Re = 100, ρ=1, μ=0.01





Lid-Driven Cavity Flow
Results

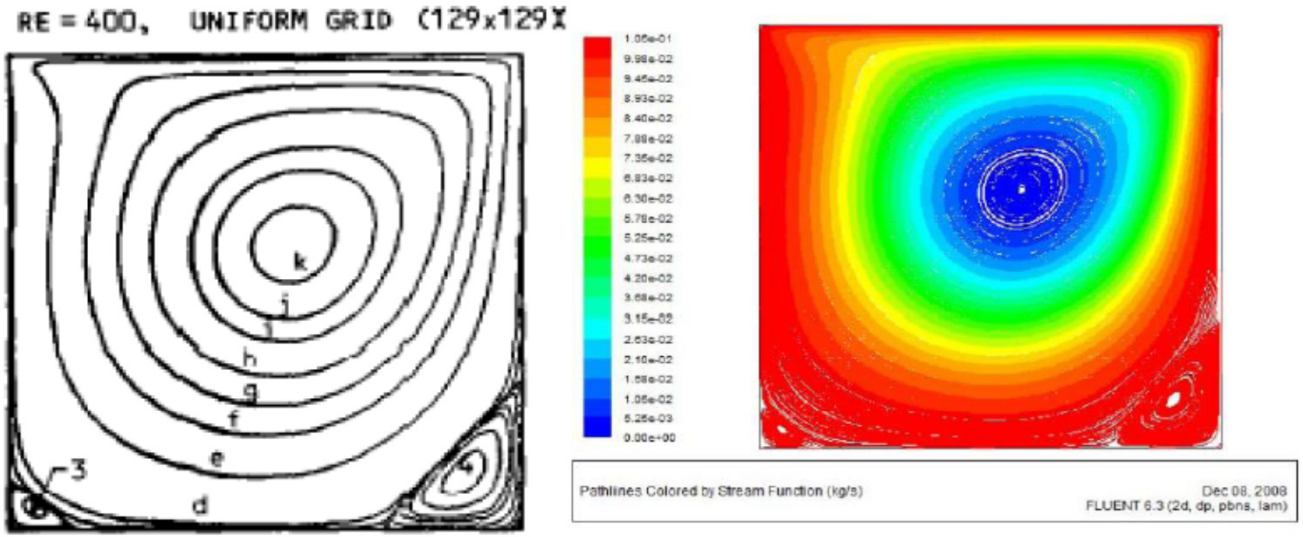
- Streamlines of the flow field; t = 0.5, 1 and10 respectively.

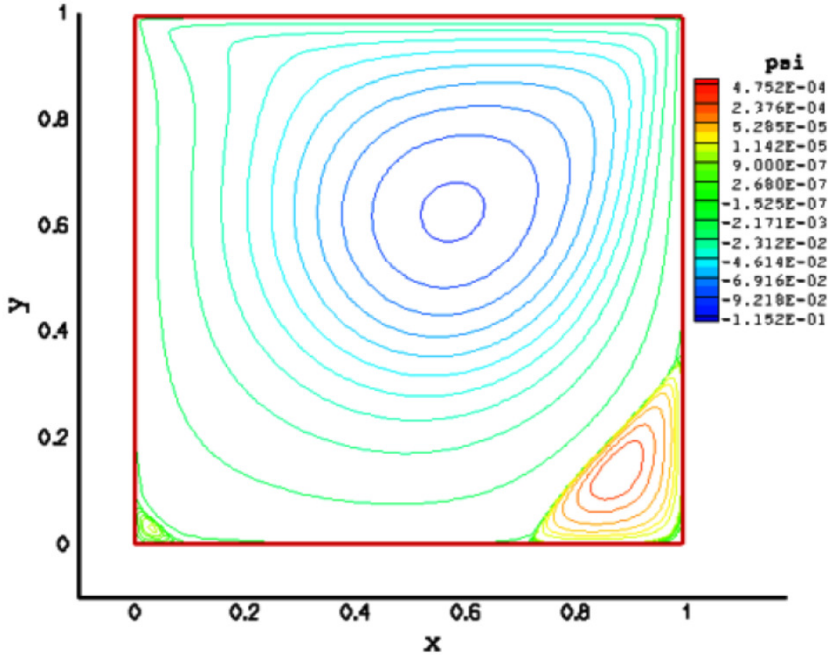


Lid-Driven Cavity Flow

Results

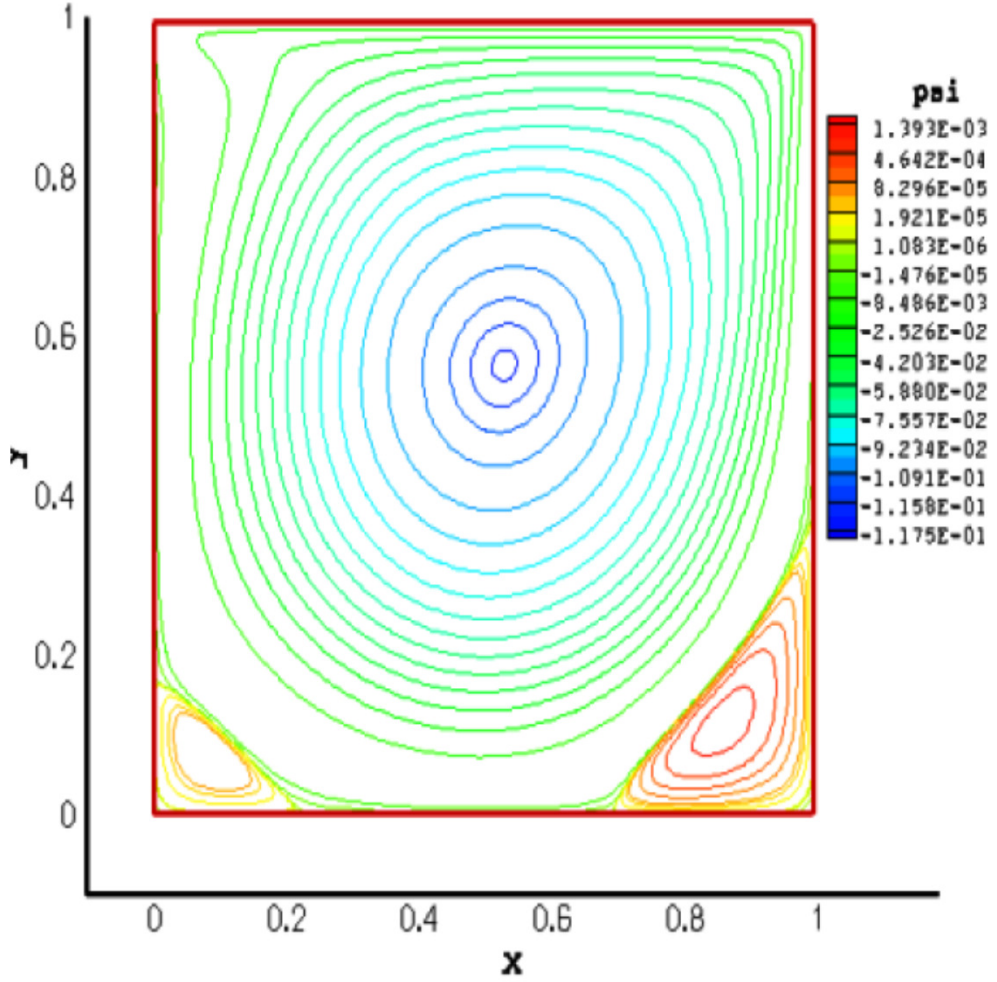
Comparison of Streamfunction, Re=400, with Ghia et. al. and FLUENT™ Solution.





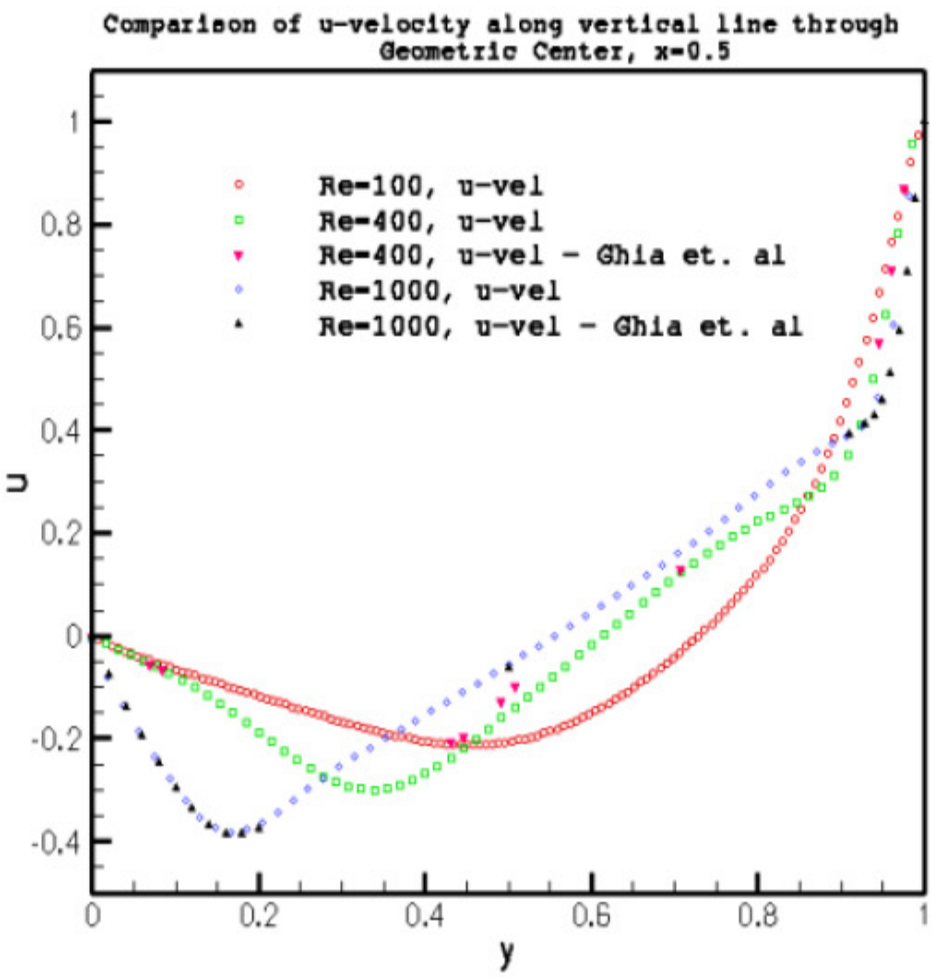
Lid-Driven Cavity Flow
Results

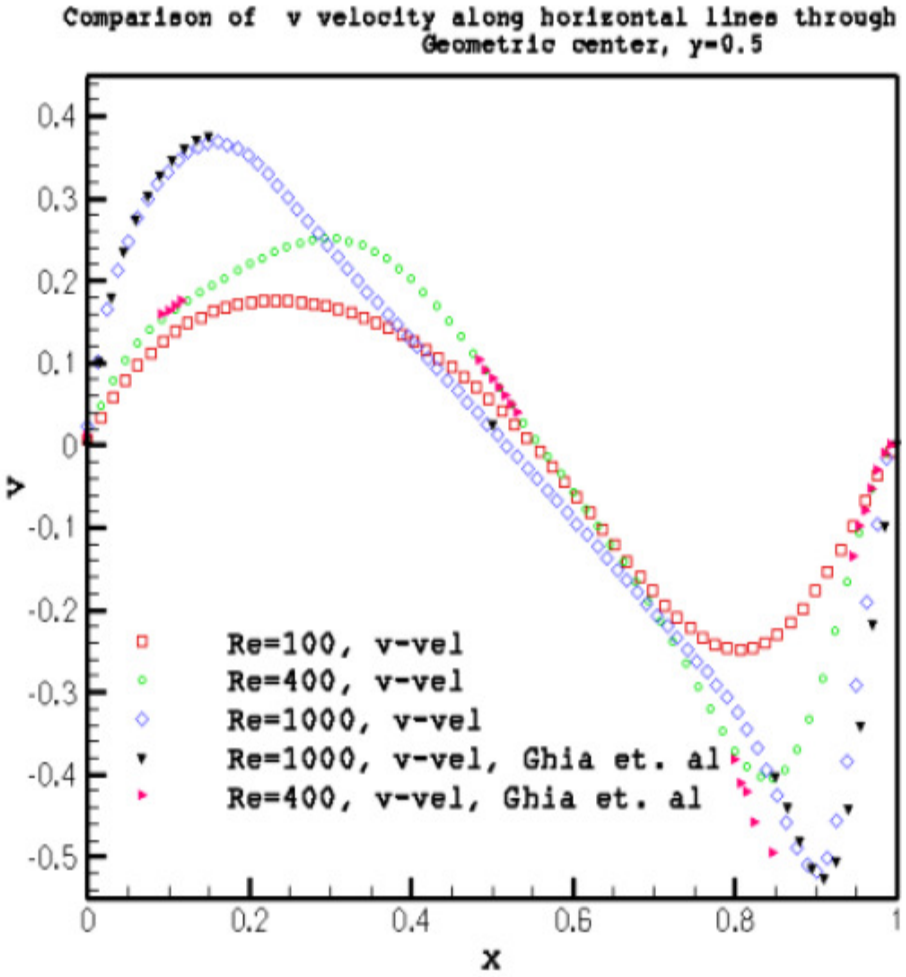
Comparison of Streamfunction high Reynolds number, Re=10000, with Ghia et. al.



Lid-Driven Cavity Flow
Results

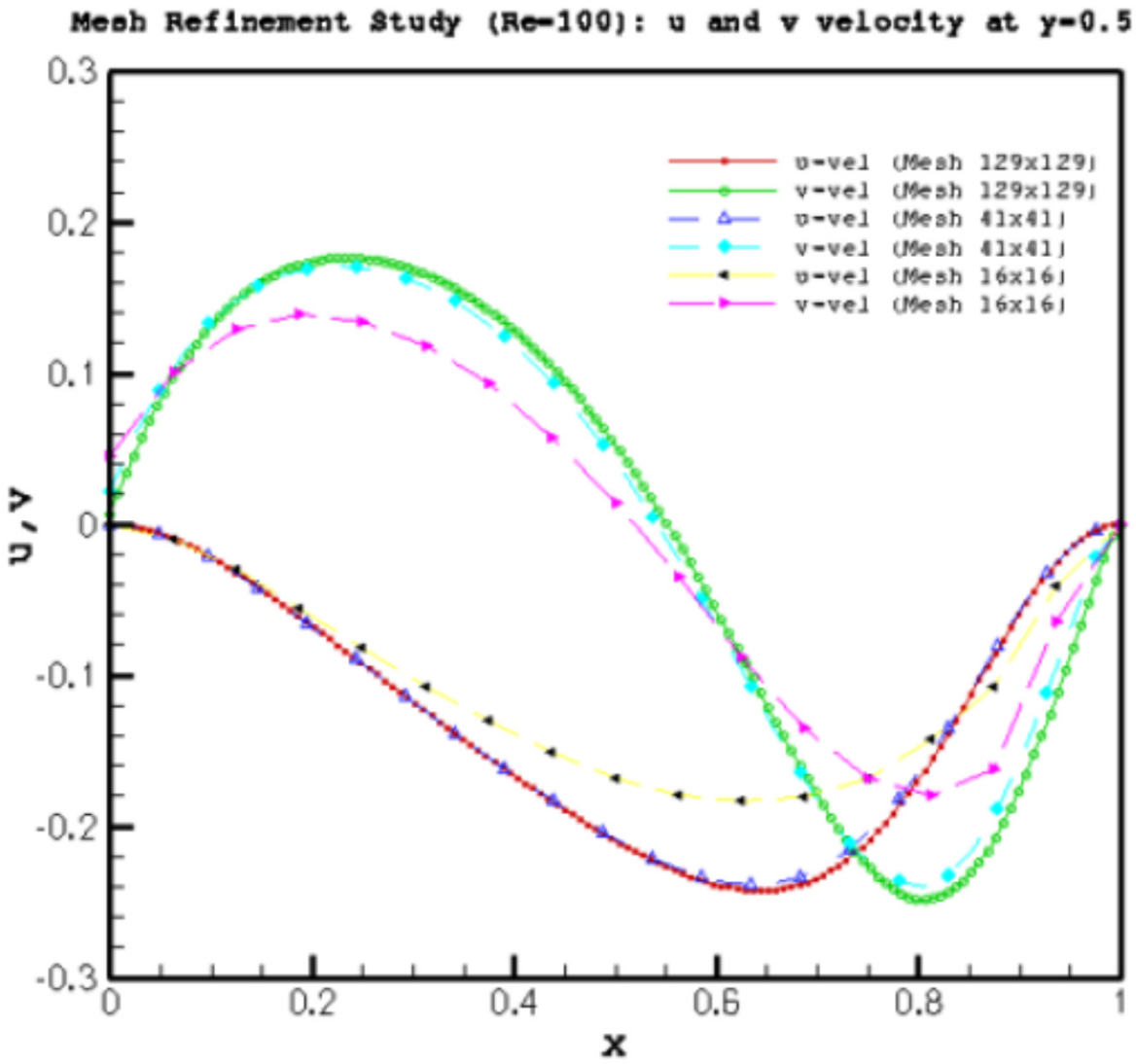
Comparison of Velocities along a vertical line through geometric center, x=0.5, with Ghia et. al.



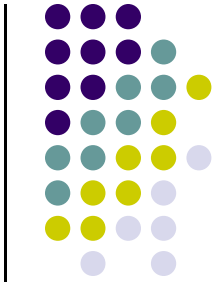


Lid-Driven Cavity Flow
Results

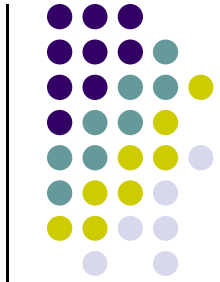
Mesh Refinement Study



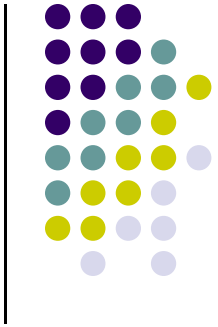
Lid-Driven Cavity Flow
Summary of Results



- Robust Incompressible Navier-Stokes solver developed
  - Solution shows close correlation with Published Results and Commercial Software.
  - Mesh Refinement Study showed that the solution converged with increasing node points.
  - Successfully implemented diverse and effective numerical techniques to solve CFD problems.
  - A working knowledge of key spatial and temporal discretizations used in CFD
- First hand code development experience with Fortran 90
- Strong understanding of fluid flow models and associated numerical challenges



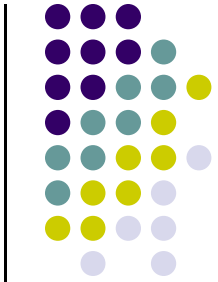
Questions?



Parallel Laminar Shock Boundary Layer Interaction (SBLI)

Parallel Code Development

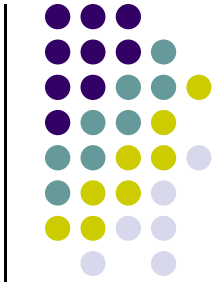
Background



- Relevance of Boundary Layer Shock Interaction
  - Transonic airfoils and wings.
  - Supersonic air intakes and nozzle flows.
  - Deflected control surfaces at transonic or supersonic speeds.

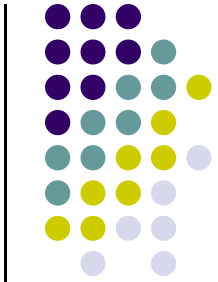
- Motivation
  - Bank of experimental data available at University of Michigan.
  - Building Block for a High-Performance Parallel Code to solve complex flow phenomena for research.
  - Towards the development of an LES solver.
  - Excellent data structure and parallel I/O routines and libraries available from NGA, a code developed at Stanford.
  - Master's Research Project at University of Michigan.

NGA Flow Solver
Overview



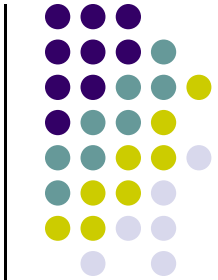
- Parallel variable density, low Mach number Flow Solver.
- Capable of performing Three Dimensional Large Eddy Simulation (LES) of reactive and multi-phase flows.
- Efficient Non-Uniform Structured code with higher order accurate formulation capabilities of spatial derivatives.
- Code written in Fortran 90 with MPI calls to implement parallelization.
- Efficient data structure and Parallel I/O gives a solid foundation to develop a parallel shock boundary layer interaction code.

SBLI Code
NGA Utilization



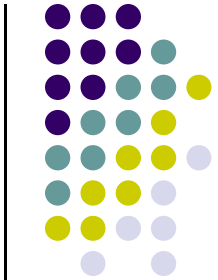
- Data structure and Parallel I/O was extracted from NGA
  - Data structure cleaned to remove unwanted variables and parameter.
  - Primitives converted to conserved quantities as SBLI algorithm developed using conserved quantities.
  - Storage changed from Staggered to Collocated.
  - Parallel I/O adjusted to write cell-centered quantities.
- Flow Initialization performed by NGA Library as this setup was ideal for restarting simulation files.
- Parallel topology and communication extensively utilized and modified to suit current problem.

SBLI Code
Algorithm Overview

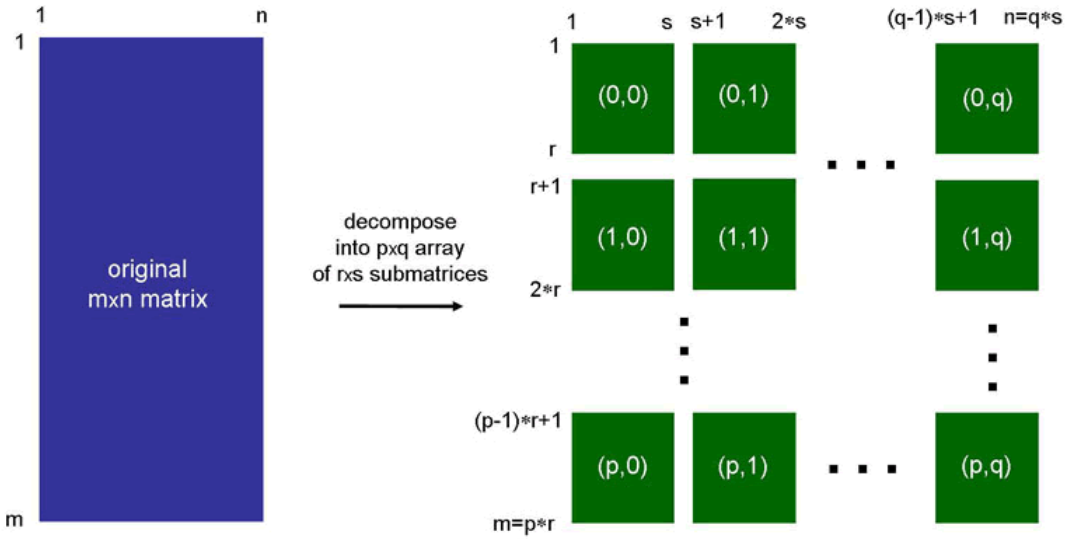


- Governing Equation
  - 3-D Compressible Navier-Stokes equations.
  - Written in Integral form for Finite-Volume Implementation.
- Numerical Method
  - Conservative Finite Volume Method.
  - Quadratic Interpolation (Second order accurate in space) to compute Interface states.
  - Inviscid Flux Computation – Roe Approximate and Exact Riemann Solvers.
  - Viscous Flux Computation – Second order Finite Difference Type.
  - Flux Limiting – Koren Limiter and Double Minmod.
  - Time Integration – Runge Kutta ODE type, third order accurate.
  - Non-uniform grid capability incorporated for better Solution Resolution near walls

SBLI Code
Algorithm Overview



- Parallelization Overview
  - Grid Topology Constructed using MPI_CART_CREATE.

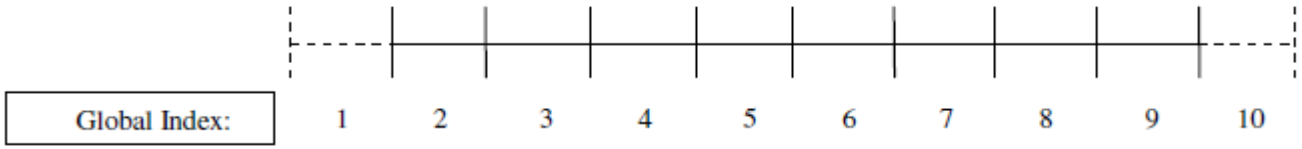


  - New communicator constructed in all 3 dimensions.
  - Solution assumed to be periodic in the z-direction (Quasi 3D Simulation).
  - Ghost cells incorporated in each sub domain for communication among processors and to implement boundary conditions.

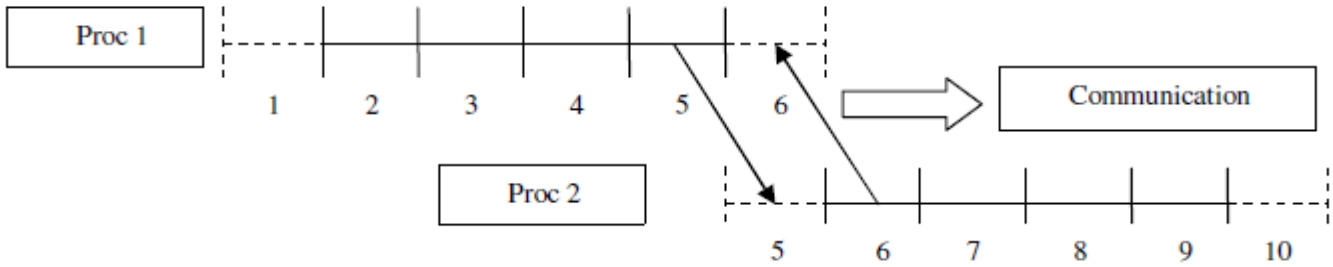
SBLI Code
Algorithm Overview

- Domain Decomposition and Parallel Communication.
  - Parallel mesh decomposed to Local mesh keeping in mind the number of ghost cells
  - Communication carried out at last few cells as demonstrated using MPI_SENDRECV

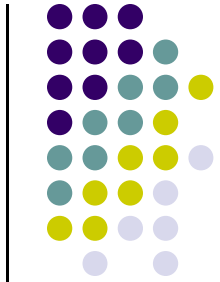
- **Global Mesh with virtual cells: N_x =**



- **Local Mesh with virtual cells: N_x = 8, nproc = 2**



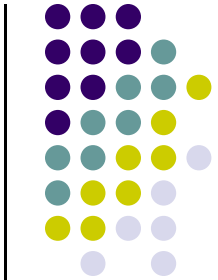
SBLI Code
Algorithm Overview



- Code Snippet to show communication as demonstrated

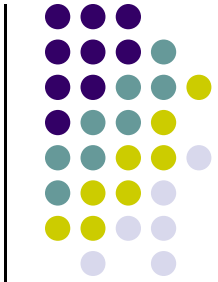
```
! Initialize buffer
allocate(buf1(no,n2,n3))
allocate(buf2(no,n2,n3))
icount = no*n2*n3

! Copy left buffer
buf1(:,:,:) = A(1:no,:,:)

! Send left buffer to left neighbour
call MPI_CART_SHIFT(comm,0,-1,isource,idest,ierr)
call MPI_SENDRECV
     (buf1,icount,MPI_REAL_WP,idest,0,buf2,icount,MPI_REAL_WP,isource,0,comm,istatus,ierr)

! Paste left buffer to right
if (isource.NE.MPI_PROC_NULL) A(n1-no-no+1:n1-no,:,:) = A(n1-no-no+1:n1-no,:,:) + buf2(:,:,:)

! Copy right buffer
buf1(:,:,:) = A(n1-no+1:n1,:,:)

! Send right buffer to right neighbour
call MPI_CART_SHIFT(comm,0,1,isource,idest,ierr)
call MPI_SENDRECV
     (buf1,icount,MPI_REAL_WP,idest,0,buf2,icount,MPI_REAL_WP,isource,0,comm,istatus,ierr)

! Paste right buffer to left
if (isource.NE.MPI_PROC_NULL) A(1+no:1+no+no-1,:,:) = A(1+no:1+no+no-1,:,:) + buf2(:,:,:)
```

SBLI Code
Algorithm Overview



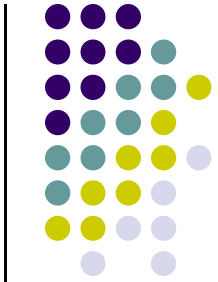
- MPI_SENDRECV Communication Operation
  - Operation is very useful for executing a shift operation across a chain of processes.
  - If blocking sends and receives are used for such a shift, then one needs to order the sends and receives correctly (for example, even processes send, then receive, odd processes receive first, then send) so as to prevent cyclic dependencies that may lead to deadlock.
  - The communication subsystem takes care of these issues.
  - Operation performed in all three directions.

SBLI Code
Problems Solved, Performance Analysis



- Inviscid Shock Reflection Problem
- Compressible Boundary Layer Solution
- Shock Boundary Layer Interaction
- Parallel Performance

SBLI Code
Results – Inviscid Shock Reflection Problem

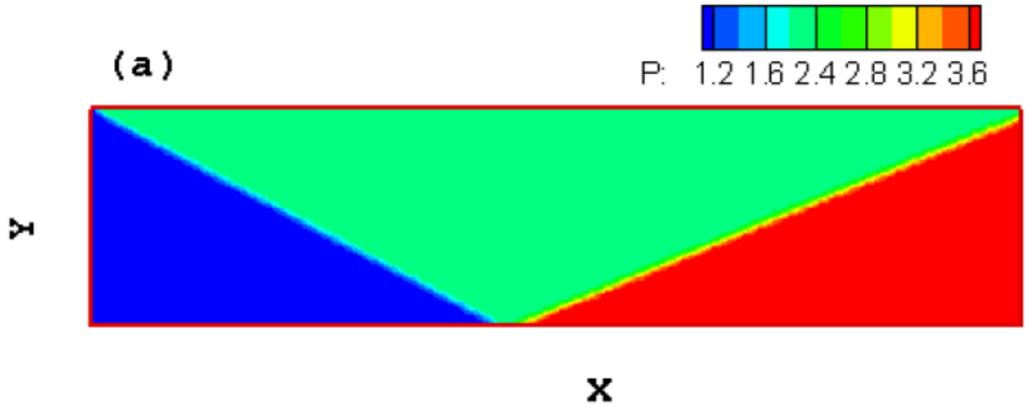


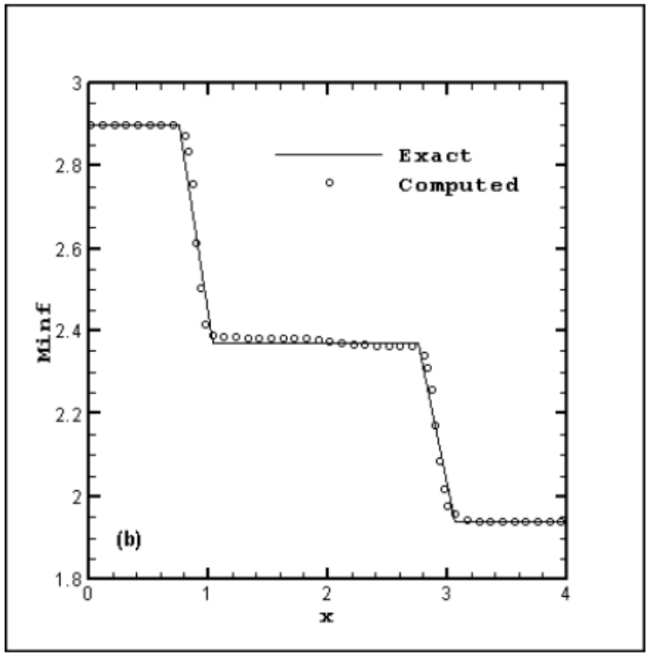
- Solution Domain: $0 \le x \le 4$, $0 \le y \le 1$, $0 \le z \le 1$
- Mesh: 121 x 41 x 21 Uniform Orthogonal Grid
- Parameters: M_∞=2.9, Shock Angle, $\beta = 29°$
- Boundary Conditions
  - South Wall : $v.\boldsymbol{n} = 0$, Ghost point values calculated using a reflection procedure
  - North Wall : Neumann Type Outflow, Second-Order Extrapolation
  - East Wall : Neumann Type Outflow, Second-Order Extrapolation
  - West Wall : Dirichlet Type Inflow, Fixed Inflow Profile
  - Shock created by consistently over specifying variables in two cells using jump relations
  - Front and Back : Periodicity Applied

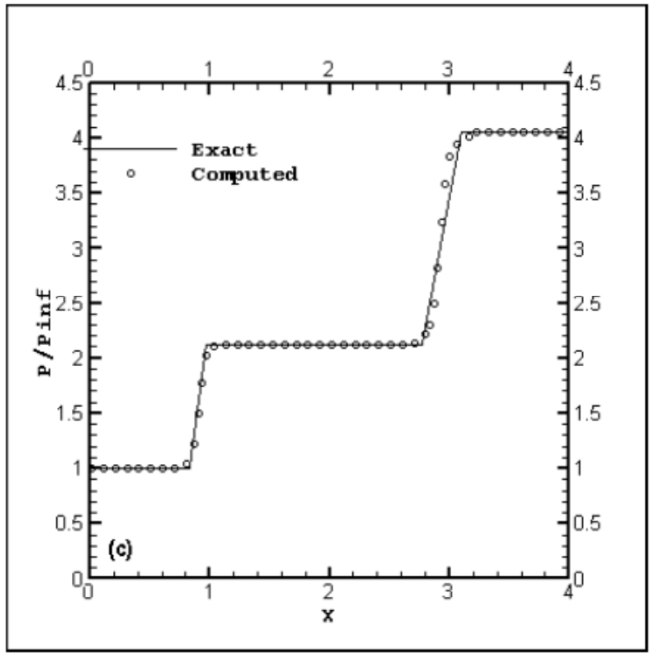
SBLI Code

Results – Inviscid Shock Reflection Problem

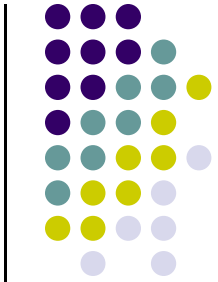
(a) Pressure Contour, (b) M_∞ distribution at y = 0.4878, (c) P distribution at y = 0.4878.







SBLI Code
Results – Compressible Boundary Layer

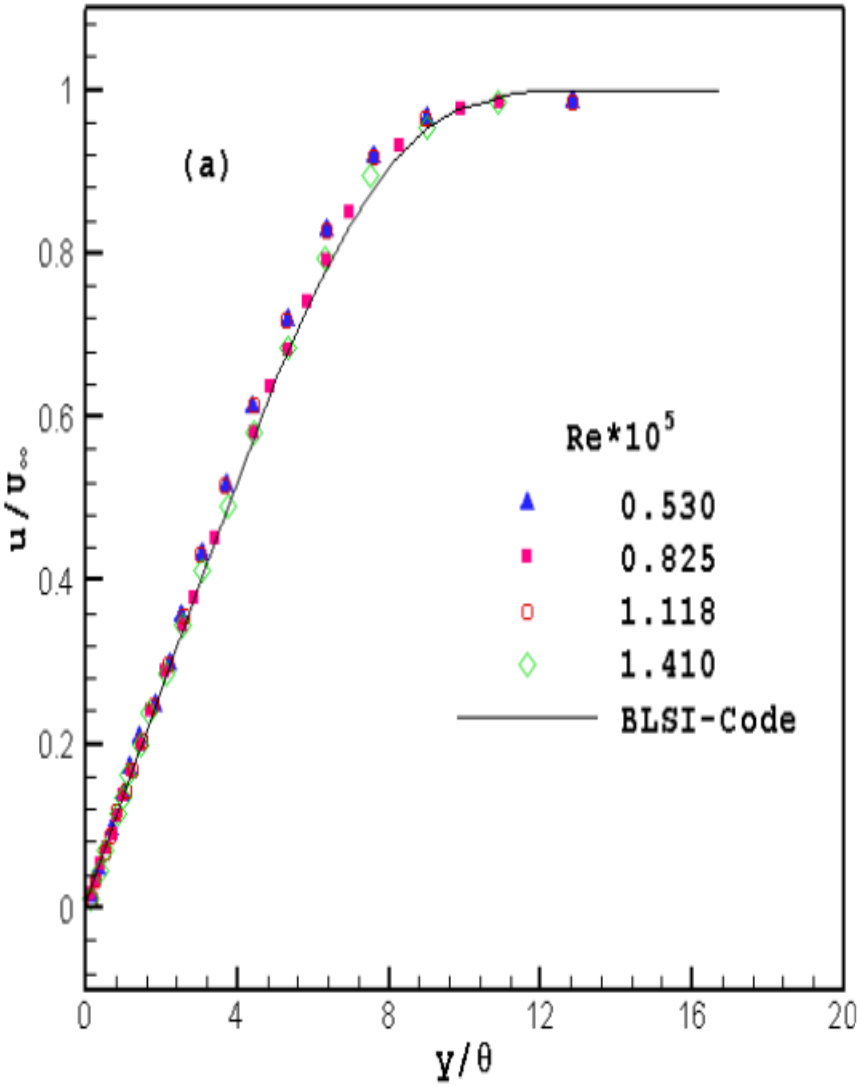


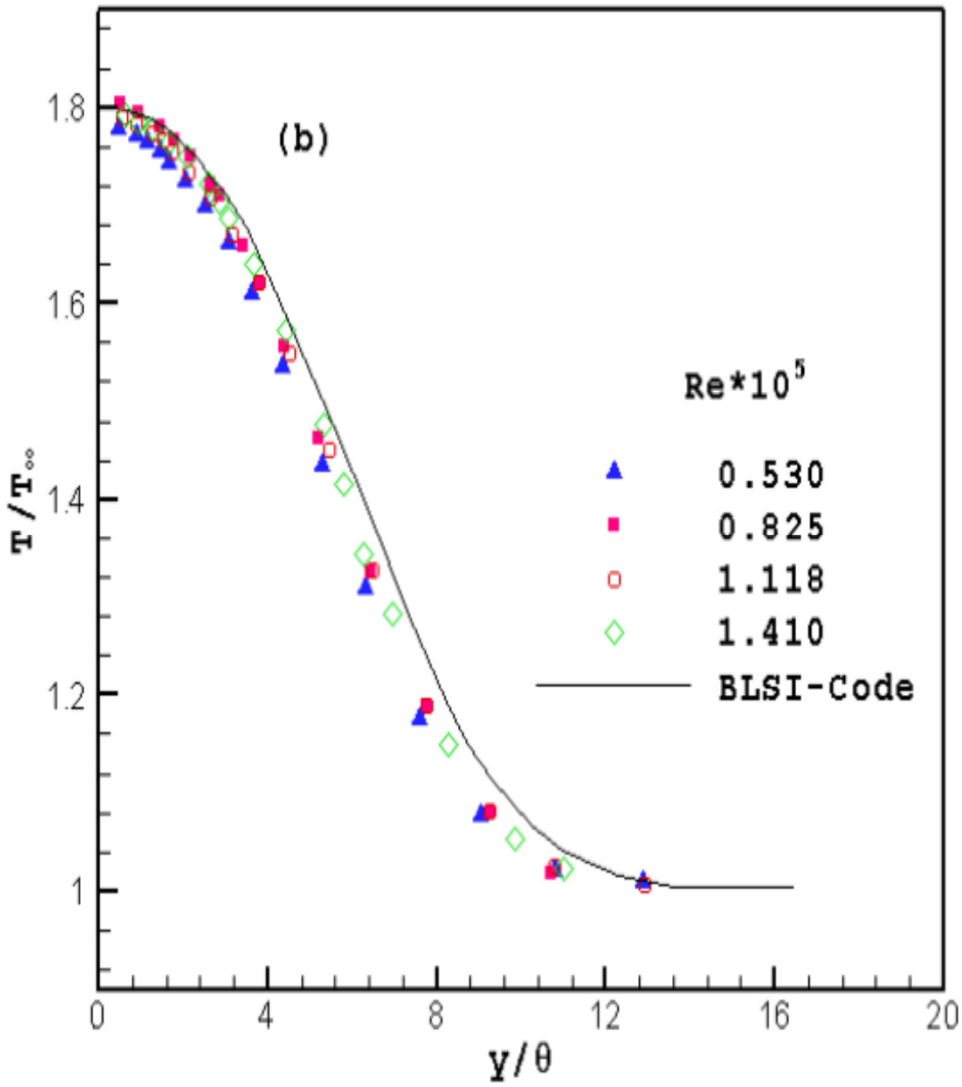
- Solution Domain: $0 \le x \le 4$, $0 \le y \le 2$, $0 \le z \le 1$
- Mesh: 74 x 62 x 21 Non-Uniform Orthogonal Grid
- Parameters: M_∞=2.2, Re = 9.8645×10^4.
- Boundary Conditions
  - South Wall : No-Slip Condition.
  - North Wall : Neumann Type Outflow, Second-Order Extrapolation
  - East Wall : Neumann Type Outflow, Second-Order Extrapolation
  - West Wall : Dirichlet Type Inflow, Analytical Solution Inflow (sine function as prescribe in Schlichting Text.
  - Front and Back : Periodicity Applied

SBLI Code

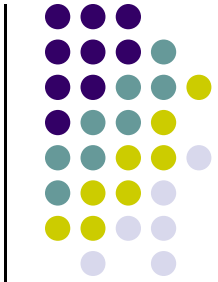
Results – Compressible Boundary Layer

(a) Normalized u-velocity profile, (b) Normalized temperature profile.





SBLI Code
Results – Shock Boundary Layer Interaction

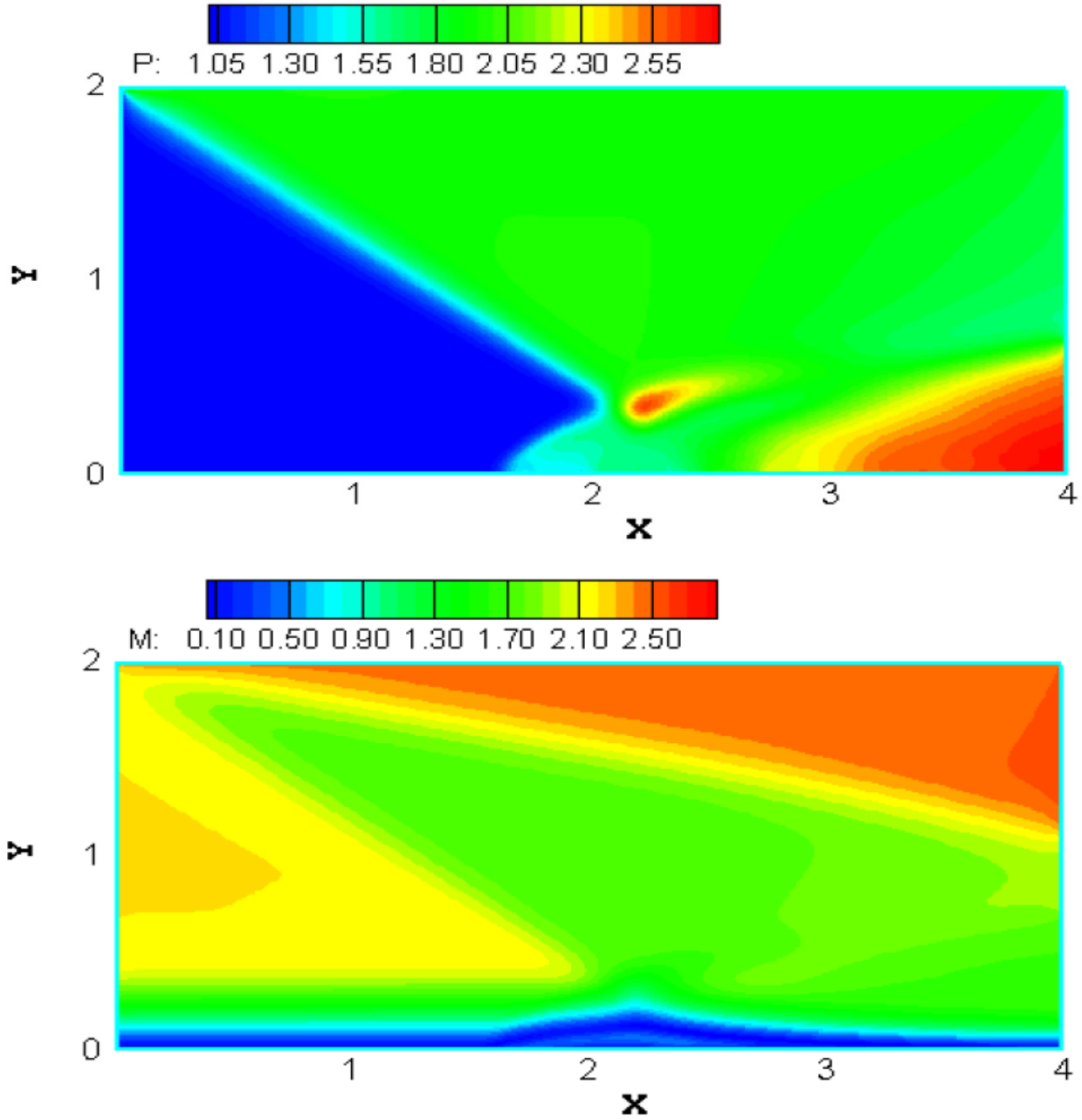


- Solution Domain: $0 \le x \le 4$, $0 \le y \le 1$, $0 \le z \le 1$
- Mesh: 72 x 64 x 21 Non-Uniform Orthogonal Grid
- Parameters: M_∞=2.2, Shock Angle, $\beta = 29°$, Re = 9.8645×10^4
- Boundary Conditions
  - South Wall: No Slip Condition
  - North Wall: Neumann Type Outflow, Second-Order Extrapolation
  - East Wall: Neumann Type Outflow, Second-Order Extrapolation
  - West Wall : Dirichlet Type Inflow, Analytical Solution Inflow (sine function as prescribe in Schlichting Text.
  - Shock created by consistently over specifying variables in two cells using jump relations
  - Front and Back: Periodicity Applied
- Solution Iterated until Converged Compressible Boundary Layer.
- Shock Introduced into the domain and iterated until convergence.

SBLI Code

Results – Shock Boundary Layer Interaction

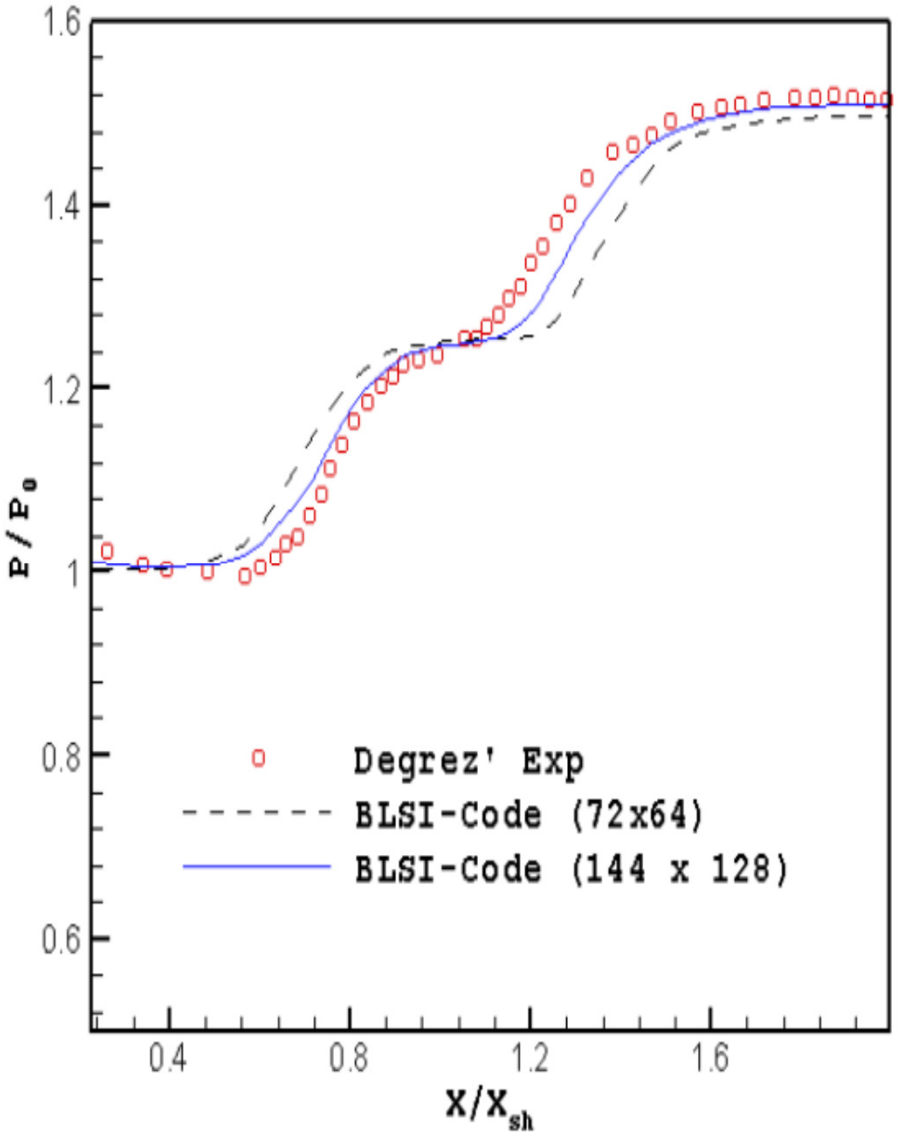
(a) Pressure Contour, (b) Mach Number Contour.

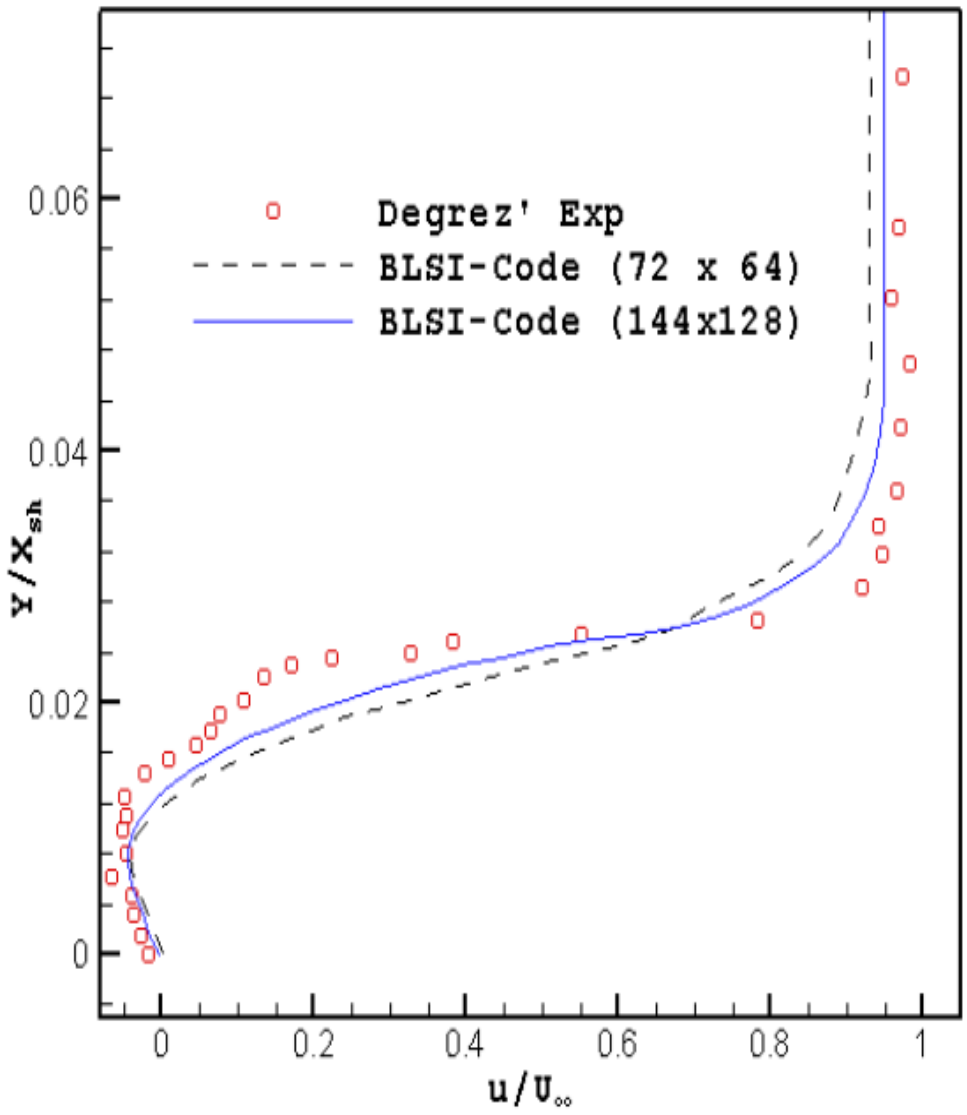


SBLI Code

Results – Shock Boundary Layer Interaction

Mesh Refinement Study

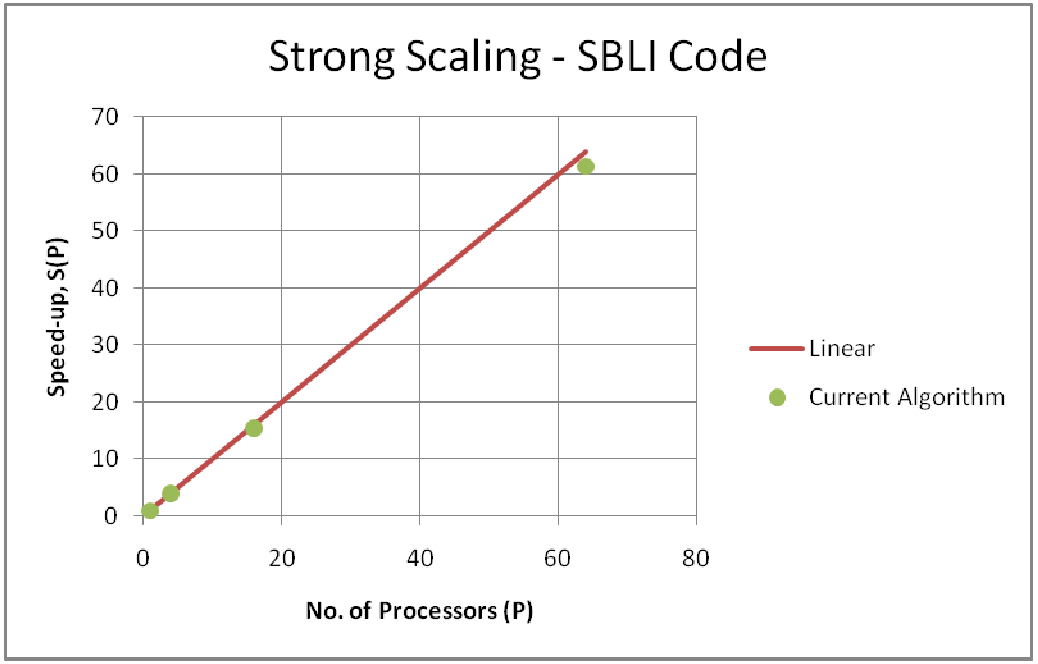




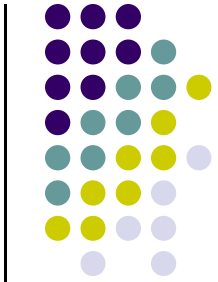
SBLI Code
Results – Shock Boundary Layer Interaction

- Parallel Performance – dual-dual core Opetron CPUs.
  - Strong Scaling (Mesh 144x128x21)

Number of Processors	Time (hrs)	Speed-up	Efficiency (%)
1	7.12	1	100
4	1.79	3.97765363	99.44134078
16	0.46	15.4782609	96.73913043
64	0.116	61.3793103	95.90517241



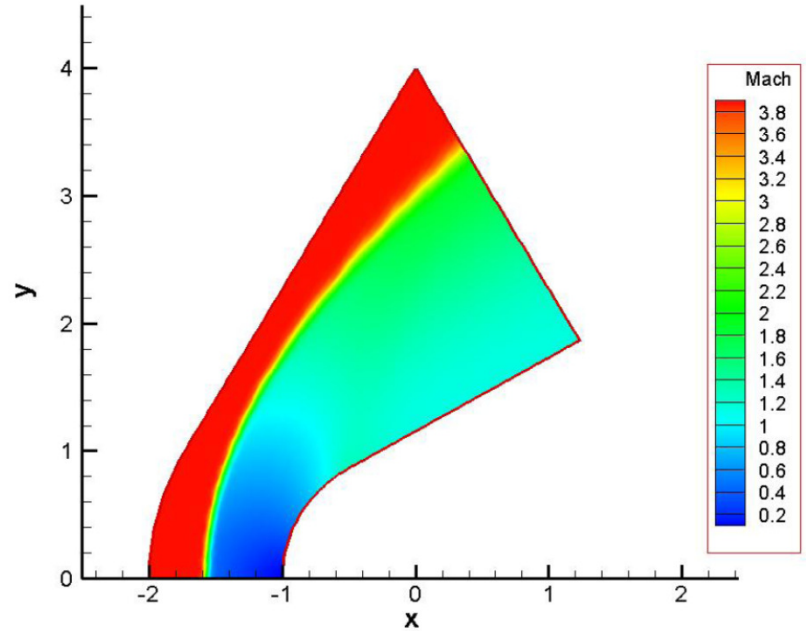
SBLI Code
Summary of Results

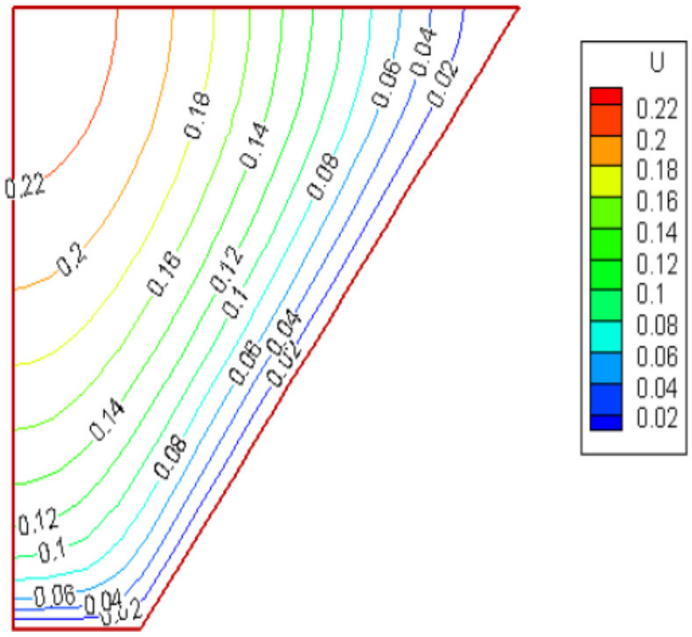


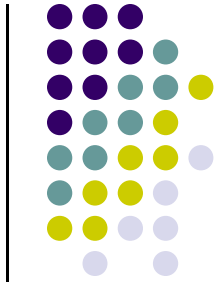
- Parallel Laminar Shock Boundary Layer Interaction Code
  - Results validated and shows close correlation with experimental data.
  - Discrepancies in Pressure and Velocity profiles (SBLI) seen.
  - Experimental results not completely accurate due to the presence of seed-particles.
  - In order to gain confidence a grid convergence study conducted.
  - Solution convergence observed.
- Code Successfully Parallelized
  - Shows strong scaling with increasing processors.
  - Loss in efficiency encountered with P≥64.
  - Code timing not repeated for multiple cycles and this could give more confidence in Speed-Up calculations.

Conclusion

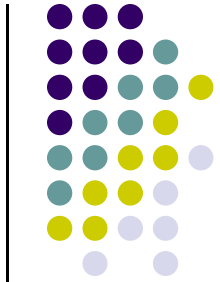
- Solid Engineering background in Computational Fluid Dynamics and Numerical Methods.
- Vast and Diverse experience in code development.
- First hand experience in Parallel Computing.
- Enthusiastic and Quick Learner.
- Passion for design problems in variety of Engineering Problems.







Questions?



Thank You!