ABSTRACT

KULKARNI, ANUPAM AJAY. Development of a Parallel Lagrangian Particle Tracking Code for 3D Multi-Block Curvilinear Grids. (Under the direction of Dr. Jack R Edwards.).

This work focuses on the development of a complete particle tracking code, capable of coupling with the existing CFD code REACTMB, developed at NCSU and used for a variety of flow simulations using both LES and RANS methods. A complete framework has been developed, capable of finding the location of particles in multi-block curvilinear grids, interpolating solution values to exact particle location and advecting the particles under the influence of drag and other forces. The code has been parallelized using OpenMPI, and the algorithm accounts for particle information exchanged between various blocks and processors to successfully track a large number of particles. The drag forces are calculated based on the solution data generated by the REACTMB code, at every time step. To account for turbulence, the code uses a Discrete Random Walk (DRW) model, allowing for stochastic particle tracking to account for turbulent effects. The code is highly modular and flexible, thereby allowing addition of modules to account for higher complexities, in terms of force models and stochastic particle tracking.

© Copyright 2015 Anupam Kulkarni

All Rights Reserved

Development of a Parallel Lagrangian Particle Tracking Code for 3D Multi-Block Curvilinear
Grids

by
Anupam Ajay Kulkarni

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Aerospace Engineering

Raleigh, North Carolina
2015

APPROVED BY:

Dr. Hong Luo

Dr. Tarek Echekki

Dr. Jack R. Edwards
Committee Chair

DEDICATION

To my parents and my late grandmother, for their unwavering faith in me.

BIOGRAPHY

The author was born on the 17th of December, 1989 in the village of Jalna, in the state of Maharashtra, India. He received his Bachelor's degree in Mechanical Engineering from the University of Pune in June 2011. The author went on to study Aerospace Engineering in North Carolina State University (NCSU), Raleigh, North Carolina, USA in Fall 2013. Here he began working in the Computational Fluid Dynamics laboratory from January 2014. His academic and research advisor is Dr. Jack Edwards.

ACKNOWLEDGMENTS

I would like to thank Dr. Jack Edwards, for his support and belief in my work, and especially for keeping me focused and being patient with my unending line of questions. Additionally I would like to thank my committee members, Dr. Hong Luo and Dr. Tarek Echehki, who have both taught and guided me through my time at NC State, along with their insightful inputs and support. Last but not the least, I would like to thank my parents, and family and friends for their support and keeping me on my feet.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES.....	viii
INTRODUCTION	1
PARTICLE TRACKING ALGORITHM.....	6
2.1 GRID CONNECTIVITY.....	6
2.2 PARTICLE INJECTION	9
2.3 PARTICLE STATE	9
2.3 THE ALGORITHM.....	10
PARTICLE SEARCH.....	13
3.1 GEOMETRICAL FORMULATION.....	13
3.2 PARTICLE SEARCH ALGORITHM	16
INTERPOLATION	18
INTEGRATION.....	22
5.1 GOVERNING EQUATIONS AND FORCE MODELS	22
5.2 NUMERICAL METHODS FOR INTEGRATION	23
5.3 STOCHASTIC PARTICLE TRACKING	25
PARALLELIZATION	27
6.1 IMPLEMENTATION	27
RESULTS	29
7.1 PARTICLE TRACKING ON A 2-D SINGLE BLOCK GRID.....	29
7.2 PARTICLE TRACKING ON A 3-D SINGLE BLOCK GRID.....	36
7.3 PARTICLE TRACKING ON A 3-D MULTI-BLOCK GRID	41
7.4 PARTICLE TRACKING COUPLED WITH REACTMB.....	44
CONCLUSIONS	53
8.1 SUMMARY OF COMPLETED WORK.....	53
8.2 FUTURE WORK.....	54

LIST OF TABLES

TABLE 1: SUM OF COEFFICIENTS FOR VARIOUS PARTICLE LOCATIONS	17
TABLE 2: PARTICLE DIAMETERS FOR EACH TRAJECTORY	35

LIST OF FIGURES

FIGURE 1: NEIGHBOR ZONES FOR A BLOCK	7
FIGURE 2: LINKED LISTS FOR CONNECTIVITY	8
FIGURE 3: PARTICLE SEARCH: BASIC IDEA.....	14
FIGURE 4: ORIENTATION OF VERTICES FOR A UNIT CELL	19
FIGURE 5: U VELOCITY IN 2-D CHANNEL.....	30
FIGURE 6: V VELOCITY COMPONENT IN 2-D CHANNEL	30
FIGURE 7: PARTICLE LOCATIONS AT T = 5 SEC.....	31
FIGURE 8: PARTICLE LOCATIONS AT T = 12.5 SEC.....	31
FIGURE 9: PARTICLE LOCATIONS AT T = 22.5 SEC.....	32
FIGURE 10: PARTICLE LOCATIONS AT T = 5 SEC, WITH GRAVITY	32
FIGURE 11: PARTICLE LOCATIONS AT T = 12.5 SEC, WITH GRAVITY	33
FIGURE 12: PARTICLE LOCATIONS AT T = 22.5 SEC, WITH GRAVITY	33
FIGURE 13: TRACKS AT T = 22.5 SEC FOR PARTICLES WITH DIFFERENT DIAMETERS	34
FIGURE 14: 3-D SINGLE BLOCK GRID	36
FIGURE 15: U AND V VELOCITY COMPONENTS	37
FIGURE 16: PARTICLE LOCATIONS AT T = 5E-5 SEC.....	38
FIGURE 17: PARTICLE LOCATIONS AT T = 2E-4 SEC.....	38
FIGURE 18: PARTICLE LOCATIONS AT T = 5E-4 SEC.....	39
FIGURE 19: PARTICLE LOCATIONS AT T = 0.01 SEC; ISOMETRIC VIEW.....	40
FIGURE 20: PARTICLE LOCATIONS AT T = 0.01 SEC; TOP VIEW.....	40
FIGURE 21: PARTICLE DISTRIBUTION AT T = 0.0 S.....	41
FIGURE 22: PARTICLE DISTRIBUTION AT T = 0.0025 S.....	42
FIGURE 23: PARTICLE DISTRIBUTION AT T = 0.005 S.....	42
FIGURE 24: PARTICLE DISTRIBUTION AT T = 0.01 S.....	43
FIGURE 25: THE GRID (786 BLOCKS)	44
FIGURE 26: U - VELOCITY PROFILE AT Z = 1E-04.....	45
FIGURE 27: V - VELOCITY PROFILE AT Z = 1E-04.....	45
FIGURE 28: U-VELOCITY PROFILE AT Z = 1E-04 (MAGNIFIED VIEW OF THE INJECTION SITE).....	46
FIGURE 29: V-VELOCITY PROFILE AT Z = 1E-04 (MAGNIFIED VIEW OF THE INJECTION SITE).....	46
FIGURE 30: DIFFERENTIAL PARTICLE TRAJECTORIES WITHOUT DRW.....	47
FIGURE 31: PARTICLE LOCATIONS AFTER 400 TIME-STEPS.....	48

FIGURE 34: PARTICLE LOCATIONS AFTER 800 TIME-STEPS.....	49
FIGURE 35: PARTICLE LOCATIONS AFTER 1200 TIME-STEPS	49
FIGURE 36: PARTICLE LOCATIONS AFTER 1600 TIME STEPS	50
FIGURE 37: COMPARISON OF COMPUTATION TIME.....	51
FIGURE 38: COMMUNICATION AND DATA OUTPUT TIMES PER ITERATION	52

Introduction

Particle tracking has become an essential tool used in conjunction with CFD codes to simulate real world flows. Many flows occurring in nature, from environmental flows to internal flows in oil and gas, are laden with particles. As a result, there is a need for a robust and efficient particle solver to track a large number of particles. Its also important that it work on a variety of meshes. Such a code was developed to directly couple with the CFD code REACTMB. Multiple particle tracking codes exist, as a part of open source and commercial codes as used in Fluent or OpenFOAM. Each package uses individual approaches to address particle tracking, with its advantages and disadvantages. Development of an indigenous code, however, gives the best possible coupling with an existing CFD code. It allows for optimization of each aspect of particle tracking to the need of the application. This method also provides the maximum possible flexibility and portability. Additionally, most of the developed codes view particle tracking as a post-processing step passed on a steady state flow field, however, the code developed works in tandem with the CFD code. This allows for transient particle tracking, although one-way coupled at this point. The coupling of the particle tracking engine with the main CFD code significantly affects certain aspects of the particle tracking algorithm including the way parallelization of the code takes place. This code as been developed as a modular framework, which can then allow modification of individual functions to capture additional complexities. It also allows the addition of more capabilities with additional functions.

Particle tracking consists of three basic steps:

- Particle search
- Interpolation
- Integration

There are multiple ways to perform each step for a variety of different grids. Additionally, with respect to curvilinear grids, there are two different paradigms to address these steps. Some codes employ a mapping from physical space (P-space) to computational space (C-space), where the 3-D curvilinear grid is mapped to a 3-D Cartesian grid. [5]. This makes the search operation much simpler and also allows for direct trilinear interpolation of flow variables. However, the particle trajectories calculated at each time-step must be remapped to the P-space. A comparison of such C-space codes with P-space codes [1,9] reveals that the performance of C-space algorithms is affected by distortions and curvature in the grid. Distortions in grids are common and necessary to fit properly into complex geometries. The resulting curvature affects the particle trajectories in such cases, giving incorrect results [9]. The accuracy obtained with such codes is also inferior, or at best, the same as the accuracy of P-space algorithms [1]. Even in terms of efficiency, due to the cost of mapping P-space to C-space and then re-mapping displacement from C-space to P-space, C-space algorithms can sometimes become more computationally expensive [9] in comparison to P-space algorithms. Dealing with P-space algorithms, however, complicates the particle search, although P-space algorithms tend to be less complex and more intuitive.

Particle search is the first step required in the tracking algorithm, and is required for each particle at every time-step. Particle search locates the cell containing the particle at a given time step. Usually, particle search in itself is an iterative process, thereby requiring an initial guess. A fast search algorithm is thus necessary while dealing with real world problems. Multiple ways to search particles in curvilinear grids exist [2-3]. The method proposed by P. J. Oliveira et al. [2] proposed an idea which involves mapping each hexahedral cell to a unit cube, using iso-parametric functions. However, this method had certain drawbacks. The mapping functions for each cell are usually unique, increasing computational costs drastically. The idea discussed by X. Q. Chen [3] has been used and modified for the purpose of the developed code. This idea employs a geometrical consideration of whether a particle lies within a general hexahedral cell. A coefficient corresponding to each face is generated, by comparing the particle location and the location of the cell center with respect to the face. Based on the calculation of coefficients for each face using simple algebraic calculations, the algorithm determines if the particle lies within the cell. If the particle is outside the cell, the algorithm determines which adjoining cell might best contain the particle. By continuing this iterative process, the particle location in the correct cell is determined. Since simple algebra is used to achieve this, computational cost is less. Additionally, it can be seen that a better initial guess for the search will also reduce search time

and improve efficiency. For this purpose, the data structures used are such that particle locations for the previous time instant are stored for each time step. Since displacements scale with cell width, the particle is found in the next time-step within very few iterations.

Since this code accounts for one-way coupling, solution data generated by the CFD code is used to determine the particle displacement. This data includes fluid velocity, density, viscosity and turbulent kinetic energy. However, this data is calculated at the cell centers of the grid. This data is then averaged to the grid nodes. The particle location may or may not coincide with the nodes of the grid. As a result, solution data must be interpolated to exact particle location. The solution data at the nodes of this cell is used for interpolation. The solution data generated by the REACTMB code is second order accurate, which implies a linear variation of quantities within the cell. As a result, a trilinear interpolation scheme has been used to determine solution values at the particle location. However, trilinear interpolation requires a Cartesian cell. For this purpose, the iso-parametric equations used for interpolation are first used to map the given cell to a unit cube. Similarly, the particle location in the grid must be mapped to the unit cube. Once the particle location in the unit cube is determined, trilinear interpolation is performed to determine flow quantities at the particle location. While higher order methods exist [4] and can be readily incorporated in the developed code, for the purposes of the conducted study, linear variation within a mesh cell is assumed.

Depending on the application, every particle tracking code must account for a variety of forces, either using models or an exact formulation. The paper presented by Mahdavianesh et. al provides the formulations many such forces [6]. The code developed currently only accounts for drag forces, which are the primary forces affecting the particle trajectory in most applications. Addition of terms for buoyancy and other constant forces, which do not depend on particle velocity can be done very easily, since these forces are constant and do not vary with particle velocity, but depend only on particle properties. These properties are already available and calculated in the existing code. Additionally, there are multiple ways to actually integrate the particle path, numerically using explicit or implicit schemes. While explicit schemes can work for smaller displacements, or smaller time-steps, in some cases they lead to an additional restriction on the time step. Higher time-steps can cause instabilities, resulting in particles being lost from the domain, giving erroneous results [7], especially in high speed flows. While use of higher order Runge Kutta methods can reduce this, these methods necessitate the storage of particle locations at previous time steps. This increases complexity and memory requirements.

Additionally, the number of particles in the simulation for each time step can be different, and thus the arrays that store displacements for previous time steps must be re-dimensioned. Use of an implicit formulation relaxes the restriction on a choice of time step, and the formulation is not as complex. The drag force experienced by the particle is a function of the relative velocity magnitude as well as direction. This results in a non-linear variation of the drag force with respect to the relative velocity. For this code, a linearized formulation has been used to facilitate an implicit treatment, which is illustrated in detail in Chapter 5.

Stochastic particle tracking to account for turbulent flow fields has also been addressed in the code, using a discrete random walk model. Stochastic particle tracking is one of the methods to account for particle dispersion in a turbulent flow field. A discrete random walk (DRW) model as discussed by B. Zhao et. al. [8] has been used in this code. The discrete random walk model uses a normally distributed random number and the turbulent kinetic energy at the particle location to determine an additional fluctuating component. The mean flow component is derived from the solution data and interpolated to the particle location. The instantaneous component of flow velocity at the particle location is thus derived using a combination of both the mean flow and fluctuating components. Many other discrete random walk models exist, with some accounting for multiple aspects of turbulent flow fields [6]. These consider additional complexities like eddy interaction times. However, a simplified discrete random walk model is used in this code, since the results presented by B. Zhao et. al., provide sufficient agreement with experimental data. Furthermore, the framework developed is such that more sophisticated and complex models to account for particle dispersion due to turbulence can be eventually accommodated. The purpose of incorporating a simplified DRW model is have a basic turbulence modeling capacity to account for particle dispersion. A detailed description of the formulations for the DRW model used can be found in Chapter 5.

A key feature of this code is its ability to run on a parallel architecture. CFD simulations for realistic meshes have significant computational costs, often making parallelizable codes necessary. A transient particle tracking code adds to the cost, especially when required to track millions of particles to provide enough statistical data. These particles are tracked through a few million grid cells, every time step. The use of the right type of parallelization is thus essential. As discussed by O. Byrde et.al. [7], there are multiple ways to consider parallelization of multiple particles. Since most codes use particle tracking as a post processing step, as discussed by O. Byrde et. al., the parallelization procedure and data mapping used for tracking particles can be

different from the one used for obtaining the flow solution. In such a case, if particles do not interact with each other, every processor can be allotted a certain number of the total particles and allowed to track them independently. However, this requires each processor to have the access to the entire domain, which results in excessive memory overheads and may not always be even possible. For the specific purpose of this code, the existing domain decomposition used by the CFD code is maintained. The particles tracked by a particular processor are those that exist in the part of the domain which has been mapped to that processor. As the particles cross the multiple blocks in the domain, relevant particle data is sent to the right processor using a predefined connectivity. The generation and nature of this connectivity is explained in detail in Chapter 2.

Particle Tracking Algorithm

The basic algorithm for particle tracking consists of the three primary steps considered in the last chapter: particle search, interpolation and integration. However, the presence of multiple processors makes the algorithm more complex. Similarly, there are certain auxiliary functions which must be performed, like injecting particles, as well as pre-processing steps like establishing grid connectivity. These nuances, along with the detailed algorithm, will be discussed in this chapter.

2.1 Grid Connectivity

Establishing grid connectivity is essential for a multi-block mesh. Once a particle has left the block it previously was present in, for the purpose of tracking it successfully and efficiently, its new block identification (id) must be known. In a multi-processor environment, the processor to which the particle data must be sent has to be determined. This is done as a pre-processing step to generate a connectivity file in a pre-defined format. The grid connectivity used by the REACTMB code specifically specifies the face neighbors of each block. While this information is sufficient for the CFD code, particles can cross blocks across edges and corners. In a multi-block grid, each grid will have 26 neighboring zones:

- 6 face neighbor zones
- 12 edge neighbor zones
- 8 corner neighbor zones

Particles in the block may cross the block boundary and go to any of these zones.

Figure 1: Neighbor Zones for a block

The above figure shows the face, edge and corner neighbors in red, blue and green colors respectively. The connectivity works on the idea that each zone can be mapped to a unique set of ghost cells. If the particle is found in a certain ghost cell, its zone can be determined. Based on the zone, the tracking code will refer to the connectivity file and determine what is to be done with the particle.

Each entry in the connectivity file has three numbers, with each delineating the following information about a given zone:

- Boundary Type (Inlet/Outlet, Wall, Interface, Symmetry, etc.)
- Processor ID
- Local Block ID

Figure 2: Linked Lists for Connectivity

As seen from Figure 2, the neighbor zones ids are consistent for all blocks. The first part of the linked list is a 27 element, 3-D array, which stores the neighbor zone ids for a block. A function uses this array to return a value between 1 and 26, based on the ghost cell id. The second part of the linked list is a 1-D array, which stores the pointers to the actual data file. Since each block has 26 neighbor zones, the 1-D array has a total number of elements which is a product of the

total number of blocks in the domain and 26. For a given zone, there is at least one block entry if the boundary type is an interface. However, there may be multiple blocks in the same neighbor zone. In this case, the 1-D array points to the first neighbor block in that zone.

2.2 Particle Injection

Particles are injected from the defined inlet faces at every time-step. The input file used for this purpose gives the total number of inlet faces along with the global ids and face ids of each block-face. Each processor will read this file and identify which of the blocks belong to it. For every time-step, each face injects a certain number of particles, which is calculated based on the total number of particles to be injected and the total number of inlet faces. The locations of injection, for each time-step can be randomized to give better statistical results.

In case of curvilinear grids, the inlet faces may be curved. As a result, generating a random coordinate for a particle, which lies on the face, is complex. For a given block, based on the face id, a random cell id is generated, such that the cell lies on the specified face. The cell coordinates (x, y, z) are then mapped to a unit cube (ϵ, ζ, η) , using the iso-parametric formulation for trilinear interpolation. The details of this interpolation are specified in Chapter 4. Once mapped to a unit cube, a function generates a random value between 0 and 1, for ϵ , ζ and η . Based on the face through which particles are to be injected, one of ϵ , ζ , η is set to 0. These values are then used to generate randomized x , y and z coordinates for the particle. The advantage of knowing the cell whose face has been used for injection for a corresponding particle is that the cell ids of that particle are already known for the time-step. Thus, the search time for newly injected particles is negligible.

2.3 Particle State

The ‘state’ of a particle is an important attribute that determines if the particle is to be tracked, or deleted from the list maintained on a processor. Particles that are being tracked for the current time step have a default state of 1. When a particle is either lost, or when it flows out of the block domain, the state is set to 0. Such particles and their corresponding data is eventually deleted from the list maintained on the processor. In some cases, the particle state is set to various pre-defined numbers which can help to gather specific data; for example, the number of particles

hitting the wall. This flag also helps track when multiple entries of the same particle have to be used in the case of multiple neighbor blocks in a single zone.

2.3 The Algorithm

The particle tracking code works in the same time loop as the main CFD code. The solution data generated by the CFD code is then accessed by the particle tracking code and used for integration. Note that the algorithm specified next runs simultaneously on all processors.

```
do t=1,tmax  
  Generate solution data  
  Inject particles  
    do np = 1, no. of particles  
      Search particle  
      if particle state is 0,  
        skip the rest and move to next particle  
      Use trilinear interpolation to get solution data at particle location  
      Calculate particle displacement  
      if particle is close to boundary  
        Calculate new particle position  
        Search particle again  
        if particle is in ghost cell  
          Read connectivity file to determine boundary type  


- For walls, calculate intersection point with the wall
- For outlets, tag particle state as 0
- For interfaces, determine new processor and block id

  
          if processor is same as current then  
            Update local block id  
          else  
            Tag particle state as zero  
            Pull out all blocks to which particle is to be sent  
            Update counters for each processor to which particle is to be sent
```



```

        Update array which holds sending information
    Recalculate original particle position
    Calculate new particle position
end
Calculate how many particles are to be sent to each processor
Compile lists of particles to be sent to each processor
synchronize processes
Transfer data
Reallocate memory to accommodate sent, received and lost particles
synchronize processes
end

```

At the start of the main time loop, the solution data is first calculated by the CFD code. The solution data is then used for particle tracking. A pre-defined number of particles is injected into the domain from randomized locations on the inlet face. During injection, particles are given a specified inlet velocity and diameter. Similarly, other attributes such as the cell containing the particle, the local block id, the processor id, and the state flag are also set. These new particles are added to the list of existing particles on each processor. The location of each particle is then determined for the next time step in a loop over the particles, handling one particle at a time.

In the loop, each particle is first searched in the local block that contains it. For this purpose, the cell containing the particle at the previous time step is used as an initial guess. Once the cell containing the particle is determined, this cell id corresponding to the particle is updated. This search is performed on the real domain of the block, not including the ghost cells. This is done to handle the case when multiple blocks may be present in a given neighbor zone. In such a case, the same particle data is sent to each block, however, it will be found in only one block. Particles which are not searched successfully are tagged by setting their state to 0 and the loop cycled to the next particle. The solution data is then interpolated to the particle location and stored in a separate array. Based on the interpolated values, the displacement for the particle is calculated.

Particles that are close (within two cell widths) to a boundary are re-checked, to ensure that they are still present in the domain at the next time step. For this purpose, the new particle location is

determined from the calculated displacement, and the particles are searched again. In this case, the search is conducted to allow for the particles to be found in the ghost cells of the block. If the particle lies in the ghost cell, its neighbor zone id is determined. Based on this, the grid connectivity information is referenced to determine the boundary type. If the boundary is a wall, the intersection of the particle with the boundary surface is determined. In case of an outlet, the particle state is tagged to zero, so that its final position will be saved and the particle will no longer be tracked for the further time steps. In case of an interface, the list of blocks to which the particle data is to be sent is pulled from the connectivity information. The particle state is tagged to two, which is used to determine that the particle data must be sent to the relevant processor. If the new block lies on the same processor, the local block id is simply updated. There are two arrays, which hold information about particle transfers; these arrays are suitably updated.

Finally, once the location and other attributes for all particles have been determined, the send buffer is filled with relevant data. At this point, all processes are synchronized. This is done so that all the processes can send and receive data simultaneously and efficiently. As discussed in depth in Chapter 6, data transfer is done using a non-blocking send and a blocking receive function call. In some cases, it is possible that some processors have too few or no particles at all, and in such cases, these processes may start sending garbage data. To ensure this doesn't occur, a synchronization step is done to then begin data transfer. After particle data has been transferred between each processor, the three arrays that hold all particle data are re-dimensioned to accommodate the new number of particles. Particles that have hit walls, or outlets or have been transferred are no longer tracked and hence deleted. Similarly new particle data that has been received is added to the existing arrays. It is important to note that the re-dimensioning has to be done all three arrays, to ensure that the ' i^{th} ' particle data referenced from each array corresponds to the same particle. After reallocation of arrays, the processors are synchronized again, so that time accuracy is maintained.

The individual functions of particle search, interpolation and integration, along with data transfer are discussed in further detail in the following chapters.

Particle Search

Particle search is required to be done for every particle for every time-step, and thus having the fastest possible search algorithm is essential. It is important to note that a particle is said to lie within a cell even if it lies on one of the faces, edges or coincides with one of the corners. The only purpose of particle search is to allow interpolation of flow variables to the exact particle location. The condition to check if the particle is located in a given cell is divided into six sub-checks, one for each face. Particle search is an iterative process that begins from an initial ‘guess’ cell value. The algorithm is such that it determines if the particle lies within this guessed cell. If not, it determines which adjoining cell might contain the particle. This way, the algorithm proceeds from the initial guessed cell to sequentially move towards the cell which actually contains the particle. Naturally, the speed of the particle search depends on the initial guess. To speed up this process, the cell number that contained the particle for the initial time step is stored and used as the initial guess for the particle search operation at the next time step. It should be noted that a particle is searched within a specific block. The local identification number of a block must therefore be stored. This search algorithm thus assumes that the local block identification number of the particle being searched is known.

3.1 Geometrical formulation

For each plane, if the particle lies on the same side as the cell center, the particle must lie within the cell, since the cell center of a general convex hexahedral cell must always be in the interior of the cell. Consider a general hexahedral cell as shown in Figure 3. Let us consider the East face of this cell, formed by points P_1 , P_2 and P_3 . Let the cell center of the cell be denoted by P_0 and the particle location by X_p . Let $\vec{r}_p$ and $\vec{r}_0$ be vectors joining points P_2 and X_p and points P_2 and P_0

respectively. Let the normal vector to the east face be represented by $\vec{n}$. For a particle to lie on the same side of the East face as the cell center, the dot products of $\vec{r}_p$ and $\vec{r}_0$ with $\vec{n}$ each must not have opposite signs.

Let,

$$\vec{n} = \overrightarrow{P_2P_1} \times \overrightarrow{P_2P_3}$$

$$f_1 = \vec{r}_0 \cdot \vec{n}; \quad f_2 = \vec{r}_p \cdot \vec{n}$$

$$i = \text{sgn}(f_1 \cdot f_2)$$

The sign function sgn will return a value of 1, if the product is positive; -1, if it is negative and 0 if the product is zero. Note that f_1 can never zero, but f_2 may be so, in the case the particle lies on the face. Note that this works with both the outward or the inward facing normal for a face.

Figure 3: Particle Search: Basic Idea

Such coefficients (i) will be generated for each face of the hexahedron. Let us represent each coefficient by its corresponding face, to get i_e , i_w , i_n , i_s , i_t and i_b for the east, west, north south, top and bottom faces respectively. The sum of these coefficients and their exact value will be used to determine if a particle lies within a cell, and if not, give an update for the next guess. When the particle is located on the same side of the plane as the cell center of the cell, it returns a value of 1, as seen from the calculation of i . Similarly, if the particle lies on the plane, the coefficient value will be 0, and if it lies on the opposite side as the cell center, the value will be -1.

For a particle in the interior of the cell, the sum of the coefficients generated for each face will be 6, since for each face, the particle and the cell center lie in the interior. Similarly, if the particle lies on the face of a cell, the sum will be 5, since the coefficient for the face the particle is on will be 0, while the rest of the faces will generate a coefficient of 1. However, this condition of sum of coefficients being greater than or equal to 5 is not sufficient to handle the cases where a particle may lie on the edge or the corners of a given block. For a particle on the corner of cell which is the intersection of 3 faces, the sum of coefficients will be 3. Similarly, for a particle on the edge of the cell, which is an intersection of two faces, will be 4. However, particles lying entirely outside the cell also return values of 3 and 4. As a result, an additional condition is set to check if the particles lie inside the cell. For this, we consider the difference between the values of the opposite faces.

$$del_i = 0.5(i_w - i_e)$$

$$del_j = 0.5(i_s - i_n)$$

$$del_k = 0.5(i_b - i_t)$$

The new condition states that a particle is located inside the cell if the values of del_i , del_j and del_k are all zero. This holds true for points in the interior and those on the faces, edges or corners of the cell.

Once a particle is determined to be outside the guessed cell value, del_i , del_j and del_k also act as updates to calculate the cell identification numbers (i , j , k) for the new cell. This is explained in further detail in the particle search algorithm.

3.2 Particle Search Algorithm

The particle search function begins its search from an initial guess value. From this cell, using the del_i , del_j and del_k values generated, a new adjoining cell that may contain the particle is determined. This iterative process is continued until the cell containing the particle is found. The following algorithm explains the exact working of the basic idea discussed in the previous section.

```
Initialize ntry = 0
Set initial cell to ( $i_g, j_g, k_g$ )
while ( $i_n + i_s + i_e + i_w + i_t + i_b$ ) < 5
    ntry = ntry+1
    Calculate values of  $i_e, i_w, i_n, i_s, i_t$  and  $i_b$ 
    Calculate increments  $del_i, del_j, del_k$ 
    if  $del_i=0$  &  $del_j=0$  &  $del_k=0$  exit loop
    Calculate updated guess values
    if new cell is out of bounds then revert to original value
    if ntry > ( $\Sigma$  difference of bounds + tolerance) then exit loop
if ntry > ( $\Sigma$  difference of bounds + tolerance) then tag the particle state to 0
```

The particle search is an iterative process, which starts from an initial guess for a cell which may contain the particle, say (i_g, j_g, k_g) . There is a counter variable, $ntry$, which counts the number of iterations required to search a specific particle. Alternatively it also provides an escape to the while loop, to ensure that the code does not get stuck in an infinite loop. As described in the previous section, for the current cell, the coefficients for each face are first calculated. These are then used to calculate del_i , del_j and del_k . If all three are zero, the loop exits. Note that each of the ‘del’ variables are integers, so they only values they will take are 1, -1 and 0.

Consider a scenario as shown in Figure 3. Let the cell id of the cell shown be (i_g, j_g, k_g) . For this cell, i_e would be -1, i_w would be 1, and the rest of the coefficients would be 0. As a result, del_i would be 1. Thus, the updated guess for the next iteration would be (i_{g+1}, j_g, k_g) . As shown in Figure 3, that is the cell which contains the particle. If the i, j, k axes are oriented differently, say in the exact opposite sense, the del_i would be -1, and the updated guess would be (i_{g-1}, j_g, k_g) . Thus, this algorithm works independent of the orientation of the blocks, which is essential, since

blocks in the grid may be oriented in various directions. The following table gives different scenarios for a particle location with respect to the cell and the sum of coefficients in each case.

Table 1: Sum of coefficients for various particle locations

Location of particle	Sum of coefficients	del_i	del_j	del_k
Inside the cell	6	0	0	0
On one of the faces	5	0	0	0
On one of the edges	4	0	0	0
On one of the corners	3	0	0	0
Outside, across a face	4	0/1/-1	0/1/-1	0/1/-1
Outside, across an edge	2	0/1/-1	0/1/-1	0/1/-1
Outside, across a corner	0	0/1/-1	0/1/-1	0/1/-1

As seen from the table above, for a particle to lie within a given cell, the condition where all the ‘del’ variables are zero is necessary. Figure 4 shows a schematic of how the particle search will proceed in a Cartesian mesh. In case of curvilinear meshes, the path taken by the search algorithm is slightly longer. In general, the iterations follow an almost linear path, from the guessed cell to the cell which actually holds the particle. This makes this search algorithm extremely fast.

Interpolation

Once the cell containing the particle is successfully located, the solution data corresponding to this cell must be interpolated to the exact particle location. The flow data usually includes values of velocity, fluid density, fluid viscosity and turbulent kinetic energy necessary for calculation of particle displacement. However, trilinear interpolation can only be used for cubic cells. As a result, each hexahedral cell must first be mapped to a unit cube. This is done using the same isoparametric form to the coordinates of the hexahedron as used for interpolation, which is as follows.

$$f = a_{f1} + a_{f2} \cdot \epsilon + a_{f3} \cdot \zeta + a_{f4} \cdot \eta + a_{f5} \cdot \epsilon \cdot \zeta + a_{f6} \cdot \zeta \cdot \eta + a_{f7} \cdot \epsilon \cdot \eta + a_{f8} \cdot \epsilon \cdot \zeta \cdot \eta$$

Each of the coefficients a_{fi} depends on the value f of at each of the 8 vertices of the hexahedron. Note that every such f (for example, x-velocity, y-velocity, z- velocity, density, etc.) will have its own set of coefficients. For simplicity we map the hexahedral cell to a unit cube. Thus,

$$0 \leq \epsilon, \zeta, \eta \leq 1$$

Each vertex of the unit cube is named from 1-8. The coefficients a_{fi} can be easily determined from the values of f , which are known at known, thus giving us 8 equations and 8 unknowns. Let us consider coefficients a_i , b_i and c_i corresponding to the f for the x, y and z coordinates of the particle. For a_i ,

$$f_1 = a_1$$

$$f_2 = a_1 + a_2$$

$$f_3 = a_1 + a_2 + a_3 + a_5$$

$$f_4 = a_1 + a_3$$

$$f_5 = a_1 + a_4$$

$$f_6 = a_1 + a_2 + a_4 + a_6$$

$$f_7 = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8$$

$$f_8 = a_1 + a_3 + a_4 + a_7$$

Figure 4: Orientation of Vertices for a Unit Cell

Thus,

$$a_1 = f_1$$

$$a_2 = -f_1 + f_2$$

$$a_3 = -f_1 + f_4$$

$$a_4 = -f_1 + f_5$$

$$a_5 = f_1 - f_2 + f_3 - f_4$$

$$a_6 = f_1 - f_2 + f_6 - f_5$$

$$a_7 = f_1 - f_4 + f_8 - f_5$$

$$a_8 = -f_1 + f_2 - f_3 + f_4 + f_5 - f_6 + f_7 - f_8$$

Similarly, we can get coefficients b_i and c_i corresponding to the y and z coordinates of the particle. Let the particle location be given by $\vec{X} = (x, y, z)$. Let its corresponding location in the unit cell be given by $\vec{\epsilon} = (\epsilon, \zeta, \eta)$. Thus, we can express x , y and z as follows:

$$x = a_1 + a_2 \cdot \epsilon + a_3 \cdot \zeta + a_4 \cdot \eta + a_5 \cdot \epsilon \cdot \zeta + a_6 \cdot \zeta \cdot \eta + a_7 \cdot \epsilon \cdot \eta + a_8 \cdot \epsilon \cdot \zeta \cdot \eta$$

$$y = b_1 + b_2 \cdot \epsilon + b_3 \cdot \zeta + b_4 \cdot \eta + b_5 \cdot \epsilon \cdot \zeta + b_6 \cdot \zeta \cdot \eta + b_7 \cdot \epsilon \cdot \eta + b_8 \cdot \epsilon \cdot \zeta \cdot \eta$$

$$z = c_1 + c_2 \cdot \epsilon + c_3 \cdot \zeta + c_4 \cdot \eta + c_5 \cdot \epsilon \cdot \zeta + c_6 \cdot \zeta \cdot \eta + c_7 \cdot \epsilon \cdot \eta + c_8 \cdot \epsilon \cdot \zeta \cdot \eta$$

However, $\vec{\epsilon}$ is not known, and is determined iteratively. We have a linear system given by $G(\vec{\epsilon}) = \vec{X}$. Let

$$F(\vec{\epsilon}, \vec{X}) = \vec{G}(\vec{\epsilon}) - \vec{X} = 0$$

Using Newton's Method to solve this system,

$$\vec{F}^{n+1} = 0 = \vec{F}^n + \vec{F}_{\vec{\epsilon}}^n (\vec{\epsilon}^{n+1} - \vec{\epsilon}^n)$$

$$\therefore \vec{\epsilon}^{n+1} = \vec{\epsilon}^n - \vec{F}^n \cdot (\vec{F}_{\vec{\epsilon}}^n)^{-1}$$

$$\text{where, } \vec{F}_{\vec{\epsilon}}^n = \frac{\partial \vec{F}^n}{\partial \vec{\epsilon}^n} = \left(\frac{\partial F_i}{\partial \epsilon_j} \right)^n = M_{ij}$$

M_{ij} will thus be as follows.

$$M = \begin{bmatrix} a_2 + a_5\zeta + a_7\eta + a_8\zeta\eta & a_3 + a_5\epsilon + a_6\eta + a_8\epsilon\eta & a_4 + a_6\zeta + a_7\epsilon + a_8\epsilon\zeta \\ b_2 + b_5\zeta + b_7\eta + b_8\zeta\eta & b_3 + b_5\epsilon + b_6\eta + b_8\epsilon\eta & b_4 + b_6\zeta + b_7\epsilon + b_8\epsilon\zeta \\ c_2 + c_5\zeta + c_7\eta + c_8\zeta\eta & c_3 + c_5\epsilon + c_6\eta + c_8\epsilon\eta & c_4 + c_6\zeta + c_7\epsilon + c_8\epsilon\zeta \end{bmatrix}$$

M is the Jacobian of the isoparametric transformation. Hence M^{-1} must exist since the mapping is one-to-one. Since M is a 3x3 matrix, its inverse can be calculated directly.

$$M^{-1} = \frac{1}{|M|} \begin{bmatrix} M_{22}M_{33} - M_{23}M_{32} & -(M_{12}M_{33} - M_{13}M_{32}) & M_{12}M_{23} - M_{13}M_{22} \\ -(M_{21}M_{33} - M_{23}M_{31}) & M_{11}M_{33} - M_{13}M_{31} & -(M_{11}M_{23} - M_{13}M_{21}) \\ M_{21}M_{32} - M_{22}M_{31} & -(M_{11}M_{32} - M_{12}M_{31}) & M_{11}M_{22} - M_{12}M_{21} \end{bmatrix}$$

$$|M| = -(M_{11}M_{22}M_{33} + M_{12}M_{23}M_{31} + M_{13}M_{21}M_{32}) + (M_{13}M_{22}M_{31} + M_{12}M_{21}M_{33} + M_{11}M_{23}M_{32})$$

The function $F(\vec{\epsilon}, \vec{X}) = (F_1, F_2, F_3)$ is as follows:

$$F_1 = a_1 - x + a_2.\epsilon + a_3.\zeta + a_4.\eta + a_5.\epsilon.\zeta + a_6.\zeta.\eta + a_7.\epsilon.\eta + a_8.\epsilon.\zeta.\eta$$

$$F_2 = b_1 - y + b_2.\epsilon + b_3.\zeta + b_4.\eta + b_5.\epsilon.\zeta + b_6.\zeta.\eta + b_7.\epsilon.\eta + b_8.\epsilon.\zeta.\eta$$

$$F_3 = c_1 - z + c_2.\epsilon + c_3.\zeta + c_4.\eta + c_5.\epsilon.\zeta + c_6.\zeta.\eta + c_7.\epsilon.\eta + c_8.\epsilon.\zeta.\eta$$

An initial guess of $\vec{\epsilon} = (0.5, 0.5, 0.5)$ is used, and the iterative process is carried out till the normalized error drops below a tolerance of 1E-08.

This gives the value of $\vec{\epsilon} = (\epsilon, \zeta, \eta)$ for a given particle. Once that is known, flow quantities at the grid nodes can be interpolated to the particle location. Note that, for each flow variable, a unique set of coefficients, a_i , have to be calculated.

Integration

The interpolated flow variables including flow velocity, fluid density and viscosity, along with turbulent kinetic energy, are used to calculate the displacement of the particle. A discrete random walk based approach is used for stochastic particle tracking, to account for turbulence effects.

5.1 Governing Equations and Force models

The governing equations for the motion of a particle can be given by

$$m_p \frac{d\vec{v}_p}{dt} = \vec{F}_{net}$$

$$\therefore \frac{d\vec{v}_p}{dt} = \frac{\vec{F}_{net}}{m_p} = \vec{f}_{net}$$

$$\frac{d\vec{x}_p}{dt} = \vec{v}_p$$

For the purpose of the results presented, drag forces and gravity are assumed to act on the particle. The drag force acting on a particle is given by

$$\vec{F}_d = \frac{3\rho_f m_p C_d |\vec{v}_r| \cdot (\vec{v}_r)}{4\rho_p \cdot d_p}$$

where C_d denotes the drag coefficient, and v_r represents the relative velocity of the fluid with respect to the particle.

$$\vec{v}_r = \vec{v}_f - \vec{v}_p$$

$$C_d = \frac{24}{Re_p} (1 + 0.15 Re_p^{0.687})$$

$$Re_p = \frac{\rho_f D_p |\vec{v}_r|}{\mu_f}$$

The particle is assumed to be perfectly spherical. Thus,

$$m_p = \frac{\pi \rho_p D_p^3}{6}$$

The ‘p’ subscripts denote particle properties, and ‘f’ subscripts denote interpolated fluid properties.

5.2 Numerical Methods for Integration

The drag force term can be linearized and written as

$$\frac{d\vec{v}_p^{n+1}}{dt} = \vec{f}_d(\vec{v}_p, \vec{v}_p^{n+1})$$

$$\frac{\vec{v}_p^{n+1} - \vec{v}_p^n}{\Delta t} = \vec{f}_d(\vec{v}_p, \vec{v}_p^{n+1})$$

Using a Taylor series expansion,

$$\vec{f}_d(\vec{v}_p, \vec{v}_p^{n+1}) = \vec{f}_d(\vec{v}_p, \vec{v}_p^n) + \frac{\partial \vec{f}_d}{\partial \vec{v}_p} (\vec{v}_p^{n+1} - \vec{v}_p^n)$$

$$\frac{\vec{v}_p^{n+1} - \vec{v}_p^n}{\Delta t} = \vec{f}_d(\vec{v}_p, \vec{v}_p^n) + \frac{\partial \vec{f}_d}{\partial \vec{v}_p} (\vec{v}_p^{n+1} - \vec{v}_p^n)$$

$$\left(\frac{1}{\Delta t} - \frac{\partial \vec{f}_d}{\partial \vec{v}_p}\right) (\vec{v}_p^{n+1} - \vec{v}_p^n) = \vec{f}_d(\vec{v}_p, \vec{v}_p^n)$$

$$\vec{v}_p^{n+1} = \vec{v}_p^n + \vec{f}_d(\vec{v}_p, \vec{v}_p^n) \cdot \left(\frac{1}{\Delta t} - \frac{\partial \vec{f}_d}{\partial \vec{v}_p}\right)^{-1}$$

From the drag force formulations we can determine that

$$\frac{\partial \vec{f}_d}{\partial \vec{v}_p} = -\frac{3\rho_f C_d |(\vec{v}_r)|}{4\rho_p \cdot d_p}$$

Gravity can be incorporated as follows:

$$\left(\frac{1}{\Delta t} - \frac{\partial \vec{f}_d}{\partial \vec{v}_p}\right) (\vec{v}_p^{n+1} - \vec{v}_p^n) = \vec{f}_d(\vec{v}_p, \vec{v}_p^n) + \vec{f}_g$$

$$\vec{v}_p^{n+1} = \vec{v}_p^n + [\vec{f}_d(\vec{v}_p, \vec{v}_p^n) + \vec{f}_g] \left(\frac{1}{\Delta t} - \frac{\partial \vec{f}_d}{\partial \vec{v}_p}\right)^{-1}$$

Thus, using linearization, the first equation of motion can be solved semi-implicitly. This allows for a wider range of time steps to be used for integration of the particle paths and is especially essential in higher speed flows.

The displacement of the particle is given by

$$\begin{aligned} \frac{d\vec{x}_p}{dt} &= \vec{v}_p \\ \frac{\vec{x}_p^{n+1} - \vec{x}_p^n}{\Delta t} &= \vec{v}_p^{n+1} \end{aligned}$$

Using a Taylor series expansion,

$$\vec{v}_p^{n+1} = \vec{v}_p^n + \frac{d\vec{v}_p^n}{dt} \Delta t$$

Combining both equations, we get

$$\overrightarrow{x_p^{n+1}} = \overrightarrow{x_p^n} + \overrightarrow{v_p^n} \cdot \Delta t + 0.5(\overrightarrow{a^n} \cdot \Delta t^2)$$

where,

$$\overrightarrow{a^n} = \frac{\overrightarrow{v_p^{n+1}} - \overrightarrow{v_p^n}}{\Delta t}$$

The above equations are solved for each particle for every time-step to thus calculate its displacement for that time-step.

5.3 Stochastic Particle Tracking

There are multiple ways to account for turbulence in flow fields and its effects on particle tracking. In this work, the dispersion of particles in the turbulent flow field is accounted by using a stochastic approach. A Discrete Random Walk approach is utilized, where instantaneous velocity fluctuations due to turbulence at the particle location are determined. The fluid velocity with which the particle interacts can be considered to be a sum of two components. One component is the mean flow velocity, interpolated at the particle location from the mean flow field solved by the REACTMB code. The second component is the fluctuating component, which is determined from the turbulent kinetic energy (TKE) and a normally distributed random number. Based on the model used by B. Zhao et. al. [8], the DRW model assumes that the fluctuation velocities follow a Gaussian distribution.

$$\overrightarrow{v'} = \vec{\zeta} \cdot \sqrt{2k/3}$$

Here, $\vec{\zeta}$ is a vector of three normally distributed random numbers, k is the interpolated TKE at the particle location, and $\overrightarrow{v'}$ is the fluctuating fluid velocity component. Each component of $\vec{\zeta}$ is generated randomly from a Gaussian distribution. A separate $\vec{\zeta}$ is generated for each particle, for every time step, for better statistical results.

Thus, for all the calculations mentioned in the current section,

$$\overrightarrow{v_f} = \overrightarrow{v_{fI}} + \overrightarrow{v'}$$

where $\overrightarrow{v_{fI}}$ is the interpolated mean fluid velocity.

Parallelization

The particle tracking code, along with the REACTMB code, is parallelized using OpenMPI. It is parallelized to run on distributed memory architecture, thereby making it highly scalable. A distributed memory architecture is a multiple processor system where each processor has its own memory. Computational tasks can operate only on local data, and remote data can be accessed using MPI (message passing interface) functions. The physical domain is decomposed into multiple parts and each processor can only access that part of the domain. In a particle tracking sense, a particular processor can see, and thus track, only a certain number of particles from the total set of particles that are being tracked. The domain decomposition used by the REACTMB code is also used by the particle tracking code. Thus, the same processor tracks particles contained in the blocks mapped to a certain processor. This is necessary as particle search requires the data of the entire block in which the particle is contained. At every time step, some particles may be advected out of any given block. Similarly, new particles enter the block from adjoining blocks. It is necessary to successfully track this exchange of information in case of multi-block, multi-processor environments, so that no particles are lost.

6.1 Implementation

Every processor maintains its own list of particles which may change at each iteration. Particles may encounter walls, or flow out of the domain or may be transferred from one block to the other after crossing block interfaces. In cases where particles cross an interface, if the new block to which the particle is advected lies on a different processor, all data relevant to that particle must be transferred to the new processor. Similarly, the particle data must be deleted from the old processor.

At each time step, when a particle is located in the ghost cell of a given block, the connectivity information determines what boundary type has been encountered by the particle. If this boundary is an interface, then a separate list records the particle identification number (id). The new processor id and local block id is determined from the connectivity information. Through the course of the particle tracking simulation corresponding to a given time step, multiple such particles may have to be sent to multiple processors. All of these are recorded in a 'send list'. Once all particles have been advected and their positions for that time step have been calculated, the send list is accessed. All the particles which are to be transferred to other processors are determined by the send list. Using this information, processor-specific lists of particles to be sent are made. The relevant particle data is bundled and sent to each processor.

Relevant particle data is as follows:

- Particle positions (x, y, z)
- Particle properties (3 velocity components and diameter)
- New local block ID

Thus 8 values corresponding to each particle are stored contiguously, and the process is repeated for every particle being sent to a specific processor. This information is bundled and sent using a non-blocking send using one of OpenMPI's intrinsic send functions.

In cases where no particle data is to be sent to a processor, a null value is sent. This has to be done, since the processor sending the information knows what information is to be sent to which processor. However, a receiving processor does not know whether to expect incoming information.

While receiving information from a processor, a blocking receive function intrinsic to OpenMPI is used. Prior to receiving information, another intrinsic function probes every other processor to see how much data is being sent by that processor. Based on the value returned by this probe, suitable memory is allocated and the data is received in a buffer. If the probe returns a null value, then receipt of information from that processor is skipped. This process is repeated for all the remaining processors, and the buffer data is then added to the existing particle data on the processor.

CHAPTER 7

Results

The results generated by the particle tracking code have been documented for four cases:

- Particle Tracking on a 2-D single block grid
- Particle Tracking on a 3-D single block grid
- Particle Tracking on a 3-D multi-block grid
- Particle Tracking coupled with REACTMB on a 3-D multi-block grid

The first two cases were run in a serial configuration, while the third and fourth cases were run in a parallel configuration.

7.1 Particle Tracking on a 2-D Single Block Grid

For the 2-D channel grid, 50 particles were injected at intervals of 250 time-steps with a diameter of $200\mu\text{m}$. Overall, 500 particles were injected in the domain over the first 2500 time-steps. The velocity field in the domain has been shown in Figures 5 and 6. Figures 7-9 show the particle locations at various time-steps due to their motion in the given flow field. In this case, the effect of gravity is ignored. Figures 10-12 show particle locations at the same time intervals, but with the addition of gravity. Figure 13 documents the differential trajectories of particles with different masses. For all simulations, the time-step was 0.001 seconds.

Figure 5: U velocity in 2-D channel

Figure 6: V velocity component in 2-D channel

Figure 7: Particle locations at $t = 5$ sec

Figure 8: Particle locations at $t = 12.5$ sec

Figure 9: Particle Locations at $t = 22.5$ sec

Figure 10: Particle Locations at $t = 5$ sec, with gravity

Figure 11: Particle Locations at $t = 12.5$ sec, with gravity

Figure 12: Particle Locations at $t = 22.5$ sec, with gravity

Figure 13: Tracks at $t = 22.5$ sec for particles with different diameters

Figures 7 – 9 show the motion of the particle cloud through the domain in the absence of gravity. As seen, the particles in the core region of the channel move quicker as compared the particles close to the wall. Because of the parabolic velocity profile which exists in the domain, the central core has a higher velocity than the regions close to the wall. The higher velocity of the fluid will generate higher drag forces on the particles in that region, accelerating them. This can be clearly seen from the results. Figures 10 – 12 show the same particle cloud moving through the domain under the influence of gravity. As seen from the results, the particles tend to settle towards the bottom of the channel.

Figure 13 shows trajectories for five particles injected at the same location, with the same initial velocities, but with different diameters. These particles are allowed to traverse the domain under the influence of drag forces and gravity. The diameters of the particles corresponding to each trajectory is as follows:

Table 2: Particle diameters for each trajectory

1	$d_p = 5.0 \times 10^{-4} \text{ m}$
2	$d_p = 3.5 \times 10^{-4} \text{ m}$
3	$d_p = 2.0 \times 10^{-4} \text{ m}$
4	$d_p = 1.0 \times 10^{-4} \text{ m}$
5	$d_p = 1.0 \times 10^{-5} \text{ m}$

As seen from the figure, the varying masses of the particles result in different responses to the fluid flow and gravity. The particle ‘1’ gets caught in the recirculation zone and thus flows slightly backwards. As the mass of the particles decreases, the particles tend to follow the streamlines to a greater extent. Similarly, the smaller particles quickly relax to the fluid velocity.

7.2 Particle Tracking on a 3-D Single Block Grid

100 particles were injected from a face for the first 5 time-steps, and their motion was tracked by the code. Overall, 500 particles were tracked through the block. The time-step for integration was 1E-05 seconds. The diameters of all particles were considered to be uniformly distributed over a range of 10 to 100 μm . The 3-D grid and the velocity field is shown in Figures 14-15. The results for three time intervals are in Figures 16-18. For the results shown henceforth, gravitational effects are not considered.

Figure 14: 3-D single block grid

Figure 15: U and V velocity components

Figure 16: Particle locations at $t = 5E-5$ sec

Figure 17: Particle locations at $t = 2E-4$ sec

Figure 18: Particle locations at $t = 5E-4$ sec

As seen from Figures 16 – 18, the injected particles are successfully tracked in the 3-D domain.

In the second part of the simulation, two particles with different diameters, and hence different masses, were ejected normal to the flow with a velocity of 10 m/s, along the Y-axis, and their motion was observed. The results for their trajectories are shown in Figures 19-20. The trajectory on the left (bottom, in the top view as shown in Figure 20) corresponds to the larger particle with diameter equal to $100\mu\text{m}$, while the one on the right is for the smaller particle with diameter equal to $10\mu\text{m}$. The densities of both particles are the same. The time-step for the simulation was $1E-05$ seconds. As seen from these results, the smaller particle, with the smaller Stokes number, relaxes to the fluid velocity much quickly. The larger particle traverses a larger distance normal to the direction of the flow-field, and slowly starts moving in the direction of the flow. Note that the simulation for both particles was allowed to run for the same duration.

Figure 19: Particle locations at $t = 0.01$ sec; Isometric view

Figure 20: Particle locations at $t = 0.01$ sec; Top view

7.3 Particle Tracking on a 3-D Multi-Block Grid

The particle tracking code was used to simulate particle flow over a 42 block, curvilinear 3D grid. The blocks were mapped to 14 processors. 900 particles were injected in the domain, and allowed to advect using an available steady state solution. As seen from the following figures, the injected particles are driven by the flow from the nozzle inlet to the outlet. At the inlet face, particles have been injected at randomized locations. Each particle has a diameter of 100 μm . The simulation was run for a total of 0.01 seconds using a time-step of 1E-05 seconds.

Particles that hit the walls are assumed to stop at the walls. They are thus deleted from the simulation. As the simulation proceeds, almost all particles either intersect the wall, or leave the domain from the outlet.

Figure 21: Particle distribution at $t = 0.0$ s

Figure 22: Particle distribution at $t = 0.0025$ s

Figure 23: Particle distribution at $t = 0.005$ s

Figure 24: Particle distribution at $t = 0.01$ s

As seen from Figures 21-24, the particles were successfully tracked through a 3-D multi-block domain. The top surface of the grid is a wall, which creates a low velocity boundary layer at the top. The particles seen in the domain after 0.01 seconds are entrained in this boundary layer, and hence move slowly. It can be seen that the particles accelerate with the flow as they enter the converging section of the nozzle.

7.4 Particle Tracking coupled with REACTMB

The particle tracking code, integrated with the REACTMB code was used to simulate particle trajectories over a grid consisting of 786 blocks mapped to 74 processors. 100 particles were injected into the domain every 10 time steps. The simulation was run for 1600 time steps, injecting 16000 particles in the process. Figure 25 shows the grid which was used for the simulation. Figure 26 and Figure 27 give the velocity profiles at the $Z = 1e-04$ plane. Figure 28 and 29 give a magnified view of the velocity profiles at the inlet. These profiles are shown to give an idea of the flow field the particles are subjected to.

Figure 25: The grid (786 blocks)

Figure 26: U - velocity profile at $Z = 1e-04$

Figure 27: V - velocity profile at $Z = 1e-04$

Figure 28: U-velocity profile at $Z = 1e-04$ (Magnified view of the injection site)

Figure 29: V-velocity profile at $Z = 1e-04$ (Magnified view of the injection site)

Figure 30: Differential particle trajectories without DRW

As seen in Figure 30, the differential trajectories shown in various colors correspond to particles of a specific diameter. The red, blue green and black trajectories correspond to particle diameters of 10, 20, 50 and 100 μm , respectively.

As seen from the differential particle trajectories, the mass of the particles drastically changes their behavior when subjected to a flow field, due to different values of inertia. This physical phenomenon is captured perfectly by the code. The smaller particles relax to the existing flow field much faster and thus show lesser transverse displacement as compared to the larger particles.

In the next case, a realistic simulation of droplet injection was performed. It was postulated that the fuel stream from the injector undergoes spray break-up by two different phenomena in two separate regions: a core region with smaller droplets, and an annular region with larger droplets.

For the core, the following injection properties were used:

- Droplet size = $20\text{ }\mu\text{m}$
- Injection velocity = 79.8 m/s
- Droplet injection rate = $8.653\text{e}08\text{ droplets/s}$.

For the annulus, the following injection properties were used:

- Droplet size = $57\text{ }\mu\text{m}$
- Injection velocity = 76.51 m/s
- Droplet injection rate = $3.488\text{e}07\text{ droplets/s}$

Figure 31: Particle locations after 400 time-steps

Figure 32: Particle Locations after 800 time-steps

Figure 33: Particle locations after 1200 time-steps

Figure 34: Particle Locations after 1600 time steps

Overall, an average of around 150,000 particles were tracked over 1600 time-steps in this simulation. The results shown in Figures 33 – 36 show the positions of the particles at different time steps. It can be seen that the larger annular particles are slower to relax to the velocity of the flow-field. Due to higher inertia, they show greater transverse displacement than the particles from the core region, and also take longer to accelerate to the flow velocity, as can be seen from Figures 33-36.

A comparison of the cost of particle tracking and the CFD solver was performed by looking at the CPU times required to execute the flow and particle code segments individually. The results have been presented in Figure 37. As discussed in the description of the algorithm, barriers exist in the code to ensure time accuracy of particle data across processors. It is thus important to note that the times reported for particle tracking are the times required by the slowest processor.

Figure 35: Comparison of computation time

As we can see, the computation time required by the particle tracking code segment goes on increasing progressively till it oscillates. Since this is the time required by the slowest process, it is highly dependent on the number of particles per block and what processor that block is mapped to. However, the primary conclusion is the fact that the cost particle tracking on the slowest processor is larger than the cost for the CFD solver. This shows the need to optimize the parallelization of the particle tracking process. As seen, the time required for transferring particle data across processors is negligible.

Figure 36: Communication and Data output times per iteration

Conclusions

8.1 Summary of Completed Work

A Lagrangian particle tracking code capable of tracking a large number of particles on 3-D multi block curvilinear grids has been successfully developed and integrated with the REACTMB code, developed at NCSU. This code has been run on a parallel architecture to maximize performance. The results generated by the particle tracking code, both individually and in conjunction with the CFD code, have been presented.

The particle tracking is performed in three broad steps. The particles in the domain are located in the cells which contain them. A fast particle search algorithm is developed and implemented for this purpose. The initial guess search used in this algorithm is optimized to ensure minimum search time and computational cost. Flow quantities located at the grid nodes are then interpolated to the exact particle location. Trilinear interpolation is performed by mapping the hexahedral cell which holds the particle, to a unit cube using iso-parametric relations. The coordinates of the particle in this unit cube are determined iteratively and the flow variables are then interpolated. Finally, using the flow data at the exact particle position, the particle is advected in the flow-field using Newton's Second Law of Motion, with drag and gravitational forces on the body. The time step for this integration is extracted from the main CFD code to maintain time accuracy. After each particle is advected, the particles to be transferred or deleted from a certain process are marked. The particle data to be exchanged between processors is bundled and transferred between specified processors. Based on the data sent and received, the particle lists on each processor are updated, and all the processors are synchronized for the next iteration. Multiple auxiliary functions are defined to allow for specified boundary conditions and to calculate wall intersections.

The results show that the developed code can successfully track particles in a Lagrangian frame of reference, using an existing Eulerian flow field. Particles with different masses respond differently to drag and gravity, and this is suitably captured by the code. The results are in agreement with what we would expect from particles injected in the domain. The code also runs successfully in a parallel environment, tracking a large number of particles across multiple blocks and working in conjunction with the existing CFD code. This provides a very powerful and useful additional functionality to the existing REACTMB code.

8.2 Future Work

The code has been developed keeping in mind that possible extensions to existing functionalities as well as improving current modules in the code can be done easily. The work presented in this thesis serves as a starting point for more complex particle simulations. The current work assumes particles to be one-way coupled to the flow, however it can be extended to account for two-way coupling, with the particles affecting the flow field. For better and more realistic results, other aspects like inter particle interaction and droplet disintegration may also be incorporated by developing suitable models for each of these processes. Similarly, turbulence in the flow field may be better accounted for by using more comprehensive models.

A key feature of this code is its ability to run on parallel architecture. The cost comparison in the results section shows the necessity of optimizing the parallelization process. This can be achieved by adding GPU acceleration for certain parts of the code. The processes of particle search, interpolation and integration are completely independent for each particle. Thus, GPU acceleration would work extremely well for particle tracking, since the loop over the particles is massively parallelizable. Similarly, alternative parallelization procedures and paradigms may also be used to account for particle flows in addition to fluid flows while splitting the domain into blocks and mapping the blocks to different processors. Particle tracking has many and far reaching engineering applications, and the proposed future work will ensure it can be used as a viable tool to solve real world problems.

REFERENCES

- [1] Schäfer, F., & Breuer, M. (2002). Comparison of c-space and p-space particle tracing schemes on high-performance computers: accuracy and performance. *International journal for numerical methods in fluids*, 39(4), 277-299.
- [2] Oliveira, P. J., Gosman, A. D., & Issa, R. I. (1997). A method for particle location and field interpolation on complex, three-dimensional computational meshes. *Advances in Engineering Software*, 28(9), 607-614.
- [3] Chen, X. Q. (1997). Efficient particle tracking algorithm for two-phase flows in geometries using curvilinear coordinates. *Numerical Heat Transfer, Part A Applications*, 32(4), 387-405.
- [4] Marchioli, C., Armenio, V., & Soldati, A. (2007). Simple and accurate scheme for fluid velocity interpolation for Eulerian–Lagrangian computation of dispersed flows in 3D curvilinear grids. *Computers & Fluids*, 36(7), 1187-1198.
- [5] Breuer, M., Baytekin, H. T., & Matida, E. A. (2006). Prediction of aerosol deposition in 90° bends using LES and an efficient Lagrangian tracking method. *Journal of Aerosol Science*, 37(11), 1407-1428.
- [6] Mahdavianesh, M., Noghrehabadi, A. R., Behbahaninejad, M., Ahmadi, G., & Dehghanian, M. (2013). Lagrangian Particle Tracking: Model Development. *Life Science Journal*, 10(8s).
- [7] Byrde, O., & Sawley, M. L. (1999). Parallel computation and analysis of the flow in a static mixer. *Computers & fluids*, 28(1), 1-18.

[8] Zhao, B., Yang, C., Yang, X., & Liu, S. (2008). Particle dispersion and deposition in ventilated rooms: testing and evaluation of different Eulerian and Lagrangian models. *Building and Environment*, 43(4), 388-397.

[9] Van Walsum, T., Hin, A. J., Post, F. H., Sadarjoen, A., & Walsum, T. (1994). Particle tracing algorithms for 3D curvilinear grids. In *In Fifth Eurographics Workshop on Visualization in Scientific Computing*.