A Parallel 3D Flow Solver in Python Based on Vortex Methods

Prashant Agrawal Varun Puri Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

SciPy.in - 2010 Conference
Indian School of Business, Hyderabad

Outline

Outline

What did we do

- Developed a solver to simulate flow which is:
    - based on grid-free vortex methods
    - able to handle 3D flows around arbitrarily shaped bodies
    - based on a generic and extensible design
    - completely parallelized
    - well tested

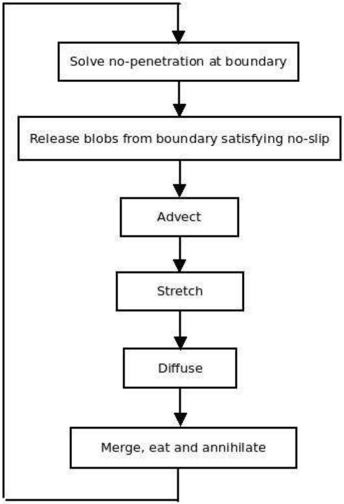
Modeling Flow using Vortex Methods

- Flow is modeled by a set of vortex particles
- Velocity field generated by these vortex particles simulates flow velocity
- These particles advect, stretch and diffuse according to Navier-Stokes equations

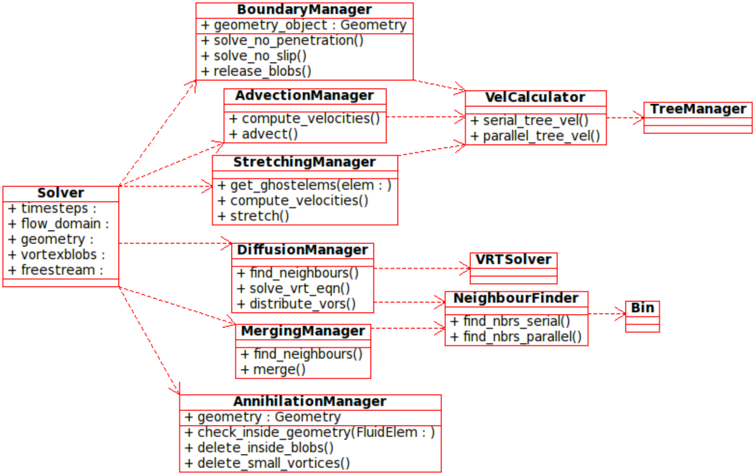
Outline

Solution Procedure

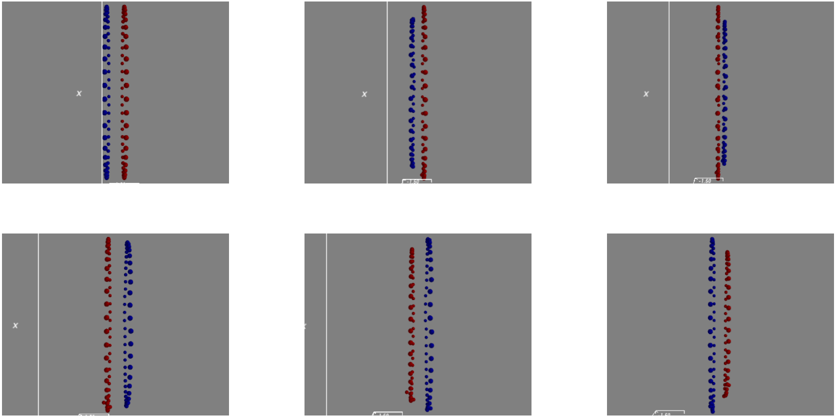
Steps involved in each iteration



Code Design


BoundaryManager
+ geometry_object : Geometry
+ solve_no_penetration()
+ solve_no_slip()
+ release_blobs()
AdvectionManager
+ compute_velocities()
+ advect()
VelCalculator
+ serial_tree_vel()
+ parallel_tree_vel()
TreeManager
StretchingManager
+ get_ghostelems(elem :)
+ compute_velocities()
+ stretch()
Solver
+ timesteps :
+ flow_domain :
+ geometry :
+ vortexblobs :
+ freestream :
DiffusionManager
+ find_neighbours()
+ solve_vrt_eqn()
+ distribute_vors()
VRTSolver
NeighbourFinder
+ find_nbrs_serial()
+ find_nbrs_parallel()
Bin
MergingManager
+ find_neighbours()
+ merge()
AnnihilationManager
+ geometry : Geometry
+ check_inside_geometry(FluidElem :)
+ delete_inside_blobs()
+ delete_small_vortices()

A Test Case: Leap-frogging of identical vortex rings



Outline

Domain Decomposition

- Aim of particle distribution: Nearby particles in same processors, Far-off in different
- SFC curves: Particles sorted according to a position based key

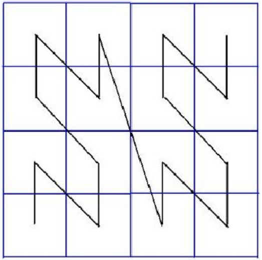


Figure: Typical SFC curve

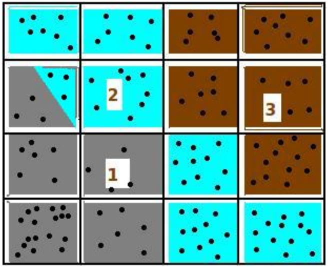
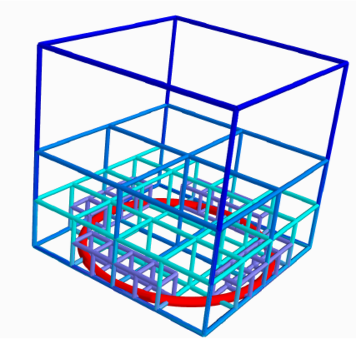


Figure: Particle distribution among processors obtained using SFC

Velocity Computation: Serial

- Velocity due to nearby particles: calculated directly
- Velocity due to far-off particles: calculated using an approximation known as pseudo particles
- Velocity calculation based on an oct-tree data structure



Velocity Computation: Parallel

- Local tree traversal
- Collective Communication
- Selective Communication

Neighbour Finder

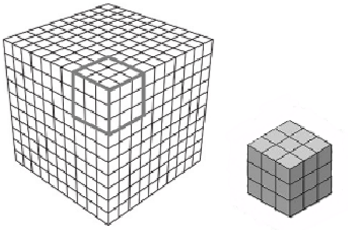


Figure: Neighbour Search Domain

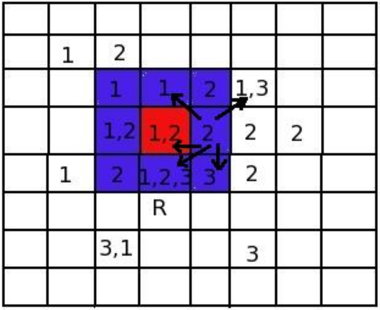


Figure: Parallel Neighbour Finder

Parallelization Scale-up

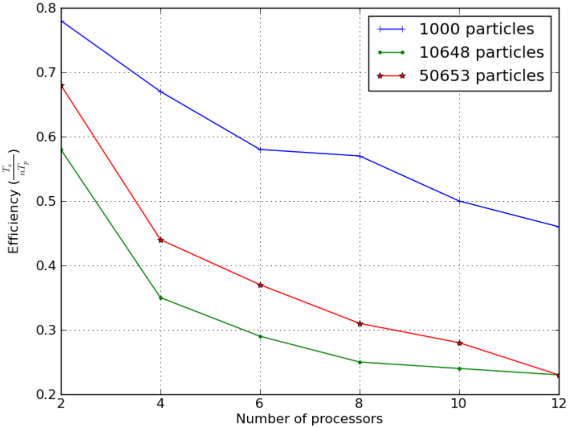


Figure: Parallelization scale-up efficiency for different number of particles

Outline

Generic Design

- General enough to handle various types of particles: pseudo particles, boundary blobs, stretching ghost particles, regular vortex blobs etc.
- Enables lot of code re-use: e.g. same `VelCalculator` is used by `Boundary`, `Advection` and `Stretching` Managers
- New algorithms can be easily added to existing framework

Importance of Using Python

Advantages

- Shorter development time
  - 3 months to develop a generalized parallel flow solver
- Shorter code-base
  - 4400 lines (including test modules)

Disadvantages

- Longer run time
  - planning to Cythonize this code for speed-up

Parallel Code

- Used `mpi4py` for all parallel implementations
- `mpi4py` closely follows all MPI-2 C++ bindings
- Used `qsub` for effective process management
- Issues with parallel debugging

Testing

- Extensive testing using Python's `unittest` module and `nose`
- 48 test functions to test 56 functions
- Almost every part of the code runs atleast once during these tests
- 1250 lines out of 4400 lines code-base are test modules

Future Work

- Improve parallelization scale-up
- Cythonize
- Solve some test cases involving boundaries
- Interpolate vorticity on grid points for flow visualization