Speedup of 3D Simulations for Complex Moving Body Problems by OpenMP

Yu Peng

Department of Mechanical Engineering
National University of Singapore

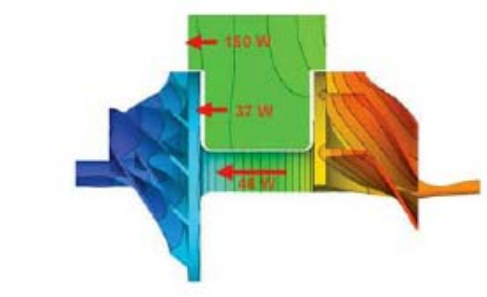
Outline of the Presentation

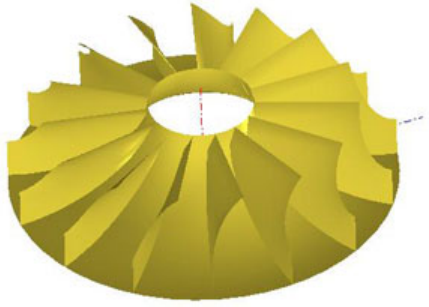
- **Introduction**
  - Background
  - Objective
  - Significances
- **Numerical Method**
- **Parallelization**
  - Why OpenMP
  - Parallelization Strategy
  - Efficiency & Speedup
- **Applications**
- **Conclusions**

Introduction

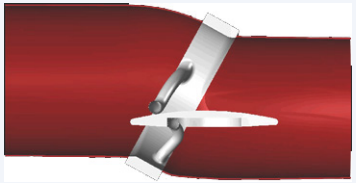
➢ Background

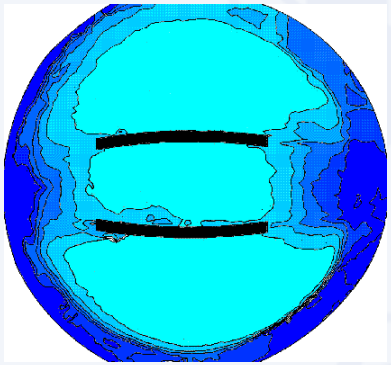
- o Many flows involves moving boundary
  1. *Flows in turbomachines*





  2. *Flows through artificial heart valves*





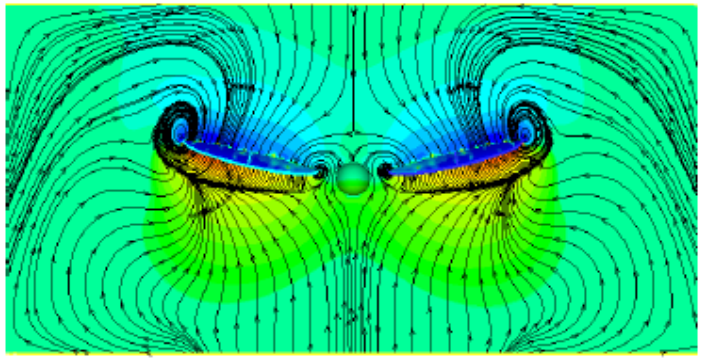
Introduction

➢ Background

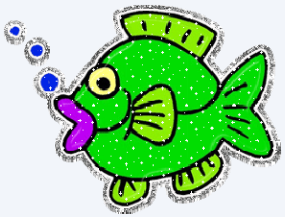
- o Many flows involves moving boundary

3. Flapping wing





4. Fish swimming





Introduction

➢ Background

- o Numerical method in literature

1. *Conventional FEM & FVM*
   - ✓ ALE method, Fully developed & widely applied
   - ✓ Grid regeneration

2. *Cartesian grid methods*
   - ✓ Simple & Efficient
   - ✓ Low order of accuracy

3. *Overset or chimera grid methods*
   - ✓ Different domains, Coordinates
   - ✓ Data interpolating

Introduction

- **Objective**
  - Develop a fresh approach
    - minimize some of the cumbersome and potentially error-prone features of existing schemes
    - strengthen numerical representation in the area of the boundary to keep a high accuracy
  - Novel hybrid meshfree-Cartesian grid approach
    - Cartesian grid for whole domain
    - Meshfree grid for the moving body and its immediate fluid neighbourhood

Introduction

- **Significance and Impact of Research**
  - Numerical Significances
    - easy to build high-order scheme
    - good efficiency
    - precise implementation of boundary conditions
    - grid flexibility
  - Application Impact
    - many flows including insects or birds flying
    - help to understand the complex and often novel aerodynamics underlying the flight of insect or birds
    - spark novel ideas on engineering applications such as flow control, thrust production etc.
    - develop novel propulsive mechanisms that emulate the motion of flapping wing

Numerical Aspects

➢ Governing Equations

$$\partial_t \mathbf{u} + (\mathbf{u} - \mathbf{u}_c)\nabla \mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2 \mathbf{u}$$

$$\nabla \bullet \mathbf{u} = 0$$

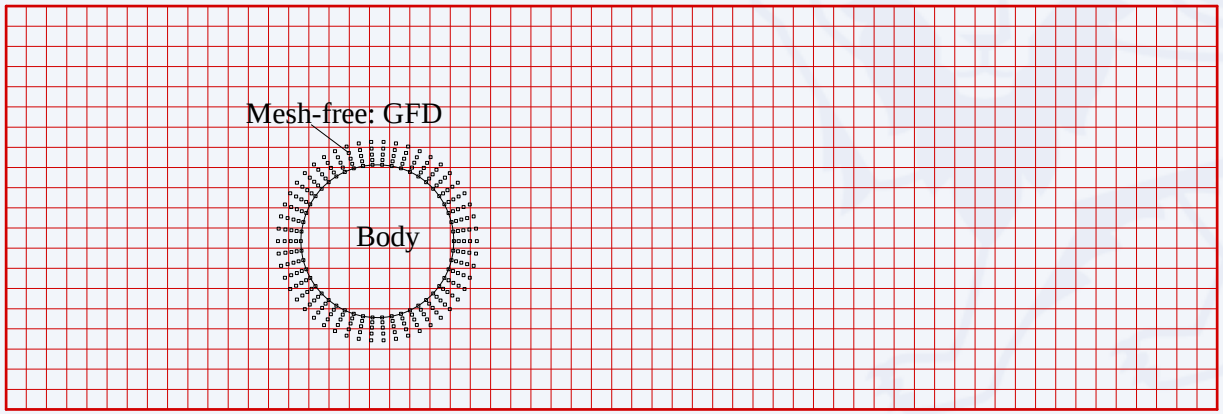
- $\mathbf{u}$: velocity vector
- p: pressure field,
- $\mathbf{u}_c$: convection velocity of the computational node.
- Re: Reynolds number, Re = UL/ν; U and L are reference velocity and length, ν is the kinematic viscosity.

Note: $\mathbf{u}_c = 0$ for the background Cartesian nodes

Numerical Aspects

➢ Discretization

- **Cartesian nodes : standard finite difference method**
- **Mesh-free nodes : generalized finite difference method (GFD)**
- **Body Motion: mesh-free nodes convect against Cartesian nodes**



Numerical Aspects

➢ GFD-SVD

- o Generalized Finite Difference (GFD):

$$f(\mathbf{x}_1) = f(\mathbf{x}_0) + \sum_{1 \le j_1+j_2+j_3 \le m-1} \frac{\Delta x_1^{j_1} \Delta y_1^{j_2} \Delta z_1^{j_3}}{j_1!\, j_2!\, j_3!} \left[\partial_x^{j_1} \partial_y^{j_2} \partial_z^{j_3} f \right]_{\mathbf{x}_0} + O(|\Delta \mathbf{x}_1|^m)$$

Truncated after the third-order derivative:

$$\Delta \mathbf{f}_{n\times 1} = [S]_{n\times 19}\, \partial \mathbf{f}_{19\times 1}$$

$$\partial \mathbf{f}_{19\times 1} = (\partial_x, \partial_y, \partial_z, \partial_x^2, \partial_x \partial_y, \cdots, \partial_y^3, \partial_z^3)^T f\,|_{\mathbf{x}_0}$$

$$\Delta \mathbf{f}_{n\times 1} = (f_1 - f_0, f_2 - f_0, \cdots, f_n - f_0)^T$$

$$[S]_{n\times 19} = \begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 & 0.5\Delta x_1^2 & 0.5\Delta y_1^2 & 0.5\Delta z_1^2 & \cdots \\ \Delta x_2 & \Delta y_2 & \Delta z_2 & 0.5\Delta x_2^2 & 0.5\Delta y_2^2 & 0.5\Delta z_2^2 & \\ \Delta x_3 & \Delta y_3 & \Delta z_3 & 0.5\Delta x_3^2 & 0.5\Delta y_3^2 & 0.5\Delta z_3^2 & \\ \cdots\cdots & & & & & & \\ \Delta x_n & \Delta y_n & \Delta z_n & 0.5\Delta x_n^2 & 0.5\Delta y_n^2 & 0.5\Delta z_n^2 & \end{bmatrix}$$

Numerical Aspects

➢ SVD

o Characterization of equation system

- 19 supporting points – the matrix is often ill-conditioned
- Additional supporting nodes are needed to yield an over-determined algebraic system (n > 19)
- Seeking a best-approximation solution based on the minimization of the-norm $L_2$of the residual error vector

$$\mathbf{E} = \Delta\mathbf{f}_{n\times 1} - [S]_{n\times 19}\,\partial\mathbf{f}_{19\times 1}$$

- Least square-like method

Numerical Aspects

➢ SVD

o Singular value decomposition

Any $m \times n$ matrix A can be written as:

$$A = U \Sigma V^T$$

U and V are orthogonal matrices;

Σ is a diagonal matrix with diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$

$p = \min\{m, n\}$

The pseudoinverse of matrix A in SVD

$$A^{-1} = \left(U\Sigma V^T\right)^{-1} = \left(V^T\right)^{-1}\left(U\Sigma\right)^{-1} = V\Sigma^{-1}U^T = A^+$$

Least square solution of $Ax = b$ in terms of the pseudoinverse A^+ in SVD is given by

$$x = A^+ b$$

Parallelization

- **OpenMP**
  - Portable, shared-memory threading API
  - Standardize task & loop-level parallelism
  - support incremental parallelization
  - Combine serial and parallel code in single source
  - Languages supported: Fortran and C/C++
  - Multi-vendor support for both Linux and Windows

Parallelization

➢ OpenMP vs MPI

- o MPI-Message Passing Interface
  - utilizes a distributed system of processors with local memory
  - domain decomposition
  - communicate data between adjacent blocks

- o The moving objects may travel from one computational block to another in the course of its motion

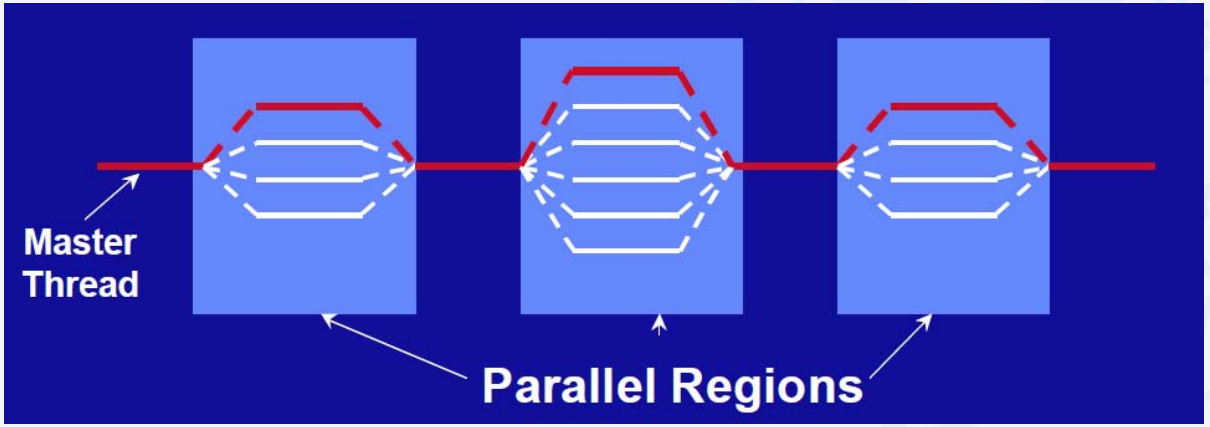
- o Domain decomposition and data communication at the interfaces very difficult

Parallelization

➢ OpenMP Parallelization Strategy

o utilize a form of fork-and-join parallelism

- master thread forks a specified number of slave threads
- the created threads run concurrently
- the slave threads join back to the master thread at the end of parallel operation



Parallelization

➢ OpenMP Parallelization Strategy

- Parallel for loops
  - simple loop

If no dependence between one iteration of the loop and another, we can simple indicate the compiler that the iteration of a for loop may be executed in parallel:

```
#pragma omp parallel for
for(ijk=0;ijk<Point_Meshless;ijk++)
    U_Ale[ijk] = (X_Meshless[ijk]-X_Meshless_Old[ijk])/Dt;
```

Parallelization

➢ OpenMP Parallelization Strategy

- Parallel for loops
  - nested loops

    For the doubly nested loops in which either loop could be executed in parallel, it is better to parallelize the outer loop so that the program only incurs the fork/join overhead once:

```
#pragma omp parallel for private(i)
for(j=0;j<IPoint;j++)
    for(i=0;i<JPoint;i++)
        A[i][j]= B[i][j]+ C[i][j];
```

Parallelization

➢ OpenMP Parallelization Strategy

- o Parallel for loops
  - reduction clause

The reduction clause is used to sore partial sums in private variables and then adding the partial sums to the shared variable after loop:

```
#pragma omp parallel for private(tmp2) reduction(+: tmp3)
    for(i=0;i<Point_Meshless;i++)
    {
        tmp2=FI[i];
        FI[i]=(U[i]-So[i]);
        tmp3+=(tmp2-FI[i])*(tmp2-FI[i]);
    }
```

Parallelization

➢ OpenMP Parallelization Strategy

- Parallel for loops
  - bulky loop

For a bulky loop, the statements can be encapsulated as a function to minimize scoping required and reduce error-prone:

```
#pragma omp parallel for shared (nb)
for(it=0;it<Point_Meshless;it++) Node_Search_A(it, nb);
```

Parallelization

➢ OpenMP Parallelization Strategy

o Parallel task

An important new feature of OpenMP is that it allows parallelizing a linked list of task. A linked class can be parallelized as:

```
#pragma omp parallel
{
     #pragma omp single
     {
          Current_Meshless_Member=C2M_Object->C2M_Head_Ptr;
          while(Current_Meshless_Member != NULL)
          #pragma omp task
               Update_Velocity_B(Current_Meshless_Member);
          Current_Meshless_Member = Current_Meshless_Member ->Next_Member;
      }
 }
```

Parallelization

➢ OpenMP Parallelization Strategy

o Functional parallelism

We can assign different threads to different portion of code:

```
#pragma omp sections
{
    #pragma omp section
    <code block1>
    #pragma omp section
    <code block2>
    #pragma omp section
    :
 }
```

Parallelization

➢ OpenMP Parallelization Strategy

- Use of OpenMP runtime library routines

omp_get_num _threads: obtain the number of active threads

omp_get_thread_num: retrieve the unique identification number of a thread

```
#pragma omp parallel private(tid)
{
    tid=omp_get_thread_num();
     t= omp_get_num _threads;
    for(i=tid;i<Point_Meshless;i=i+t)
      <code block>
}
```

Parallelization

➢ Speedup & Efficiency

- Speedup factor (S_P)

  $S_p = T_1/T_p$

- Efficiency (E_P)

  $E_p = T_1/(PT_p)$

P : the number of CPU-cores,
T_1 : the wall clock time of the sequential algorithm,
T_P : the wall clock time of the parallel algorithm with P CPU-cores.

Parallelization

➢ Speedup & Efficiency

Test case : flow induced by a rotating sphere

Intel® C++ Compiler 11.0

	Total CPU No.	CPU Model	Total CPU Cores No.
atlas3-c21	2	Dual-Core Intel Xeon 5160 @ 3.00 GHz	4
atlas4-c80	2	Quad-Core Intel Xeon E5430 @ 2.66 GHz	8
atlas4-c85	4	Quad-Core Intel Xeon E7330 @ 2.40 GHz	16

Parallelization

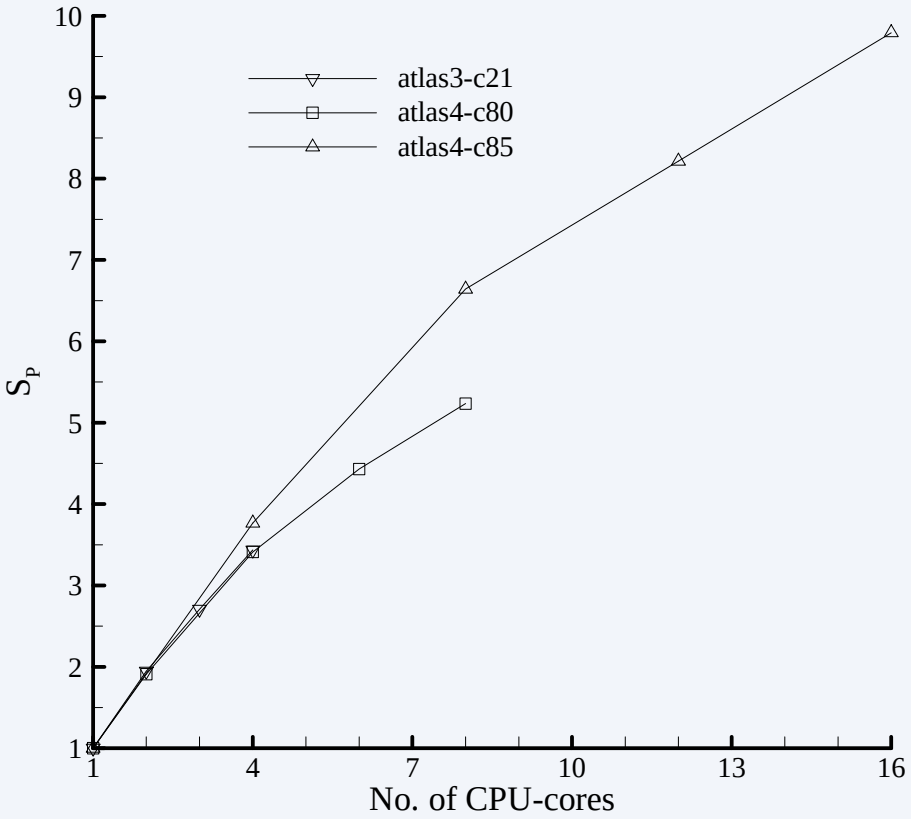
➢ Speedup & Efficiency

Table 1. Comparison of CPU time and wall clock time for different number of CPU-cores

atlas3-c21	CPU-cores	1	2	3	4	/
	CPU Time (s)	12469	12772	13718	14313	/
	Wall Clock Time (s)	12479	6415	4609	3633	/
atlas4-c80	CPU-cores	1	2	4	6	8
	CPU Time (s)	9882	10293	11418	13163	14835
	Wall Clock Time (s)	9894	5183	2900	2233	1890
atlas4-c85	CPU-cores	1	4	8	12	16
	CPU Time (s)	15842	16601	18775	22545	25508
	Wall Clock Time (s)	15855	4209	2387	1930	1619

Parallelization

➢ Speedup & Efficiency

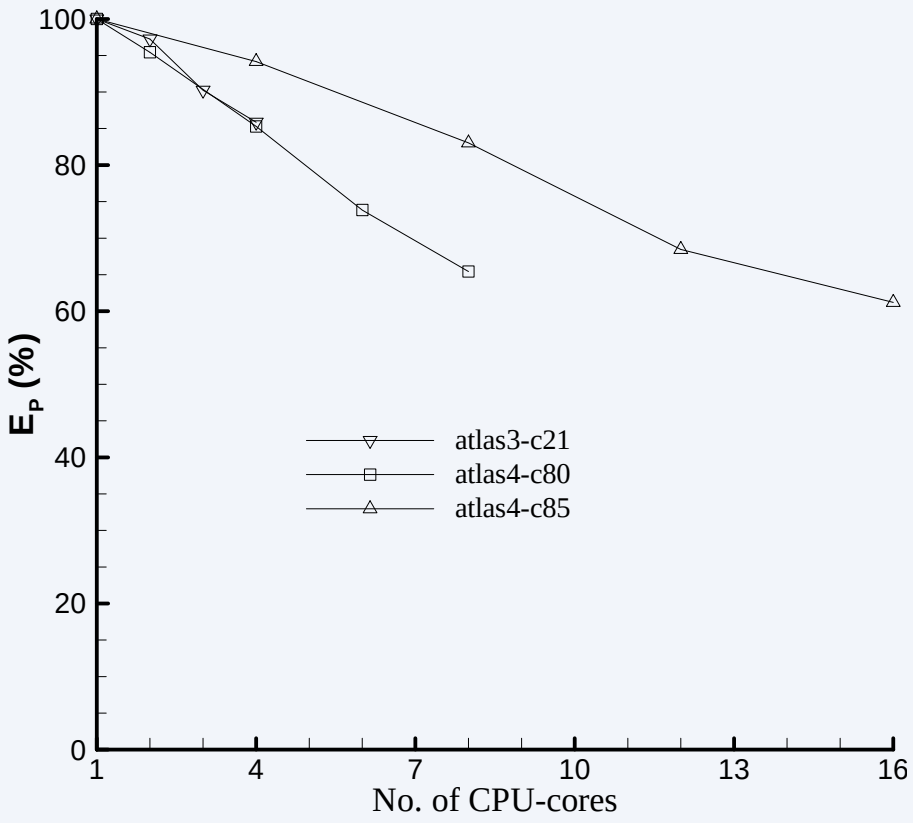


Speedup factor

1. The speedup factor rises more slowly as the number of CPU-cores *P* is increased
2. If the same number of CPU-cores is used, the speedup factor of atlas4-c85 is the biggest.

Parallelization

➢ Speedup & Efficiency

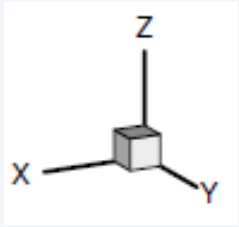


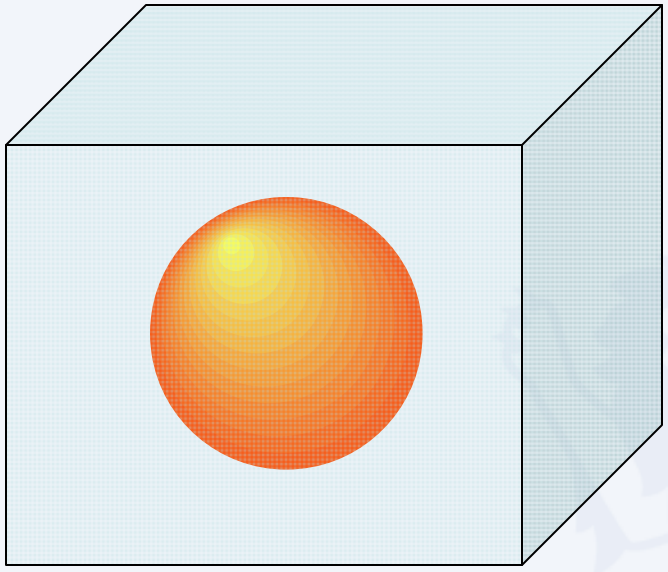
Efficiency

1. The parallel efficiency decreases with an increase in *P*.
2. The simulations on atlas4-c85 can achieve better efficiency

Numerical Examples

- **Rotating of a Sphere in a Box**
  - Computational model





$$R=L/4$$

$$\mathrm{Re}=\Omega R^2/\upsilon$$

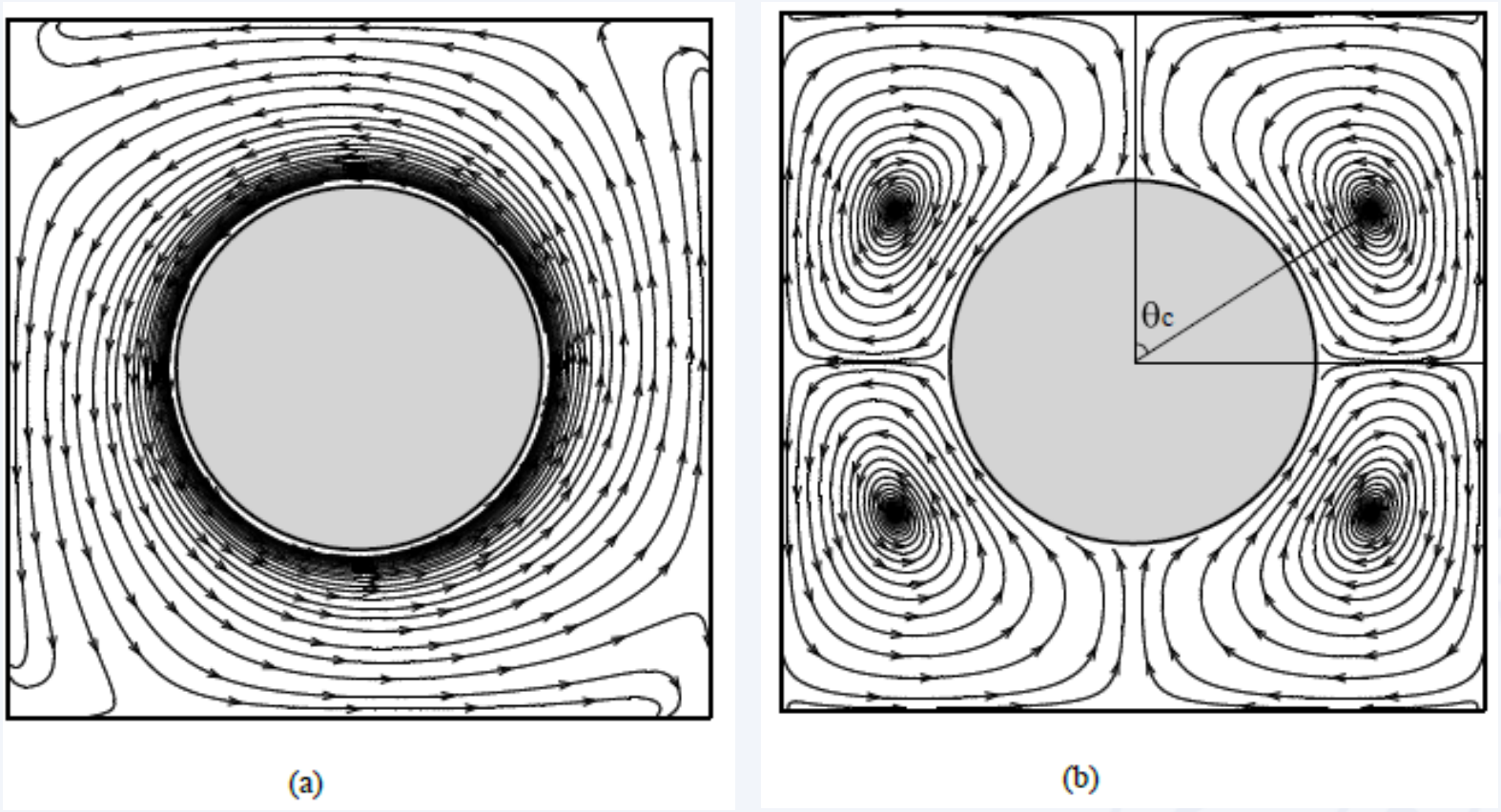
$$T=R\int_S \bar{e}_r\times\tau dS$$

$$C_T=-\frac{T}{0.5\rho R^5\Omega^2}$$

Sphere is located at the centre of the cubic box and rotated around z-axis

Numerical Examples

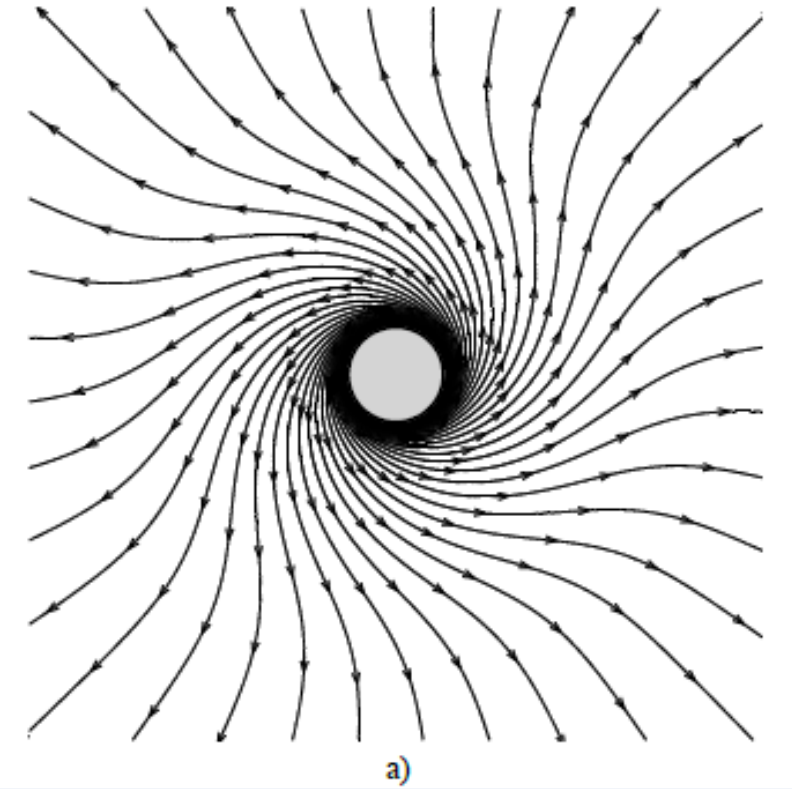
➢ Rotating of a Sphere in a Box

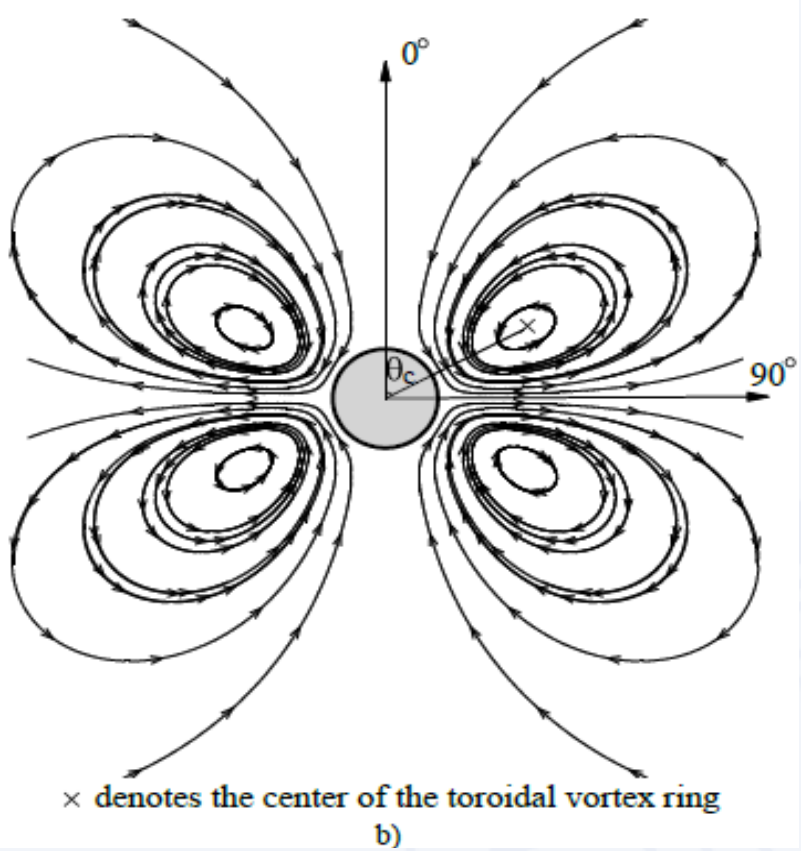


Streamtraces of steady-state flow fields around a rotating sphere in a cube at Re = 20, (a) *xy* central plane, (b) *yz* central plane.

Numerical Examples

➢ Rotating of a Sphere in an Unbounded Domain





Streamtraces of steady-state flow fields around a rotating sphere in an unbounded domain at Re = 20, (a) *xy* central plane, (b) *yz* central plane.

Numerical Examples

➢ Rotating of a Sphere in an Unbounded Domain

Comparison of the critical angle θ_c

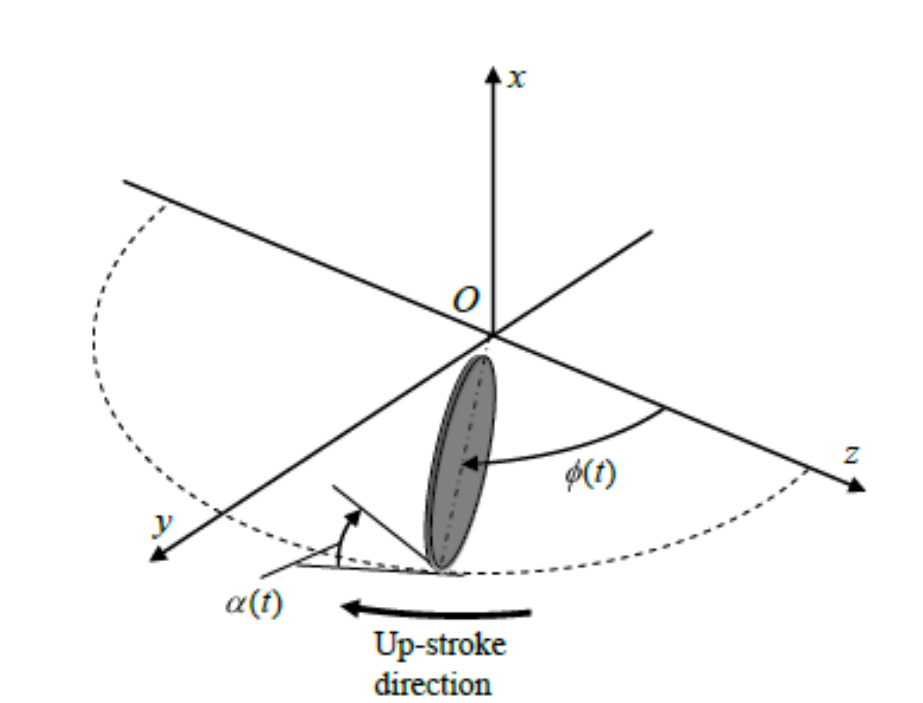
Re	Present Results	Ref. [45]	Ref. [19]	Ref. [46]
20	62.1	62.6	61.6	/
50	69.7	69.4	68.2	/
100	74.2	73.8	72.1	74.3

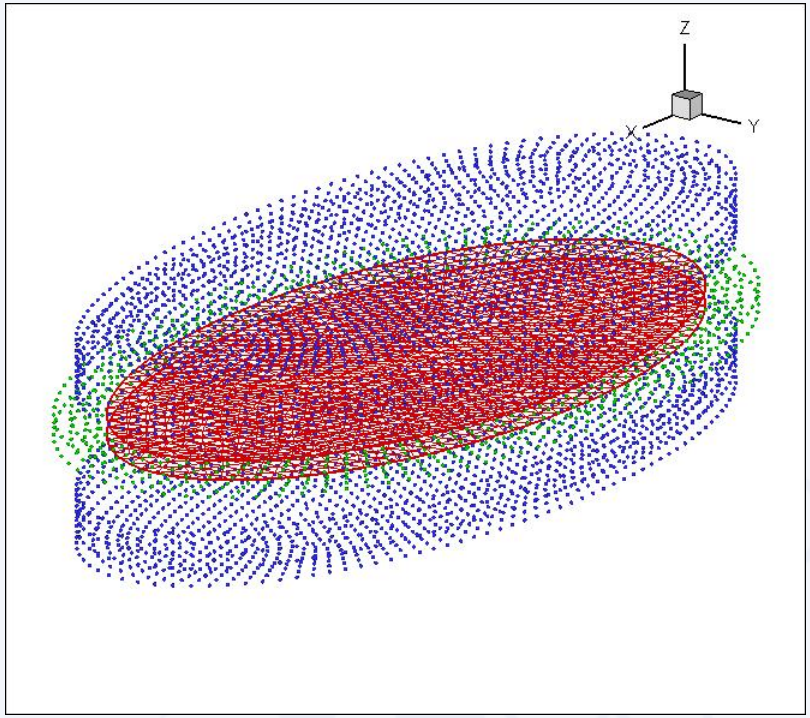
Comparison of the torque coefficient

Re	Present Results	Ref. [45]
10	5.312	5.399
20	2.990	3.048
50	1.535	1.544
100	0.956	0.966

Numerical Examples

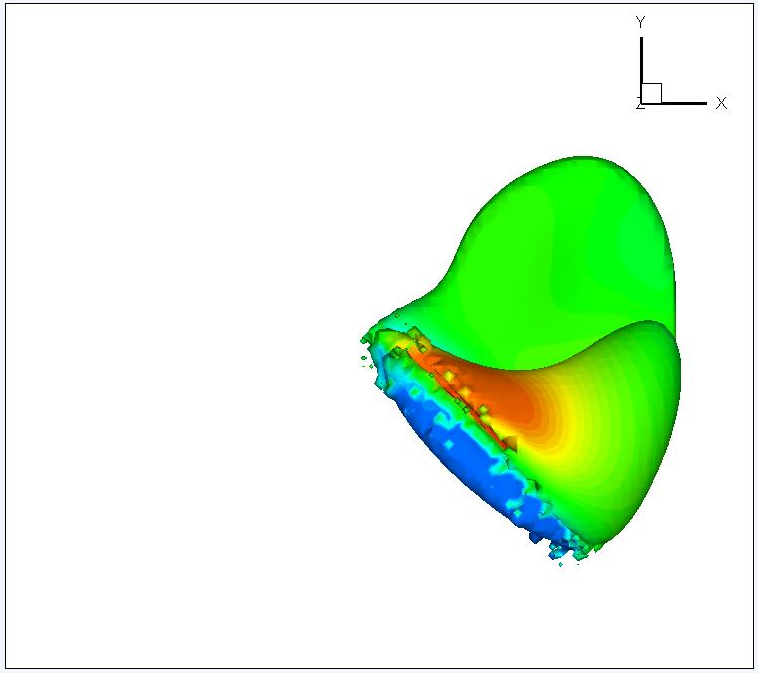
➢ Flapping Wing

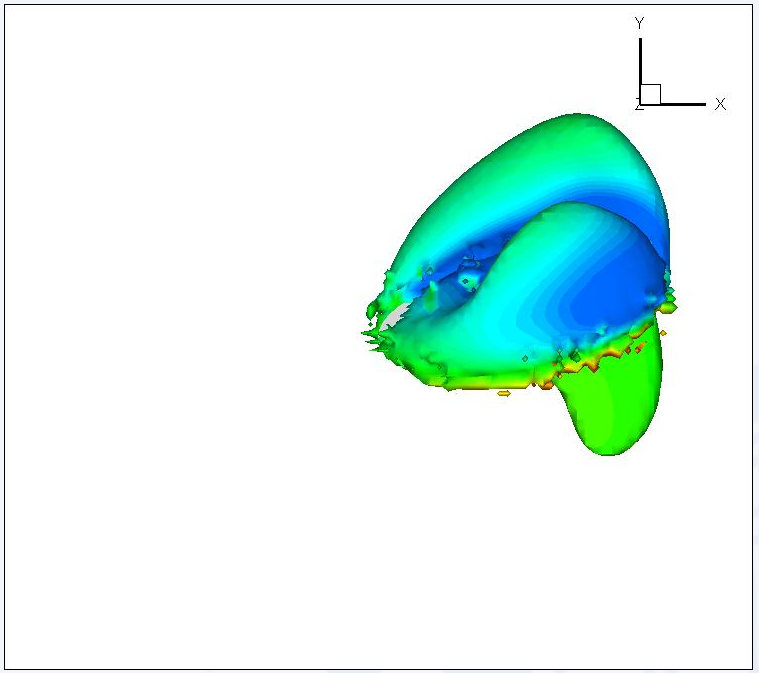




Numerical Examples

➢ Flapping Wing





Conclusion

- OpenMP technique is applied to speedup the 3D simulations of complex moving boundary problems
- The present code achieves speedup of around 10 for a computational node with 16 CPU-cores, which a corresponding parallel efficiency of 61.2%.

2 months work a CPU-core → 1 week work 16 CPU-cores

- When using 8 CPU-cores, the efficiency of atlas4-c85 is much higher than those of atlas4-c80 and atlas3-c21.
- The system configuration has a significant effect on parallel speedup and efficiency.

Acknowledgement

- The present study is supported by TSDI Grant TDSI / 07-001 / 1A

- Thanks to my supervisor Prof. Yeo K. S.

Any questions?

- Thanks to PhD students Zhang L.H., Zhao Yao

- Thanks to all the staff members of SVU Lab