DESIGN AND IMPLEMENTATION OF A PARALLEL BOUNDARY ELEMENT METHOD SOLUTION FOR 3D PARTICLE FLOW PROBLEMS IN MICROCHANNELS

A DOCTOR OF PHILOSOPHY THESIS

in

Modeling and Design of Engineering Systems (MODES)

(Main fields of study : Computer Engineering & Manufacturing Engineering)

Atılım University

by

ZIYA KARAKAYA

JANUARY 2015

DESIGN AND IMPLEMENTATION OF A PARALLEL BOUNDARY ELEMENT METHOD SOLUTION FOR 3D PARTICLE FLOW PROBLEMS IN MICROCHANNELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

BY

ZIYA KARAKAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
DOCTOR OF PHILOSOPHY
IN
MODELING AND DESIGN OF ENGINEERING SYSTEMS (MODES)
(MAIN FIELDS OF STUDY : COMPUTER ENGINEERING &
MANUFACTURING ENGINEERING)

JANUARY 2015

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

Prof. Dr. İbrahim Akman
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Abdülkadir Erden
Program Chair

This is to certify that we have read the thesis "**Design and implementation of a parallel boundary element method solution for 3D particle flow problems in microchannels**" submitted by "**Ziya Karakaya**" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Ali Yazıcı Co-supervisor	Assist. Prof. Dr. Besim Baranoğlu Supervisor

Examining Committee Members:

Prof. Dr. Ali Yazıcı

Assist. Prof. Dr. Erol Özçelik

Assist. Prof. Dr. Besim Baranoğlu

Assist. Prof. Dr. Çiğdem Turhan

Assist. Prof. Dr. Cenk Balçık

Date: January 30, 2015

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ZIYA KARAKAYA

Signature :

ABSTRACT

DESIGN AND IMPLEMENTATION OF A PARALLEL BOUNDARY ELEMENT METHOD SOLUTION FOR 3D PARTICLE FLOW PROBLEMS IN MICROCHANNELS

Karakaya, Ziya
Ph.D., Modeling and Design of Engineering Systems
Supervisor : Assist. Prof. Dr. Besim Baranoğlu
Co-Supervisor : Prof. Dr. Ali Yazıcı

January 2015, 75 pages

A new formulation for tracking multiple particles in slow viscous flow for microfluidic applications is presented. The method employs the manipulation of the boundary element matrices so that a system of equations is obtained relating to the rigid body velocities of the particle to the forces applied on the particle. The formulation is specially designed for particle trajectory tracking and involves successive matrix multiplications for which Symmetric Multiprocessing (SMP) parallelisation is applied. It is observed that the present formulation offers an efficient numerical model to be used for particle tracking and can easily be extended for multiphysics simulations in which several physics are involved.

Keywords: particle tracking, boundary element method, Stokes' flow

ÖZ

3 BOYUTLU MİKROKANALLARDA PARÇACIK AKIŞ PROBLEMLERİ İÇİN SINIR ELEMAN YÖNTEMİ TABANLI ÖZEL BİR PARALEL FORMÜLASYON TASARIMI VE UYGULAMASI

Karakaya, Ziya
Doktora, Mühendislik Sistemlerinin Modellenmesi ve Tasarımı
Tez Yöneticisi : Yrd. Doç. Dr. Besim Baranoğlu
Ortak Tez Yöneticisi : Prof. Dr. Ali Yazıcı

Ocak 2015 , 75 sayfa

Bu çalışmada mikroakışkan uygulamalarındaki kıvamlı yavaş akışta birden çok parçacığı izlemek için yeni bir formülasyon sunulmaktadır. Yöntem, sınır eleman matrislerinin manipülasyonu işlemininden sonra, parçacığın katı bünye hızları ile üzerine etki eden kuvvetleri ilişkilendiren bir denklem sistemi elde etmektedir. Formülasyon, SMP paralelleştirme yönteminin uygulandığı ardışık matris çarpımı işlemleri sonucunda özellikle parçacığın yörüngesinin takibi için tasarlanmıştır. Mevcut formülasyon, parçacık izleme işlemi için kullanılmak üzere etkili bir sayısal model sunmaktadır ve kolay bir şekilde birden çok fiziksel etkinin içerildiği çoklu-fizik simülasyonları için genişletilebilir olduğu görülmektedir.

Anahtar Kelimeler: parçacık hareketinin takibi, sınır eleman yöntemi, Stokes akışı

to my Father and Mother

ACKNOWLEDGMENTS

I express sincere appreciation to my supervisor Asst. Prof. Dr. Besim Baranoğlu and co-supervisor Prof. Dr. Ali Yazıcı for their guidance and insight through the research.

I would also like to express my gratitude to Asst. Prof. Dr. Barbaros Çetin and Prof. Dr. Yalçın Mengi for their invaluable help in developing the formulation.

I offer my sincere thank to my thesis progress juri members Asst. Prof. Dr. Çiğdem Turhan, and Asst. Prof. Dr. B. Cenk Balçık for their advice and guidance in helping to shape this dissertation.

I, finally, offer sincere thanks to my wife Songül, to my lovely daughter Öykü Şölen, and to my family for their love, understanding, patience and support during the study.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ v

ACKNOWLEDGMENTS vii

TABLE OF CONTENTS viii

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ABBREVIATIONS xv

CHAPTERS

1 INTRODUCTION 1

1.1 General Procedure in BEM 1

1.2 Advantages and Disadvantages of BEM 2

1.3 A short note on particle tracking and microfluidics 6

1.4 BEM for Stoke’s flow 10

1.5 Aim, Scope and Significance of the Study 11

2 FORMULATION 14

2.1 BEM formulation of the Stoke’s flow in 3D 14

2.1.1 Impedance formulation 21

3 IMPLEMENTATION 25

3.1 Computational Issues and Mathematical Libraries 25

3.2 Numerical Linear System Solver 26

3.3 Conventional Method Implementation 27

3.3.1 Details of the Algorithm 28

3.3.2 Optimization 36

3.4 Paralellization Strategy 38

3.5 Implementation of the impedance formulation 41

3.6 Benchmark Environment 43

4 NUMERICAL RESULTS 45

4.1 Verification of the formulation 45

4.1.1 Flow in a square channel 45

4.1.2 Two sphere problem 47

4.1.3 Three sphere problem 50

4.2 Performance of the proposed algorithm 51

4.2.1 Analysis 1 52

4.2.2 Analysis 2 53

4.2.3 Analysis 3 54

4.3 Separation problem 56

4.4 Cytometry problem 58

5 CONCLUSIONS 60
  5.1 Further Study 61

REFERENCES 64

APPENDICES

A APPENDIX 69

CURRICULUM VITAE 72

LIST OF TABLES

TABLES

Table 4.1 Comparison of the drag forces on the spheres for three sphere problem 50

Table 4.2 Time spent and performance ratio 54

LIST OF FIGURES

FIGURES

Figure 1.1 A 2D ellipse discretized in **(a)** FEM, and **(b)** BEM 2

Figure 1.2 2D **(a)** 2D Meshing of a circular particle in rectangular channel (in FEM) **(b)** as the particle moves, mesh has to be changed accordingly . . . 3

Figure 1.3 **(a)** 2D Meshing of a circular particle in rectangular channel (in BEM), a major advantage is the boundary-only discretization **(b)** as the particle moves, the domain is not re-discretized, only the corresponding change in the boundary is applied 4

Figure 1.4 2D **(a)** FEM model when the particle is near the channel wall: the mesh should be fine between the particle and the wall so that the internal stresses can be evaluated more correctly **(b)** A similar BEM model requiring no additional effort when the particle is near the channel boundary 5

Figure 1.5 Propogation of elastic waves in semi-infinite medium (half-space) . 6

Figure 1.6 An example illustration of a LOC device 7

Figure 1.7 Using hydrodynamic forces to separate, sort or focus flowing particles 8

Figure 2.1 The first step in imposing rigid body boundary conditions - the augmentation of $\mathbf{H}$ with six new columns 19

Figure 2.2 The second step in imposing rigid body boundary conditions - column swaps 20

Figure 2.3 The augmentation of the LHS coefficient matrix with six new rows 21

Figure 3.1 General Activity diagram of Conventional Method 27

Figure 3.2 Activity diagram of Create System 28

Figure 3.3 Rectangular plane triangulation 20x4 32

Figure 3.4 Sphere triangulation shown in every iteration from 1 to 9 36

Figure 3.5 Abstract base class Shape 37

Figure 3.6 Sequential computation of G and H matrices 38

Figure 3.7 Parallel computation of G and H matrices 40

Figure 3.8 General Activity diagram of impedance formulation 41

Figure 3.9 NUMA architecture of Benchmark Environment 43

Figure 3.10 CPU architecture of Benchmark Environment 44

Figure 4.1 Two views from channel flow analysis 46

Figure 4.2 Development of the velocity profile after an obstacle in the square channel 47

Figure 4.3 Schematic drawing of the second benchmark problem 47

Figure 4.4 Comparison of BEM and analytical solution for far-field two sphere problem 48

Figure 4.5 Two different problems to demonstrate size effect 49

Figure 4.6 The near-field solution with BEM for large sphere 49

Figure 4.7 The near-field solution with BEM for small sphere 49

Figure 4.8 Schematic drawing of the benchmark problem 50

Figure 4.9 Example problem for performance analyses 51

Figure 4.10 Time spent in 1000 iterations when #element on particle is fixed to 384, and #element on channel is increasing 52

Figure 4.11 Performance ratio depending on the channel size, particle size is fixed 53

Figure 4.12 Performance ratio depending on number of particles 55

Figure 4.13 Time consumption depending on the number of processors 55

Figure 4.14 Speedup ratio depending on number of processor 56

Figure 4.15 The channel geometry for separation problem 57

Figure 4.16 Starting position of the two particles in separation problem 57

Figure 4.17 Trajectory of the two particles in separation problem 58

Figure 4.18 Screen captures of 27 particle moving inside the channel 59

Figure A.1 Development of the velocity profile after an obstacle in the square channel 70

LIST OF ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ACML	AMD Core Math Library
API	Application Programming Interface
ATLAS	Automatically Tuned Linear Algebra Software
BC	Boundary Condition
BE	Boundary element
BEA	Boundary element analysis
BEM	Boundary Element Method
BLAS	Basic Linear Algebra Subprograms
CC	Continuity Condition
CPU	Central Processing Unit
DLA	Dense Linear Algebra
ESSL	IBM's Engineering and Scientific Subroutine Library
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
FVM	Finite Volume Method
FS	Fundamental Solution
GE	Governing Equation
GIT	Gauss Integral Theorem
GPU	Graphics Processing Unit
LAPACK	Linear Algebra Package
libFLAME	High performance Dense Linear Algebra Library
LOC	Lab-on-a-chip
MKL	Math Kernel Library
MPI	Message Passing Interface
NUMA	Non-uniform Memory Access

OpenBLAS	Optimized BLAS
OpenMP	Open specifications for Multi Processing
PLAPACK	Parallel Linear Algebra Package
SMP	Symmetric Multiprocessing
SMA	Shared Memory Architecture
Re	Reynold's number, $Re = \frac{\rho UL}{\mu}$
ρ	Density of the fluid
μ	Viscosity of the fluid
u_i	Components of the velocity vector at any point in the fluid domain
t_i	Components of the traction vector at any point on the fluid domain boundary
t	time
p	pressure
P	Modified pressure, $P = p - \rho g\mathbf{x}$
g	Gravitational acceleration
$\mathbf{x}$	Cartesian coordinates
σ_{ij}	Components of the hydrostatic stress tensor
$\mathbf{A}$	Fixed point
$\mathbf{P}$	Varied point
$C(\mathbf{A})$	Free term evaluated at the point $\mathbf{A}$
$C_{ij}(\mathbf{A})$	Components of the free term evaluated at the point $\mathbf{A}$
S	Boundary of the solution domain
$u^*_{ij}(\mathbf{A}, \mathbf{P})$	First fundamental solution of the Stoke's flow
$t^*_{ij}(\mathbf{A}, \mathbf{P})$	Second fundamental solution of the Stoke's flow
$p^*_i(\mathbf{A}, \mathbf{P})$	First fundamental solution of the pressure equation in Stoke's flow
$q^*_i(\mathbf{A}, \mathbf{P})$	Second fundamental solution of the pressure equation in Stoke's flow
$\mathbf{G}$	System matrix of BEM, elements being the evaluated integrals of the first fundamentals solution
$\mathbf{H}$	System matrix of BEM, elements being the evaluated integrals of the second fundamentals solution
$\mathbf{u}$	Column vector containing the nodal values of the velocity vector

$\mathbf{t}$	Column vector containing the nodal values of the traction vector
$\mathbf{u}^B$	Rigid body translation vector of the particle
ω	Rotational velocity vector of the particle
$\mathbf{r}$	The relative positon vector of the particle point to the selected center of the particle

CHAPTER 1

INTRODUCTION

Boundary element method (BEM) has been proven to be a very effective tool for simulation of many applications in engineering, ranging from thermo-fluids and acoustics [1] to elastodynamics [2] and to fluid mechanics [3]. Especially, if the mathematical model derived from the engineering application (which is usually expressed in terms of one or more differential equations) is linear, BEM, when compared to other numerical methods (such as the finite element method - FEM) provides more accurate results, especially in the zones of singularity [4].

1.1 General Procedure in BEM

The general procedure in BEM involves [5]

- transforming the governing equations into integral equations
- transforming the domain integrals appearing in the integral equations to boundary integrals (surface integrals in 3D and contour/line integrals in 2D) using the Gauss Integral Theorem (GIT)
- solving these boundary integrals numerically by discretizing the boundary of the solution region, which allows the determination of the unknown boundary quantities

In the problems where the evaluation of the internal quantities are desired, the internal field variables can be computed after the solution of the boundary quantities with the use of these boundary quantities. Note that, this stage is post-processing, meaning there is no solution in this stage. The pre-determined now-known boundary quantities are used to evaluate the internal field variables employing a special boundary integral formulation.

1.2 Advantages and Disadvantages of BEM

Generally, for most engineering applications, the advantages of the BEM can be listed as:

- The BEM discretizes only the boundary of the solution domain (Figure 1.1b). When compared to domain discretization methods, such as FEM or finite volume method (FVM), this reduces the dimension of the geometry by one (for two dimensional problems, the discretization is only over a curve - which is one dimensional, and for three dimensional problems, the discretization is made over surfaces - which is two dimensional). Especially for complex geometries in three dimensions, modelling is much more simpler in BEM.

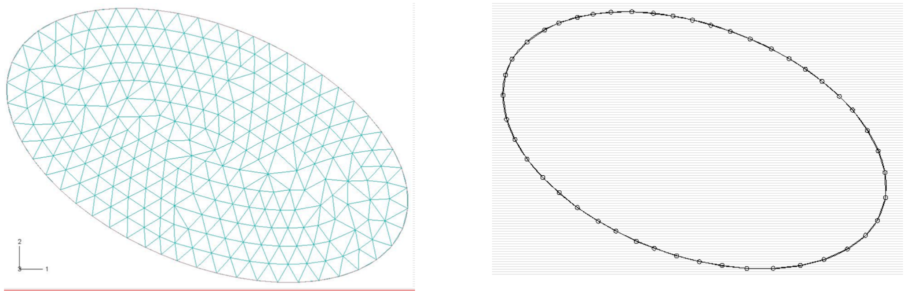
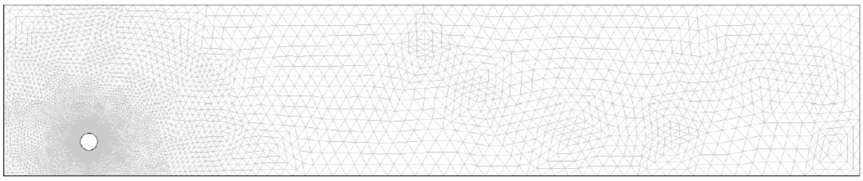

(a) (b)

Figure 1.1: A 2D ellipse discretized in **(a)** FEM, and **(b)** BEM

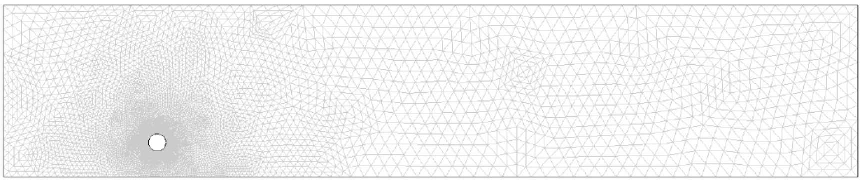
- The reduction in space dimension also serves another advantage: the number of unknowns in BEM is much less when compared with FEM or FVM. This is due to the boundary-only discretization nature of the BEM. A simple verification in 2D is given in Figure 1.1 where it can be seen that the number of nodes are much more in FEM when compared with BEM.

- FEM (or FVM) is completely numerical. That is to say, the governing equations (GE), the boundary conditions (BC) and the continuity conditions (CC) between the elements in the solution domain are satisfied approximately. BEM, on the other hand, is considered to be semi-analytical. That is because, the fundamental solutions (FS) for a given problem are determined exactly (within the solution domain and on the boundary), only the appearing integral equations are solved numerically. The CC are satisfied within the solution domain in an exact sense. That is why the boundary element (BE) solution is much more accurate than the finite element (FE) or finite volume (FV) solution.

- A major advantage of boundary-only discretization is the reduced modelling time in problems where successive remeshing is required. In FEM, if the solution domain is rapidly changing (as in the case of particle motion where moving particles change the solution domain), a considerable time is spent on the remeshing process of this changing domain. The reduction of space dimension, on the other hand, makes remeshing process comparably simple. A 2D FEM mesh of a particle within a rectangular channel is given in Figure 1.2 a. Besides from the complexity of the mesh (even it is in 2D) as the particle moves (Figure 1.2 b), the mesh has to be changed. In BEM, however, remeshing is much more simple - internal meshing is not required (Figure 1.3 a), and when the particle moves, remeshing is just moving the corresponding mesh (Figure 1.3 b).

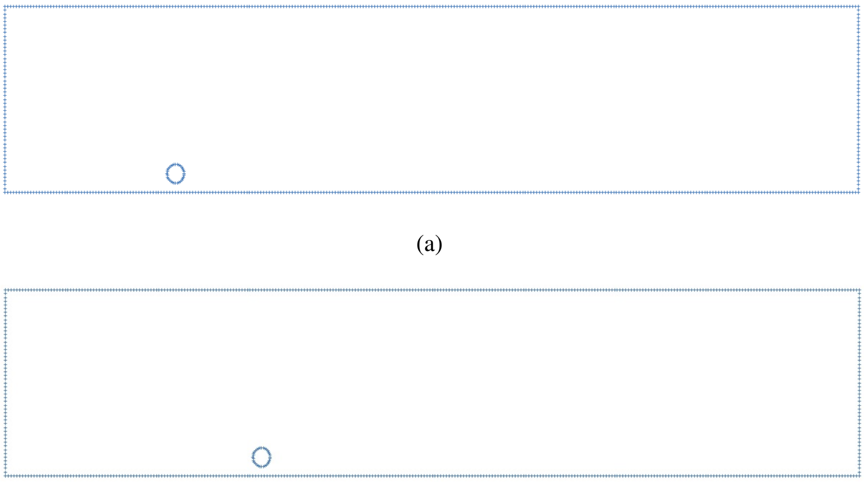


(a)



(b)

Figure 1.2: 2D **(a)** 2D Meshing of a circular particle in rectangular channel (in FEM) **(b)** as the particle moves, mesh has to be changed accordingly



(a)

(b)

Figure 1.3: **(a)** 2D Meshing of a circular particle in rectangular channel (in BEM), a major advantage is the boundary-only discretization **(b)** as the particle moves, the domain is not re-discretized, only the corresponding change in the boundary is applied

- Successive remeshing in FEM also has a major disadvantage: the field variables within the old mesh are transferred (in values) to the new mesh through interpolation. That is, the values in the new mesh are *approximated* using the values in the old mesh. Each remeshing step increases the approximation error - to a level that even solution may not be possible after several remeshing steps (for particle flow problems in 3D, this is the main reason of FEM not being an effective tool). Remeshing in BEM, on the other hand, does not have interpolation within the solution domain, since the solution domain is not discretized. This is the main advantage of BEM - even with a great amount of remeshing, the accuracy is not affected.

- FEM, and FVM, or any other numerical method that requires domain discretization determines the differential quantities in the solution domain (eg. internal flux, internal stress) by numerical differentiation. To do this, discretization should be sufficiently fine at the regions where differential quantities are to be evaluated (Figure 1.4 a). BEM, on the other hand, employs analytical differentiation within the solution domain for such differential quantities. Therefore, the differential quantities can be determined in any resolution (Figure 1.4 b) .

- If the region extends to infinity in one or more direction(s), FEM needs a bound-

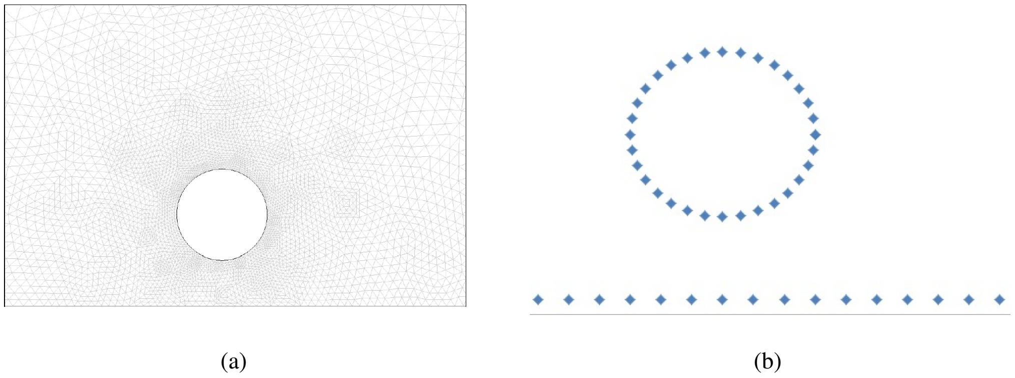

(a)
(b)

Figure 1.4: 2D **(a)** FEM model when the particle is near the channel wall: the mesh should be fine between the particle and the wall so that the internal stresses can be evaluated more correctly **(b)** A similar BEM model requiring no additional effort when the particle is near the channel boundary

ing box so that a mesh can be generated (or the use of so-called *infinite elements* required). This requirement makes it hard to model the radiation conditions in the directions that goes to infinity. Either a large bounding box should be selected (regarding the wave length in the solution domain) so that the reflected waves can be disregarded in the solution, or at the boundary of the bounding box, special boundary conditions should be imposed. BEM, on the other hand, natively takes into account the radiation conditions in the directions that extends to infinity. A demonstration of this fact is presented in Figure 1.5

With these advantages, also comes the disadvantages of BEM:

- The system matrices that is derived in BEM are full matrices possessing no symmetry. Therefore, the fast solution methods of sparse and symmetric matrices cannot be applied. In some problems, even if the number of unknowns are less in BEM, solution times can be more when compared to FEM.
- The integral equations derived in BEM involves singularities. Thus, special care should be given in numerical integration - mostly increasing total computation time.
- Most effort in research in numerical methods for the last decades is spent on the domain discretization methods, like FEM or FVM (even finite difference

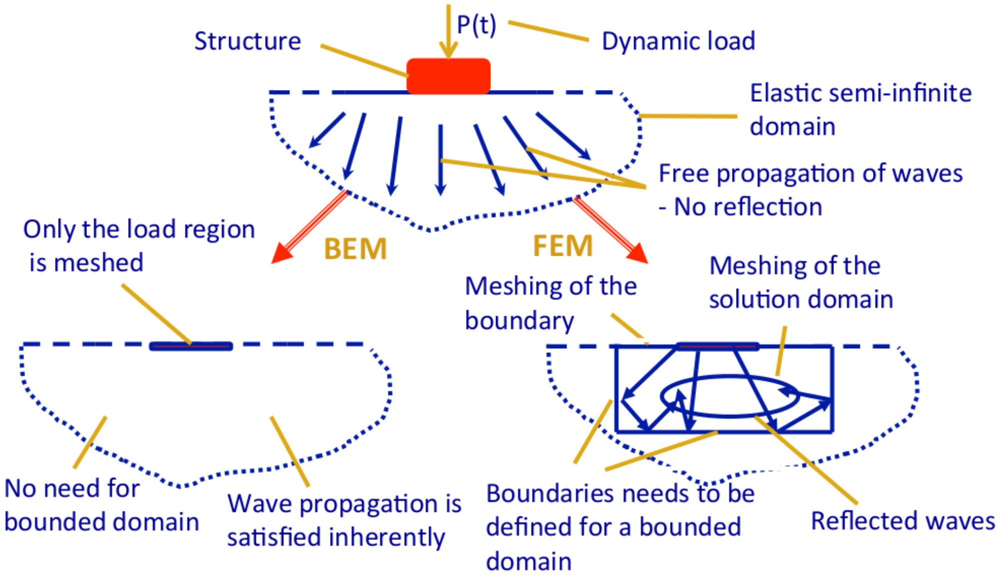

Structure
P(t)
Dynamic load
Elastic semi-infinite domain
Free propagation of waves - No reflection
Only the load region is meshed
BEM
FEM
Meshing of the boundary
Meshing of the solution domain
No need for bounded domain
Wave propagation is satisfied inherently
Boundaries needs to be defined for a bounded domain
Reflected waves

Figure 1.5: Propogation of elastic waves in semi-infinite medium (half-space)

method, FDM). The BEM, therefore, lacks of a general engineering program which has a good user interface to model the problems. This brings in some modelling difficulty when complex geometries are involved.

- A BEM formulation is mostly impossible when dealing with non-linear differential equations. Also, non-homogenous domains are hard to formulate. In such cases, special treatment of the non-linearity or non-homogeneity is required, which mostly destroys the boundary-only nature of the BEM [Brebbia-DRM].
- The BEM, having full and non-symmetrical system matrices, needs considerable amount of memory to store data. Although, with the recent developments in computer architecture, the memory sizes of even personal computers have been increased to a very high extent, this issue is still a major disadvantage of the BEM.

1.3 A short note on particle tracking and microfluidics

With new developments in material and manufacturing techniques, special devices in micro dimensions are being introduced to be used in many applications. Some examples are, micro pumps, micro actuators, MEMS devices, etc [6]. Of these listed devices, lab-on-a-chip (LOC) devices take special care [7]. These devices consist of a micro-channel within which a fluid flows with many inclusions in it. The special geometry and some additional electrical/magnetic equipment makes it possible to perform special processes over these particles, like separating cells from bacteria in blood to determine infection, or count the number of sperms in a given sample. An example illustration is shown in Figure 1.6 which is reproduced from [7], where some sample, as a droplet, is being placed on the so-called *chip* at one end and placing this *chip* to the Point-of-care testing (POCT) device, several processes, like reaction, delivery and analysis, can be performed on this sample.

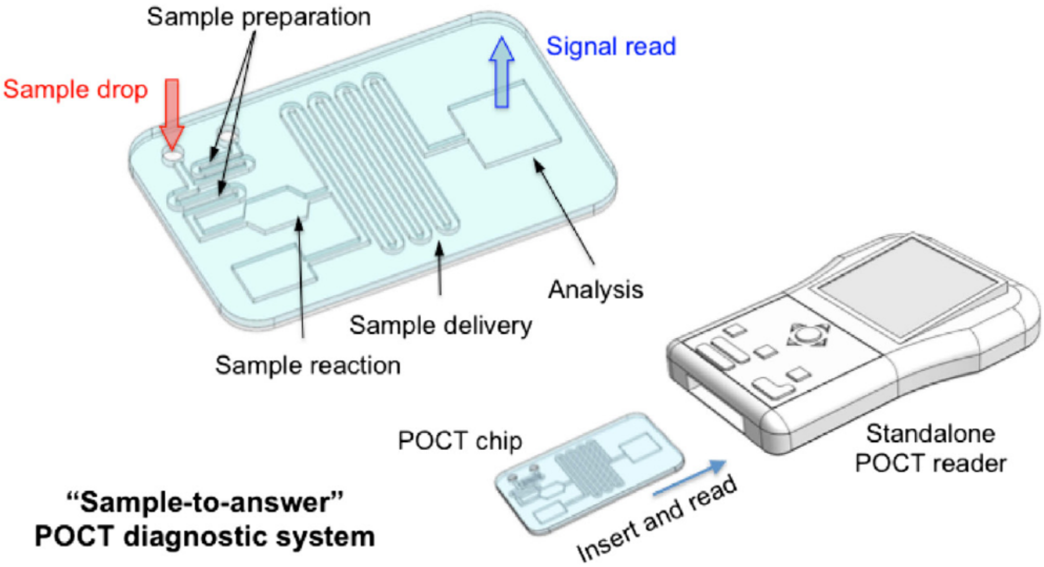
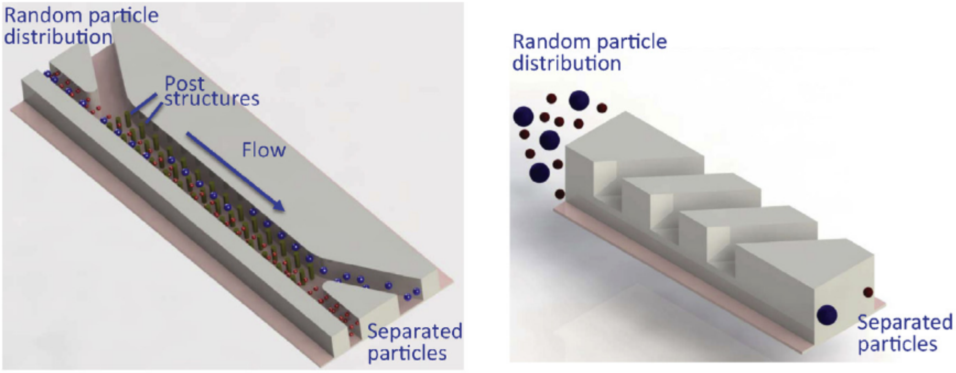
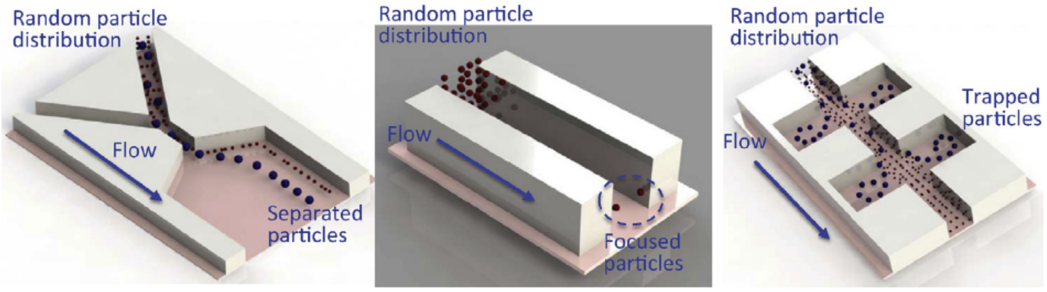


Figure 1.6: An example illustration of a LOC device [7]

To perform the tasks of a LOC device, designing proper micro-channels in which particles flow in an pre-expected trajectories play special role. The trajectory of the particles can be altered by applied hydrodynamic forces within the flow, or external excitation, like application of electrical, magnetic or acoustic field on the channel at prescribed locations, can be given. As an example process, several hydrodynamic separation methods are given in Figure 1.7. Here, DLD stands for *deterministic lateral displacement* where it is mostly used for bio-particle separation, sorting and focusing. A second method for use for the same purpose is hydrophoresis. It is also possible to reach the same goal by contraction/expansion (pinch segment) within the microchannel network together with the laminar flow profile where the particles are manipulated to flow at different streamlines. This is known as pinch-flow-fractionation (PFF). The last common approach is to use the inertia of the particles to obtain the same goal. All these are illustrated in Figure 1.7.



(a) DLD **(b)** Hydrophoresis


Random particle distribution
Flow
Separated particles
Random particle distribution
Flow
Focused particles
Random particle distribution
Trapped particles
Flow

(c) PFF **(d)** Inertial microfluidics

Figure 1.7: Using hydrodynamic forces to separate, sort or focus flowing particles [8]

Particle tracking problems have many important applications. Especially, with the recent developments in microfluidics technology, tracking several particles, especially with external forces acting, such as forces arising from the application of an electric, acoustic or optic field, gained special importance [8].

For an efficient design of microfluidic systems, the prediction of the particle tracks is crucial. Particle trajectory is the result of the interaction of the particle(s) with the external fields present.

One approach to model the particle trajectory within the microchannel is the stress tensor approach [9]. In this approach, the field variables are solved with the presence of the finite-sized particle. The resultant force on the particle can be obtained by integrating the appropriate stress tensor on the particle surface. In each incremental movement of the particle, the field variables need to be resolved. In many studies, this approach has also been successfully implemented [10, 11] to explore the nature of the particle flow within a microchannel. A rigorous simulation of the particle motion utilising tensor approach requires massive remeshing. For methods involving domain discretisation, such as finite element method (FEM) or finite volume method (FVM) not only the remeshing process is computationally expensive, but also at each remeshing step, some interpolating algorithms relating the field variables in the new mesh in terms of the variables of the old mesh are required which cause some loss in the accuracy. Moreover, the determination of the forces induced on the particles requires the calculation of gradient of the field variables. Therefore, for an accurate calculation of gradient of field variables, fine mesh is required on and within the close neighbourhood of the particle surface. Due to the computationally expensive nature, only 2D models with relatively coarse mesh and the motion of single particle have been worked on using FEM.

To overcome the remeshing problem for the simulation of particulate flow at macroscale, immersed boundary method [12] and fictitious domain method [13] have been proposed and implemented. Although these methods are computationally very efficient, to model the particle-particle interaction, some contact modelling is required which has a resolution that cannot be accepted for the simulation at microscale. Moreover, these methods are well established for flow simulations, but rare studies exist for the coupling of flow with the electrical and/or magnetic fields.

Considering the microchannel networks within the Lab-on-a-chip (LOC) devices, typical flow speed is low (resulting in very low Reynolds number) and the inertia forces are negligible (in magnitude) when compared with the pressure or the viscous forces. The flow is Stoke's flow which governs by a set of linear partial differential equations. Linear equations are suitable for Boundary Element Method (BEM). Since the BEM does not require meshing within the flow region and the exact calculation of the gradient of the field variables, it is preferable for particle tracking in a microchannel. Referring to these advantages, recently, Dustin and Luo [14, 15] implemented the BEM to simulate the particle trajectory within a microchannel under the action of electrophoretic and electro-osmotically driven flow field.

1.4 BEM for Stoke's flow

The listed advantages of BEM makes it an almost perfect tool for the analysis of Stoke's flow, since:

- The governing equations of Stoke's flow are linear differential equations
- The solution domain is mostly very large, and in some cases extends to infinity in one or more directions (some problems are named as *external flow*) where the solution domain is infinite medium.
- Most problems involve moving boundaries (eg., flow of bubbles in fluid, moving particles, fluid flowing over fluid, etc.), thus discretization and remeshing is an important issue.

Aside from the potential theory solutions of the Stoke's flow, it was Youngren and Acrivos [16] who developed the first BEM formulation for flow over axisymmetric particles. In this study, they employed rigid particles. The same authors also worked on deformable bubbles [17] in extensional flow. Later, Rallison and Acrivos further extended this formulation to viscous drops [18]. It is also important to cite the important studies of Pozrikidis, [19], where he worked on Stoke's flow over stationary and/or moving particles. In this study, the boundary conditions for a moving particle are introduced. Later, he published his work on boundary element method solutions of fluid mechanics as a book [3] and introduced a powerful library (BEMLIB) for interested researchers.

Of course, even with a tool like BEM, modelling time for many particles is extremely high. It is reported by [20] that by a 3D analysis with approximately 1000 particles took 20 minutes for a single velocity calculation on an Intel Paragon massively parallel computer using 1844 processors. Also, due to dense matrices formed in the analysis, the memory requirements (when motion of many particles are considered) become excessively high [1]. A solution to this problem appears to be the so-called fast-multipole techniques to be applied [21]. The main idea behind the multipole techniques is to use Taylor series expansions to lump (or "pole") far field effects. This way, it becomes possible to represent the influence from far-elements with larger poles, which brings in the advantage of less memory usage and computational time [22]. The method also proves to be well in parallelisation [23]. With the advantages given above, the method lacks three major disadvantages , the reason which in this dissertation it is not considered in detail:

- The boundary conditions to the problem should be given in velocity (not traction or other kind) so that a Fredholm integral equation of the first kind in direct method of BEM or Fredholm integral equation of the second kind in indirect method of BEM is obtained. This way, the stability of the system is ensured, so that an iterative solver can be used. But in pressure-driven micro-channel flows, this is not always possible; at the channel walls and at the inlet, velocity conditions are imposed, but on the exit, mostly, traction boundary conditions apply [24].
- In the context of this study, the derived work is extendable to cases where external force can be applied to the particles. No work on multipole method is presented in literature to deal with such cases.
- Although there appear several works on 3D applications of multipole method for several engineering problems, like time-harmonic elastodynamics [25], low frequency acoustic problems [26], elastic contact problems [27], etc., very few (and recent) studies exist for 3D application for particle tracking problems [28].

1.5 Aim, Scope and Significance of the Study

The main objective of this study is to present a new formulation for 3D particle tracking in micro-channels together with the implementation of a parallel code for use in microchannel flow problems. To attain this objective, this study will be restricted with the following scope:

- The continuum will be governed by the Stoke's equations. That is to say, the nonlinear convective terms in the Naviér-Stokes equations will be assumed to vanish for microchannel problems. This approximation stems from the dimension of the problem and the velocity of the fluid, resulting in a very low

Reynolds number.

- The flow will be assumed to be steady. This is to imply that the boundary conditions will not be changed in time, resulting in a steady velocity profile in the micro-channel. It will be assumed that, the movement of the particle(s) will not affect the steadiness of the flow (or the disturbance will be too small, so can be neglected). This assumption will be valid for small sized-particles (compared with the micro-channel dimensions) and sufficiently small flow velocity, which is the case in particle tracking problems in micro-channel flow. Assuming steady flow, the inertial forces will be neglected since in such a case, the magnitudes of the inertial forces will be much smaller then the pressure or the viscous forces.

- For simplicity, but without losing generality, the particle(s) involved in the analysis will be spherical. Other possible geometries, like ellipsoid, toroid, etc., will be considered in a future study. Also, the particle(s) will be assumed to be rigid.

- In parallelisation, SMP will be considered, and coding will be done using OpenMP. Other alternatives like MPI and GPU utilisation is considered as future studies.

To assess the study, several problems from literature are solved and the results are compared to analytical solutions and/or previous numerical results obtained in literature. Also, further analysis on the parallelisation characteristics is performed in comparison with the conventional method of solving Stokes flow problems.

One major novelty of the study is the derivation of a matrix relation where the motion parameters (linear and angular velocity of the center of gravity) related to a number of particles are obtained directly from the components of force and moment vectors applied at the given center. Through this matrix relation it becomes possible to directly solve the velocity of the particle under the application of forces. With only hydrodynamic effects in place, e.g., no external forces are applied to the particle(s) from other physics applications (like electric field, magnetic field, etc.), the free motion trajectories of the particles are obtained.

The matrix relation mentioned above gives rise to the second major novelty of the study, which is easy parallelisation. Since the formulation depends on matrix multiplications, it becomes possible to effectively and efficiently parallelise the algorithm in any parallel device. Although in this study only SMP parallelisation is implemented, it is possible to extend the work (as a further study) to other parallelisation techniques, such as MPI, GPU, GPU+CPU, MPI+GPU, etc.

The organisation of this thesis dissertation is as follows: The introduction chapter, which is the current chapter, is followed by Chapter 2, formulating the boundary element method for Stokes flow and presenting the proposed algorithm for particle tracking. The third chapter is on implementation of the method along with the conventional solution algorithm. Chapter 4 presents firstly the verification of the algorithm, comparing the results of several problems with analytical solutions. Later in Chapter 4, the performance of the proposed algorithm is compared with the conventional method. Lastly, in Chapter 4, two engineering problems are solved to assess the implemented code: the first example is on particle separation, and the second one is on cytometry, that is, to count the number of particles in a multi-particle flow. Last chapter, Chapter 5, gives a brief conclusion of the thesis study and proposes some recommendations on future perspectives of the study.

CHAPTER 2

FORMULATION

In this chapter, the particle tracking formulation is presented. For this, firstly, the steady Stoke's flow formulation with BEM is presented, followed by the imposition of the rigid body movement boundary conditions. This long introduction is in place since especially the imposition of the rigid body movement boundary conditions are not shown in detail in the literature. Therefore, the explicit explanation will be useful for further reference. Also, with this introduction, the later formulation will be more readable.

Later, in this chapter, the new formulation for particle tracking in microchannels, the impedance formulation, is presented in detail. The chapter will close with comments on the new formulation and its intended use.

2.1 BEM formulation of the Stoke's flow in 3D

For fluid flow in microchannels, since the dimension, L, and the flow velocity, U, is very small (in scale of micrometers and micrometer/s), the Reynolds number, defined in view of the density, ρ, and viscosity, μ, of the fluid,

$$Re = \frac{\rho U L}{\mu} \tag{2.1}$$

is sufficiently small to neglect the nonlinear convective terms in the Navier-Stokes equations. The resulting equations become

$$\rho \frac{\partial u_i}{\partial t} = -p_{,i} + \mu \left(u_{i,j} + u_{j,i} \right)_{,j} \tag{2.2}$$

It is important here to note that, in the above equation and in the equations that follow, Einstein's summation convention is in place, requiring a summation over a repeated index in its range. Also, to simplify notation, spatial derivatives are designated with a (,) in the subscript, e.g., $\frac{\partial u_i}{\partial x_j} = u_{i,j}$. In eq.(2.2), p represents the pressure and u_i are the components of the velocity vector. Further simplification in eq(2.2) can be done considering the flow to be steady. With this assumption, the inertial effects will be negligible (compared to the pressure and the viscous forces), which leads to the velocity formulation of the governing equations of the steady Stoke's flow:

$$-p_{,i} + \mu \left(u_{i,j} + u_{j,i}\right)_{,j} = 0 \tag{2.3}$$

At this point we impose the continuity condition requiring,

$$u_{i,i} = 0 \tag{2.4}$$

which would further reduce the equation to a form

$$-p_{,i} + \mu u_{i,jj} \tag{2.5}$$

The stress tensor can be defined as

$$\sigma_{ij} = -p\delta_{ij} + \mu \left(u_{i,j} + u_{j,i}\right) \tag{2.6}$$

Here, δ_{ij} are the components of the Kronecker's Delta. From eq(2.6) the traction components at the boundary of the solution domain can be defined as

$$t_i = \sigma_{ij} n_j \tag{2.7}$$

The direct formulation of BEM is possible with such definitions of the field variable u_i, the traction on the boundary, t_i and the pressure p. Note that, to incorporate the gravity, the pressure can be modified as

$$P = p - \rho g \mathbf{x} \tag{2.8}$$

Here, g is the gravitational acceleration and $\mathbf{x}$ are the Cartesian coordinates.

The BE formulation of the above equations are given in several references, one of which is replicated below [1]:

$$C_{ij}(\mathbf{A})u_j(\mathbf{A}) + \int\limits_S t^*_{ij}(\mathbf{A},\mathbf{P}) \cdot u_j(\mathbf{P}) \cdot \mathrm{d}A = \int\limits_S u^*_{ij}(\mathbf{A},\mathbf{P}) \cdot t_j(\mathbf{P}) \cdot \mathrm{d}A \qquad (2.9)$$

In this equation, $\mathbf{A}$ represents the fixed (evaluation) point and $\mathbf{P}$ represents the varied (integration) point. Note that $\mathbf{P}$ is on the boundary of the solution region, whereas $\mathbf{A}$ can be in the solution domain, on the boundary, or outside the domain. C_{ij} takes values 1, if $\mathbf{A}$ is in the solution domain, $\frac{1}{2}$ if it is on a smooth boundary or 0 if it is outside the solution domain. The field variables of eq(2.9) are the velocity components, u_i and the traction components t_i, and u^*_{ij} and t^*_{ij} are the first and second fundamental solutions of Stoke's equation.

It should be noted that, all integrals in the eq(2.9) are surface integrals (in 3D). Similar equation can be derived in case of 2D, where in this case, the resulting integral terms will be line integrals. It is obvious that, through the solution of eq(2.9), one can obtain the field variables, u_i and t_i at all points on the boundary as well as within the solution domain. With the calculated values of the mentioned field variables, it is possible to evaluate the pressure on the boundary and in the solution domain through the solution of the integral equation:

$$C(\mathbf{A})p(\mathbf{A}) = \int\limits_S q^*_j(\mathbf{A},\mathbf{P}) \cdot t_j(\mathbf{A}) \cdot \mathrm{d}A - \mu \int\limits_S p^*_j(\mathbf{A},\mathbf{P}) \cdot u_j(\mathbf{P}) \cdot \mathrm{d}A \qquad (2.10)$$

Note that, obtaining pressure is a post-processing step in BEM formulation - it is not required to obtain the boundary solution. In the above equation, the constant term C depends on the location of the fixed point $\mathbf{A}$, where the dependence is the same as C_{ij}.

Although found in literature [1], the fundamental solutions u^*_{ij}, t^*_{ij}, q^*_j and t^*_j are given in Appendix-A.

The equations (2.9) and (2.10) are total boundary integrals, from which an explicit solution of the boundary quantities is almost impossible. Thus, a numerical solution is attempted. In this study, since it can be considered as a post-processing step, the pressure equation will not be considered further, but the same discretization strategy that will be presented below can be employed to this equation too.

For simplicity, without losing generality, constant triangular elements will be used in this study to discretize the boundary of the domain. Employing a total of N elements on the boundary, eq(2.9) can be re-written as:

$$C_{ij}(\mathbf{A}_k)+\sum_{n=1}^{N}\int_{T_n} u^*_{ij}(\mathbf{A}_k,\mathbf{P}_n)\cdot t_j(\mathbf{P}_n)\cdot \mathrm{d}A = \sum_{n=1}^{N}\int_{T_n} t^*_{ij}(\mathbf{A}_k,\mathbf{P}_n)\cdot u_j(\mathbf{P}_n)\cdot \mathrm{d}A \quad (2.11)$$

which, when written for all nodes, can be expressed as a matrix equation:

$$\mathbf{H}\cdot\mathbf{u}=\mathbf{G}\cdot\mathbf{t} \quad (2.12)$$

Here, each component of the matrix $\mathbf{G}$ and the matrix $\mathbf{H}$ are obtained through the evaluation of the integrals

$$\mathbf{G}_{kn}=\int_{S_k}\begin{bmatrix} u^*_{11} & u^*_{12} & u^*_{13} \\ u^*_{21} & u^*_{22} & u^*_{23} \\ u^*_{31} & u^*_{32} & u^*_{33} \end{bmatrix}\mathrm{d}A \quad (2.13)$$

$$\mathbf{H}_{kn}=\int_{S_k}\begin{bmatrix} t^*_{11} & t^*_{12} & t^*_{13} \\ t^*_{21} & t^*_{22} & t^*_{23} \\ t^*_{31} & t^*_{32} & t^*_{33} \end{bmatrix}\mathrm{d}A \quad (2.14)$$

As can be observed, each element of the matrix $\mathbf{G}$ and the matrix $\mathbf{H}$ is a 3×3 sub-matrix. The vectors $\mathbf{u}$ and $\mathbf{t}$ contain the components of velocity and traction at each node. The imposition of boundary conditions requires the definition of one and only one of the couples (u_i^n, t_i^n) or a combination of these two at all points of the defined boundary, where n represents the node number and i represents the direction. In general case where no rigid body motion is given to any part of the solution domain, the boundary conditions are given as:

- Dirichlet condition: where the component of velocity, e.g., u_i is specified in a given direction i,

- Neumann condition: where the component of traction, e.g., t_i is specified in a given direction i

- Mixed condition: where a combination of the velocity and traction is specified in a given direction i, generally a linear relation is given as $a \times u_i + b \times t_i = c$ (where a, b and c are constants).

A possible solution requires one and only one condition to be imposed in a given direction, which implies that out of $6N$ unknowns, $3N$ are prescribed, leaving the system to be $3N$ equations for $3N$ unknowns. The imposition of boundary conditions is generally an easy task: through necessary column-swaps in the coefficient matrices $\mathbf{G}$ and $\mathbf{H}$ the unknown quantities are transferred to the left-hand-side (LHS) of the eq(2.12) and the known quantities are transferred to the right-hand-side (RHS), leaving the system as

$$\mathbf{K} \cdot \mathbf{x} = \mathbf{L} \cdot \mathbf{b} \tag{2.15}$$

where, $\mathbf{x}$ represents the unknowns vector and $\mathbf{b}$ represents the vector formed by the boundary conditions. After the multiplication of the RHS matrix and the boundary conditions vector to a single load vector $\mathbf{l}$,

$$\mathbf{l} = \mathbf{L} \cdot \mathbf{b} \tag{2.16}$$

the solution of the system of equations given in matrix form

$$\mathbf{K} \cdot \mathbf{x} = \mathbf{l} \tag{2.17}$$

determines the solution.

Imposing the rigid body motion conditions, on the other hand, requires special treatment. In the context of this study, we will consider the motion of undeformable particles. Thus, for any point on the particle, the velocity vector can be obtained through

$$\mathbf{u} = \mathbf{u}^B + \omega \times \mathbf{r} \tag{2.18}$$

where $\mathbf{u}$ is the velocity of the point on the particle, $\mathbf{u}^B$ is the velocity of the selected center of the particle, ω is the rotational velocity vector and $\mathbf{r}$ is the relative position vector of the particle point to the selected center of the particle. The imposition of eq(2.18) relates all velocity components (evaluated at the prescribed nodes on the moving particle) to six new parameters (per particle): three components of the translational velocity of the center of the particle and three components of the rotational velocity vector:

$$\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} u_1^B \\ u_2^B \\ u_3^B \end{Bmatrix} + \begin{Bmatrix} \omega_2 r_3 - \omega_3 r_2 \\ \omega_3 r_1 - \omega_1 r_3 \\ \omega_1 r_2 - \omega_2 r_1 \end{Bmatrix} \tag{2.19}$$

Therefore, this condition increases the size of the matrix system by six-per-particle. To accomplish this in the system of equations, six new columns are added to the LHS coefficient matrix. In the organisation of the matrices, it is assumed that the unknowns belonging to the moving particle are at the lower end of the vectors formed. In the schematic explanation given in Figure 2.1, $\mathbf{H}_0$ represents the columns of the matrix $\mathbf{H}$ which multiplies the components of the velocity vector at the non-moving boundary parts, $\mathbf{u}_0$, and $\mathbf{H}_P$ represents the columns of the same matrix which multiplies the components of the velocity vector on the particle, $\mathbf{u}_P$. The same notation is valid for $\mathbf{G}_0$, $\mathbf{G}$, $\mathbf{G}_P$, $\mathbf{t}_0$ and $\mathbf{t}_P$. After new columns are added to the LHS coefficient matrix, the contents of these columns are filled with the corresponding summations of the columns $\mathbf{H}_P$.

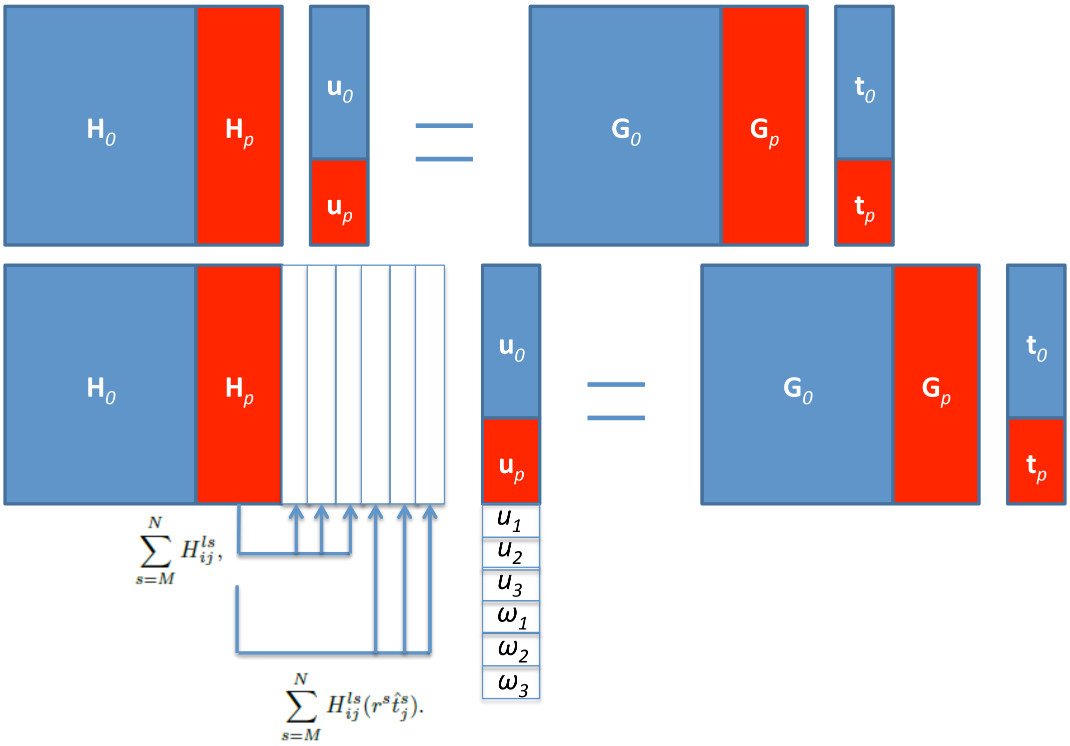


Figure 2.1: The first step in imposing rigid body boundary conditions - the augmentation of $\mathbf{H}$ with six new columns

After the first step is imposed, we can assume the particle velocities as the *knowns* of the problem, since all components of the vector $\mathbf{u}_P$ can be expressed in terms of the six new components, $\left\{ u_1^B \quad u_2^B \quad u_3^B \quad \omega_1 \quad \omega_2 \quad \omega_3 \right\}^{\mathbf{T}}$. The unknowns are the traction components on the particle, $\mathbf{t}_P$, which is transferred to the LHS by column swaps between $\mathbf{G}_P$ and $\mathbf{H}_P$, which is schematically shown in Figure 2.2

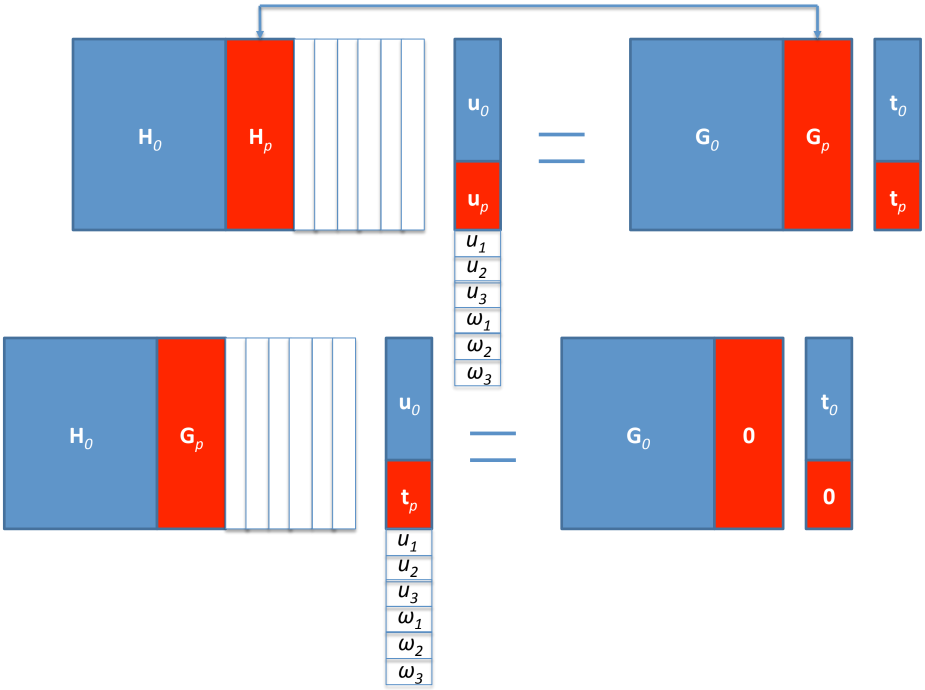


Figure 2.2: The second step in imposing rigid body boundary conditions - column swaps

At this point, the number of unknowns are $3N + 6$, where the number of equations are $3N$. Therefore, six new equations per particle are needed to make the system of equations solvable. These six equations come from the force equilibrium on the particle

$$f_i^B = \sum_{n=1}^{M} f_n = \sum_{n=1}^{M} \int_{C_n} t_i(\mathbf{P}_n)\mathrm{d}S = \sum_{n=1}^{M} t_i^n A_n \tag{2.20}$$

and the moment equilibrium

$$\mathbf{m}^B = \sum_{n=1}^{M} \mathbf{r} \times \mathbf{f}^n \tag{2.21}$$

where M is the number of elements on the particle. These static equilibrium equations can be represented in terms of six new rows added to the system as in Figure 2.3.

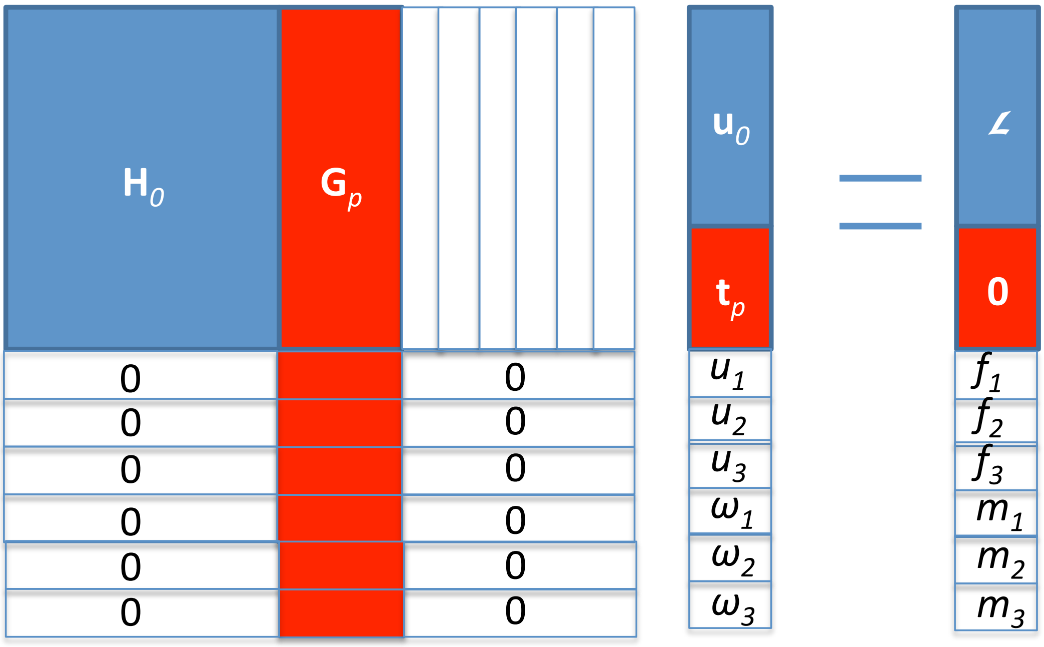


Figure 2.3: The augmentation of the LHS coefficient matrix with six new rows

It can be seen that, the imposition of rigid body boundary conditions will require addition of six new unknowns and six new equations per particle to the system of equations. Through the solution of the final system, the rigid body motion parameters can be determined, from which the velocities on each moving particle can be evaluated using eq(2.19).

2.1.1 Impedance formulation

The major drawback of the classical formulation (which is presented above) is the repeated solution of a considerably large linear system of equations. Although a very large part of the coefficient matrices that correspond to non-moving boundary is not re-evaluated, as the particle moves, parts of the coefficient matrices that are related with particle has to be updated and the system is to be re-solved.

In most applications of particle tracking problems, the main focus is on obtaining the trajectory of the particle(s). The determination of field variables on the boundary or in the solution domain does not have a significant importance. With this note in place, we present a new formulation for tracking multiple particles, the impedance formulation, which is based on a similar formulation derived by Mengi and co-workers in several studies [29, 30]. For this, we express the rigid body motion of a particle in matrix form as

$$\mathbf{u}^P = \mathbf{M} \cdot \mathbf{u}^B \tag{2.22}$$

where $\mathbf{u}^P$ is a column vector containing the velocity values at the nodes of the particle which is organised as

$$\mathbf{u}^P = \left\{ \left\{ \begin{array}{ccc} u_1 & u_2 & u_3 \end{array} \right\}^1 \left\{ \begin{array}{ccc} u_1 & u_2 & u_3 \end{array} \right\}^2 \cdots \left\{ \begin{array}{ccc} u_1 & u_2 & u_3 \end{array} \right\}^N \right\}^{\mathbf{T}} \tag{2.23}$$

and $\mathbf{u}^B$ is the vector containing the center velocity of the particle

$$\mathbf{u}^B = \left\{ \begin{array}{cccccc} u_1^B & u_2^B & u_3^B & \omega_1^B & \omega_2^B & \omega_3^B \end{array} \right\}^{\mathbf{T}} \tag{2.24}$$

and the coefficient matrix $\mathbf{M}$ is of size $(3N \times 6)$ whose elements are obtained through the eq(2.18). Similarly, defining a combined resultant force-moment vector for the particle as

$$\mathbf{f}^B = \left\{ \begin{array}{cccccc} f_1^B & f_2^B & f_3^B & m_1 & m_2 & m_3 \end{array} \right\}^{\mathbf{T}} \tag{2.25}$$

and the traction vector defined as

$$\mathbf{t}^P = \left\{ \left\{ \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \right\}^1 \left\{ \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \right\}^2 \cdots \left\{ \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \right\}^N \right\}^{\mathbf{T}} \tag{2.26}$$

a matrix relation as

$$\mathbf{f}^B = \mathbf{F} \cdot \mathbf{t} \tag{2.27}$$

can be obtained. Here, the (6×3) matrix $\mathbf{F}$ is obtained through the relations given in eq(2.20) and eq(2.21). At this point, we make a partitioning in the system of equations such as

$$\left[\begin{array}{cc} \mathbf{H}_{00} & \mathbf{H}_{0P} \\ \mathbf{H}_{P0} & \mathbf{H}_{PP} \end{array} \right] \left\{ \begin{array}{c} \mathbf{u}_0 \\ \mathbf{u}_P \end{array} \right\} = \left[\begin{array}{cc} \mathbf{G}_{00} & \mathbf{G}_{0P} \\ \mathbf{G}_{P0} & \mathbf{G}_{PP} \end{array} \right] \left\{ \begin{array}{c} \mathbf{t}_0 \\ \mathbf{t}_P \end{array} \right\} \tag{2.28}$$

where the index 0 refers to the non-moving boundary and P refers to moving boundary - thus $0P$ refers to components that are evaluated when the fixed point is on non-moving boundary and the integration is done over the moving boundary, etc. We will assume that all components $\mathbf{u}_0$ are known and $\mathbf{t}_0$ are unknown, easily obtainable when necessary column changes are performed with given boundary conditions. The first line of eq(2.28) requires

$$\mathbf{H}_{00}\mathbf{u}_0 + \mathbf{H}_{0P}\mathbf{u}_P = \mathbf{G}_{00}\mathbf{t}_0 + \mathbf{G}_{0P}\mathbf{t}_P \tag{2.29}$$

which leads to

$$\mathbf{t}_0 = \mathbf{G}_{00}^{-1}\left(\mathbf{H}_{00}\mathbf{u}_0 + \mathbf{H}_{0P}\mathbf{u}_P - \mathbf{G}_{0P}\mathbf{t}_P\right) \tag{2.30}$$

Inserting this to the second row equation of eq(2.28) we get

$$\mathbf{B}\cdot\mathbf{t}_P = \mathbf{A}\cdot\mathbf{u}_P + \mathbf{C}\cdot\mathbf{u}_0 \tag{2.31}$$

where

$$\begin{aligned}
\mathbf{A} &= \left[\mathbf{H}_{PP} - \mathbf{G}_{P0}\mathbf{G}_{00}^{-1}\mathbf{H}_{0P}\right] \\
\mathbf{B} &= \left[\mathbf{G}_{PP} - \mathbf{G}_{P0}\mathbf{G}_{00}^{-1}\mathbf{G}_{0P}\right] \\
\mathbf{C} &= \left[\mathbf{H}_{P0} - \mathbf{G}_{P0}\mathbf{G}_{00}^{-1}\mathbf{H}_{00}\right]
\end{aligned} \tag{2.32}$$

Inserting equations (2.22) and (2.27) to eq(2.31) and defining

$$\begin{aligned}
\mathbf{K} &= \mathbf{F}\mathbf{B}^{-1}\mathbf{A}\mathbf{M} \\
\mathbf{b} &= \mathbf{F}\mathbf{B}^{-1}\mathbf{C}\mathbf{u}_0
\end{aligned} \tag{2.33}$$

we obtain

$$\mathbf{K}\mathbf{u}_P = \mathbf{f}^B - \mathbf{b} \tag{2.34}$$

It is important to note here that,

- eq(2.34) is a system of linear equations where $\mathbf{K}$ is a square matrix of size $(6P \times 6P)$, and $\mathbf{u}_p$, $\mathbf{f}^B$ and $\mathbf{b}$ are column vectors of size $6P$ where P is the number of particles in the system.
- Inverting a considerably large matrix, $\mathbf{G}_{00}$ to obtain $\mathbf{G}_{00}^{-1}$, just once (at the beginning of the analysis), the rest of the formulation contains matrix multiplications and an inversion of the considerably very small matrix $\mathbf{B}$ which is of size $(3P \times 3P)$ to get $\mathbf{B}^{-1}$.
- The formulation readily involves the external forces to the system. This allows direct imposition of external forces found from different analyses involving different type of problems (electromagnetic, acoustic, etc.).

- The final form is obtained with matrix multiplications, for which parallelization is not only simple, but also effective.
- A special case of eq(2.34) is when there is no non-moving boundary, the case that arises in infinite medium. In such case, $\mathbf{b} = \mathbf{0}$ and $\mathbf{A} = \mathbf{H}_{PP} = \mathbf{H}$, $\mathbf{B} = \mathbf{G}_{PP} = \mathbf{G}$. The solution can be evaluated through

$$\begin{aligned} \mathbf{K}\mathbf{u}_P &= \mathbf{f}^B \\ \mathbf{K} &= \mathbf{F}\mathbf{G}^{-1}\mathbf{H}\mathbf{M} \end{aligned} \tag{2.35}$$

In the present study, time integration is performed in an explicit sense, using forward Euler difference, for simplicity. For more rigorous analysis, other time integration schemes, both explicit and implicit can be adopted.

CHAPTER 3

IMPLEMENTATION

In this chapter, the development of sequential and parallel code in C++, which implements both the conventional and proposed method described in previous chapter, are discussed and presented. First of all, the conventional method is implemented using the sequential algorithm and some optimization methodologies are applied to have efficient solutions. Then the equivalent parallel codes are developed.

Later, in this chapter, the development of codes, which is implementing the proposed method based on impedance formulation, is discussed. Chapter is concluded with information about the execution environment.

Before going into more depth, it would be better to discuss the preliminary decisions about the linear algebra operations that should be used during the development of application.

3.1 Computational Issues and Mathematical Libraries

The solution of the engineering problems, especially BE applications, requires excessive amount of dense linear algebra operations on floating point numbers. In such problems, there may be many operations on single dimensional arrays called vectors and two dimensional arrays called matrix. Especially vector-vector, matrix-vector and matrix-matrix operations are time consuming, and the developer would need to have efficient algorithms preventing overflow and underflow errors that would occur during computation. In addition to having large dense matrices, by the nature of problem being solved, the proposed method contains many matrix-matrix operations.

Almost all applications, developed for high computational engineering problems, use linear algebra libraries for efficiency and accuracy. Today, a number of libraries, such as BLAS (Basic Linear Algebra Subprograms) [31, 32, 33], ATLAS [34], Intel's MKL [35], AMD's ACML [36], and IBM's ESSL [37], are available in the literature. The Open Source implementation of BLAS library, named GotoBLAS, later ported and named as OpenBLAS [38], also exists and is not only the optimised version of the generic BLAS library, but also supports the multithreading on shared memory architectures. Linear Algebra PACKage (LAPACK)[39], and libFLAME [40] are linked to this multithreaded BLAS, so that they simply shifted their sequential codes to the parallel arena [41].

There is also another version of LAPACK called Parallel LAPACK (PLAPACK) which adds parallel algorithms on some operations to support more parallelism on SMP based computers. For the distributed memory architectures, the ScaLAPACK is a good example of such library using Message Passing Interface MPI [42].

The libFLAME, High Performance Dense Linear Algebra Library, underlined with OpenBLAS not only benefits from multithreaded BLAS library, but also contains many parallel algorithms supporting their operations. LibFLAME is preferred to be used in this study because of its easy of notation and high performance implementation of Dense Linear Algebra operations on Shared Memory Architecture (SMA).

3.2 Numerical Linear System Solver

For the solution of a linear system of equation given by $A \times x = b$, the matrix A is called ill-conditioned if $\mathrm{Cond}[A] = \parallel A \parallel \parallel A^{-1} \parallel$ is very large. Obviously, from the following inequality [43, 44],

$$\frac{\parallel \triangle x \parallel}{\parallel x \parallel} \leq \mathrm{Cond}[A] \frac{\parallel \triangle A \parallel}{\parallel A \parallel}$$

a large condition number may cause instability in the solution vector x.

The BE problem to be solved given in equation 2.17 and 2.35, by nature, involves a large asymmetric dense and highly ill-conditioned coefficient matrix. For such systems, instead of an iterative solver, a direct solver is opted for, among which LU-factorization is one of the most common in the literature [45, 46].

3.3 Conventional Method Implementation

An application, named *Conventional Method* in this study, is developed by implementing the algorithm given in Figure 3.1 where LU Solver is utilized to compute the results. The programming language C++ is used in order to apply Object Oriented Programming techniques, to produce flexible and efficient code.

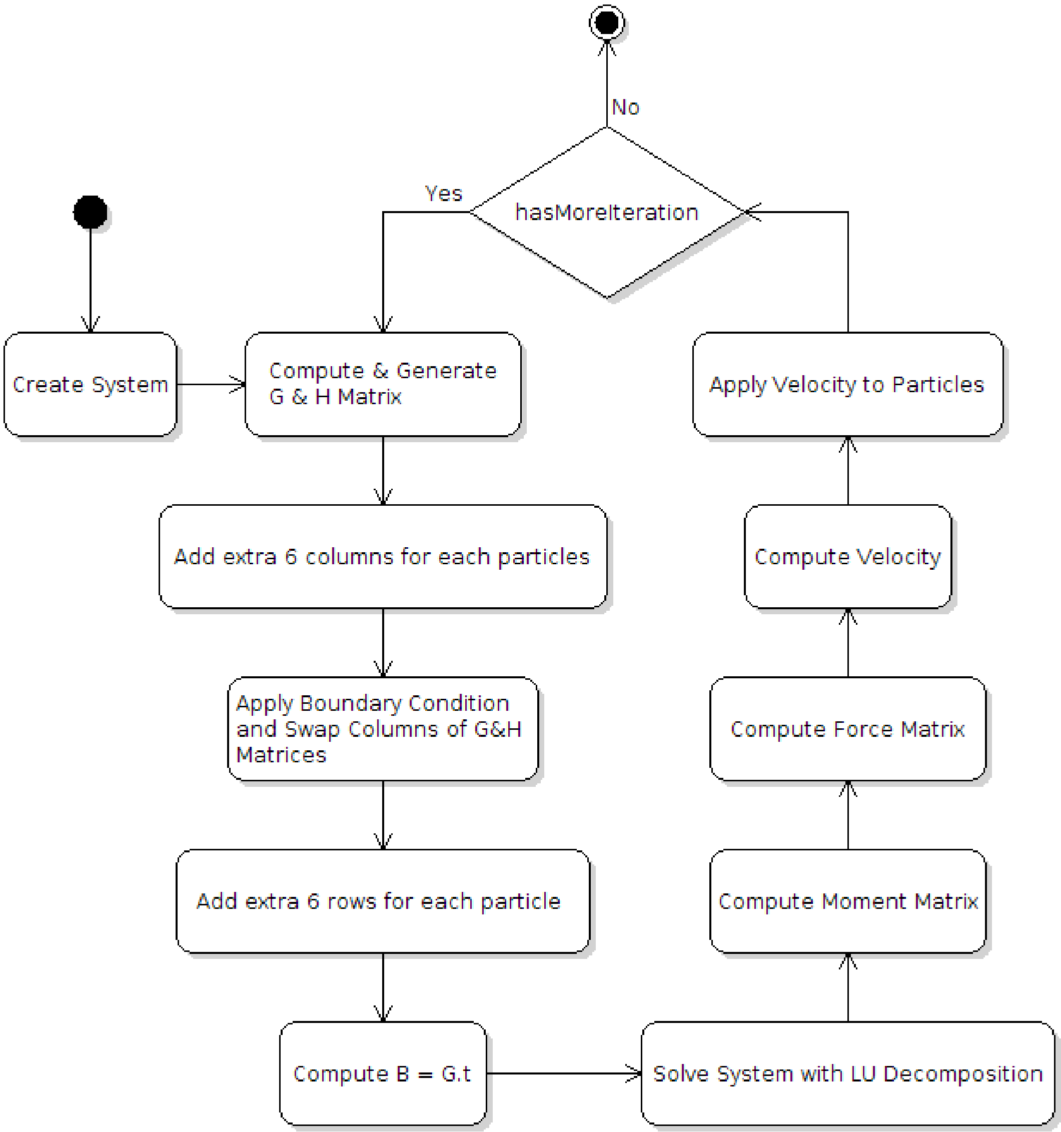

No
Yes
hasMoreIteration
Create System
Compute & Generate G & H Matrix
Apply Velocity to Particles
Add extra 6 columns for each particles
Compute Velocity
Apply Boundary Condition and Swap Columns of G&H Matrices
Compute Force Matrix
Add extra 6 rows for each particle
Compute Moment Matrix
Compute B = G.t
Solve System with LU Decomposition

Figure 3.1: General Activity diagram of Conventional Method

3.3.1 Details of the Algorithm

The algorithm, shown in Figure 3.1, starts with creation of the system, which contains a channel and particles. Channel walls are defined with many planes, and particles defined as spherical. Details of this process is given in Figure 3.2.

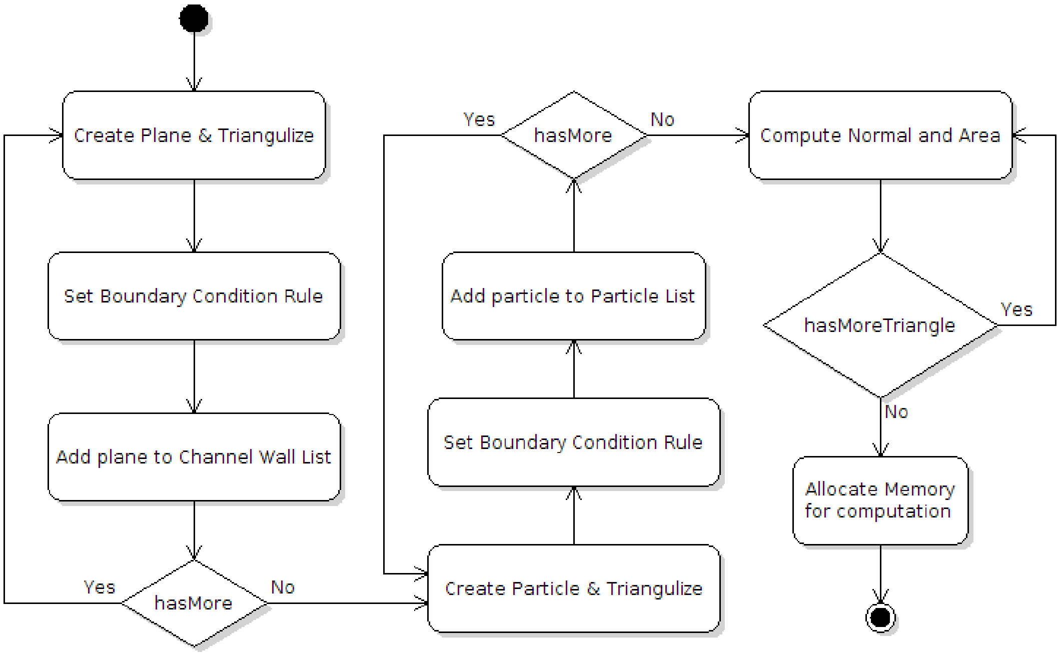

Create Plane & Triangulize
Set Boundary Condition Rule
Add plane to Channel Wall List
Yes
hasMore
No
Create Particle & Triangulize
Set Boundary Condition Rule
Add particle to Particle List
Yes
hasMore
No
Compute Normal and Area
hasMoreTriangle
Yes
No
Allocate Memory for computation

Figure 3.2: Activity diagram of Create System

During the creation of any `Shape` type, such as `Plane` or `Sphere`, triangulation of the mesh is also done in the constructor function. For example when constructing the planes of channel walls, the object constructor requires its coordinates in the order of left top (`lTop`), left bottom (`lBot`), right top (`rTop`), right bottom (`rBot`) so that the normal direction of the triangles, that will mesh the shape, can be decided. The triangulation process is defined in a way that all triangles meshing the shape has the same normal direction obtained by using the right hand rule.

The number of divisions of sides are also taken by the constructor, so that during the creation of shapes, the discretization process is done at the same time. There are two different methods of triangulation, one is defined by the number of increments where in each increment, all triangles of that shape are divided into two, the other is defined by number of divisions of sides, where the shape is divided into small rectangles first, and then each rectangle is divided into four triangle. Although the former method can be used for any kind of shape, the latter is more suitable for rectangular shapes.

A single object of `System` type keeps track of every elements of channel and particles in a thread safe list container, where by benefiting from the type provided in C++ Standard Template Library (STL). All triangles, meshing the shape, are also stored in an STL type vector container, which again is thread safe.

The following code is an example of such constructor from the `Plane` class, which is derived from `Shape` type.

```
Plane::Plane(Point lt, Point lb, Point rt,
             Point rb, int n, int m)
             : Shape(), tSplit(n+1),lSplit(m+1) {
    lTop = points->add(lt);
    lBot = points->add(lb);
    rTop = points->add(rt);
    rBot = points->add(rb);
    divideRect(lt, lb, rt, rb, n, m);
}
```

As can be observed from the above code, four `Point` objects are passed to constructor as parameters, and they are stored in a `PointContainer` type object named `points`, so that all `Point` objects are stored in the container and referenced by their index position in the rest of the application. The `add` method of `PointContainer` accepts a `Point` object as its parameter and adds to its internal `vector` container, if not already exists, and returns an integer number pointing to the index position of that object. If the passed `Point` object was already contained, it simply returns the index position without adding to container. The `add` method of `PointContainer` is as follows,

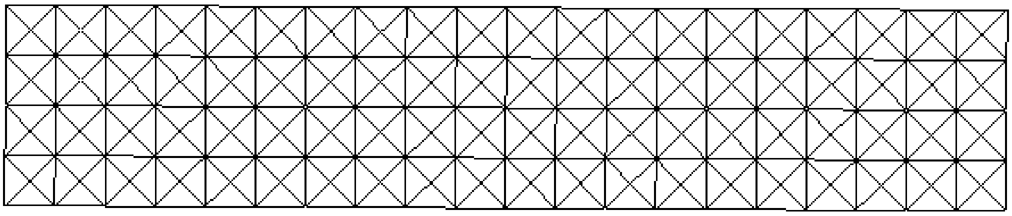
```
int PointContainer::add(Point &p){
    int place = find(p);
    if ( place != -1){
       return place;
    }
    pData.push_back(p);
    return pData.size()-1;
}
```

The last statement in the constructor of `Plane` executes the method `divideRect`, which is inherited from the abstract base class `Shape`. It is coded in a way that any rectangular shape can be mashed using triangles, which is called triangulation in this study.

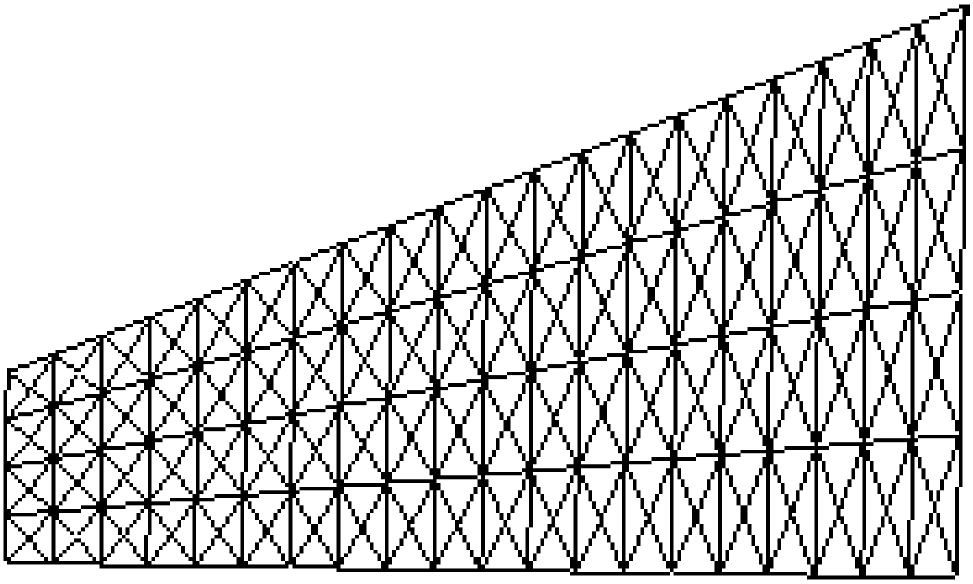
```
void Shape::divideRect(Point lt, Point lb, Point rt,
                       Point rb, int n, int m){
  int lSplit = m + 1;
  int tSplit = n + 1;
  Point pArr[lSplit][tSplit];
  int pPlace[lSplit][tSplit];

  pArr[0][0] = lt;
  pPlace[0][0] = points->find(lt);
  pArr[lSplit-1][0] = lb;
  pPlace[lSplit-1][0] = points->find(lb);
  pArr[0][tSplit-1] = rt;
  pPlace[0][tSplit-1] = points->find(rt);
  pArr[lSplit -1][tSplit-1] = rb;
  pPlace[lSplit-1][tSplit-1] = points->find(rb);

        // top side
  Point tVec = rt - lt;
  double tSize = tVec.length();
  double tSplitSize = tSize / n;
  tVec /=  tSize;

```

```
        // bottom side
  Point bVec = rb - lb;
  double bSize = bVec.length();
  double bSplitSize = bSize / n;
  bVec /= bSize;

  for (int i=0; i < tSplit; i++){
    pList.insert(pPlace[0][i] = points->add(pArr[0][i] =
                 pArr[0][0] + tVec*(i*tSplitSize)));
    pList.insert(pPlace[lSplit-1][i] = points->add(
                 pArr[lSplit-1][i] = pArr[lSplit-1][0] +
                 bVec * (i*bSplitSize)));

                // left side
    Point lVec = pArr[lSplit-1][i] - pArr[0][i];
    double lSize = lVec.length();
    double lSplitSize = lSize / m;
    lVec /= lSize;

    for (int j=1; j < lSplit-1; j++){
      pList.insert(pPlace[j][i] = points->add(pArr[j][i]
               = pArr[0][i] + lVec * (j * lSplitSize)));
    }
  }

  for (int i=1; i< lSplit; i++){
    for (int j=1; j < tSplit; j++){
       triRect(pPlace[i-1][j-1], pPlace[i][j-1],
                          pPlace[i-1][j], pPlace[i][j]);
    }
  }
}
```

The above code divides a rectangular plane into $(n+1) \times (m+1)$ small rectangles, as shown in Figure 3.3, and then calls `triRect` method overrided in `Plane` class, which triangulizes a given rectangular area with four equal triangles.



(a)



(b)

Figure 3.3: Rectangular plane triangulation 20x4

Similar to `divideRect` method, the `triRect` method is also inherited from the base class `Shape`, and is defined as a virtual function so that any child class can specialize it by overriding mechanism, so that there may be other derived types of `Shape` type, which has different triangulation strategy. This is the case in `Sphere` and `Plane` classes, where in the former class any rectangular area is divided into two triangles, but in the latter it is divided into four. The implementation is given below;

```
void Sphere::triRect(int lTop,int lBot,int rTop,int rBot)
{
  tList.push_back(Triangle(rTop, lBot, rBot));
  tList.push_back(Triangle(lBot, rTop, lTop));
}
void Plane::triRect(int lTop,int lBot,int rTop,int rBot)
{
  int rmp = points->add((*(points->at(lBot)) +
                        *(points->at(rTop)))/2);
  tList.push_back(Triangle(rTop, rmp, rBot));
  tList.push_back(Triangle(rBot, rmp, lBot));
  tList.push_back(Triangle(lBot, rmp, lTop));
  tList.push_back(Triangle(lTop, rmp, rTop));
}
```

As stated above there are two different triangulation mechanisms implemented in the codes. First mechanism is presented in `divideRect` method of `Shape` class. The second method is used in `Sphere` class and is suitable for spherical shapes. To show the methodology, it is better to start from the constructor of `Sphere` class.

```
Sphere::Sphere(Point c, double rr, int dc) :
               r(rr), divCount(dc), center(c)
{
   side = r / sqrt(3);
// side of the cube inside this sphere 3x^2 = r^2

// top
   divideRect(
     Point(center.x+side, center.y+side, center.z+side),
     Point(center.x+side, center.y+side, center.z-side),
     Point(center.x-side, center.y+side, center.z+side),
     Point(center.x-side, center.y+side, center.z-side),
     1, 1);

// bottom
   divideRect(
     Point(center.x-side, center.y-side, center.z+side),
     Point(center.x-side, center.y-side, center.z-side),
     Point(center.x+side, center.y-side, center.z+side),
     Point(center.x+side, center.y-side, center.z-side),
     1, 1);

// right is defined like above
// divideRect(...);

// left is defined like above
// divideRect(...);

// front is defined like above
// divideRect(...);

// rear is defined like above
// divideRect(...);

   this->subDivide(divCount);
}
```

The constructor of `Sphere` class accepts three parameters, first is the center point, second is the radius, and the third is the number of iterations that should be followed during triangulation process. In this code, the biggest cube that can fit into the sphere is defined and each side of the cube is triangulized by `divideRect` method. The last statement in the constructor executes `subDivide` method with the number of iterations. In every iteration, all the triangles of that shape are divided into two different triangles, as shown in Figure 3.4. For this purpose, the mid point of longest side of triangle is found and mapped on sphere, then using four points (three points from the edge of original triangle, plus one newly calculated point), two new triangles are defined according to the right hand rule in order to keep the normal direction the same as in the original triangle. In short, every triangle meshing the sphere is divided into two, and new triangle list is kept. This process is done by the following code:

```
void Sphere::subDivide(int cnt){
  int m;
  for (int i=0; i < cnt;i++){
    TriangleList newList;
    for(TriangleList::iterator i = tList.begin();
                               i != tList.end(); ++i) {
      Point mid = (i->getA() + i->getB()) * 0.5f;
      // point betwenn points A and B
      this->putOnSurface(&mid);// put in on the sphere
      pList.insert(m = points->add(mid));
      newList.push_back(Triangle(i->b, i->c, m));
      newList.push_back(Triangle(i->c, i->a, m));
    }
    tList.clear();
    tList.swap(newList); // use new set of triangles;
  }
}
```

The longest side of any triangle is always the side connecting the first and second edge. So that, there is no need to calculate the side lengths to find the longest one.

For any shape construction, in the design of the codes, object oriented techniques are being utilized so that any 3D shape can be defined, by inheriting the `Shape` abstract base class whose UML diagram is given in Figure 3.5. All other shape types, such as `Sphere` and `Plane` are derived from `Shape` class to have polymorphism, which helps to reduce the complexity of coding. Any new shape type can easily be defined by deriving the abstract base class `Shape`, which may make the codes more reusable

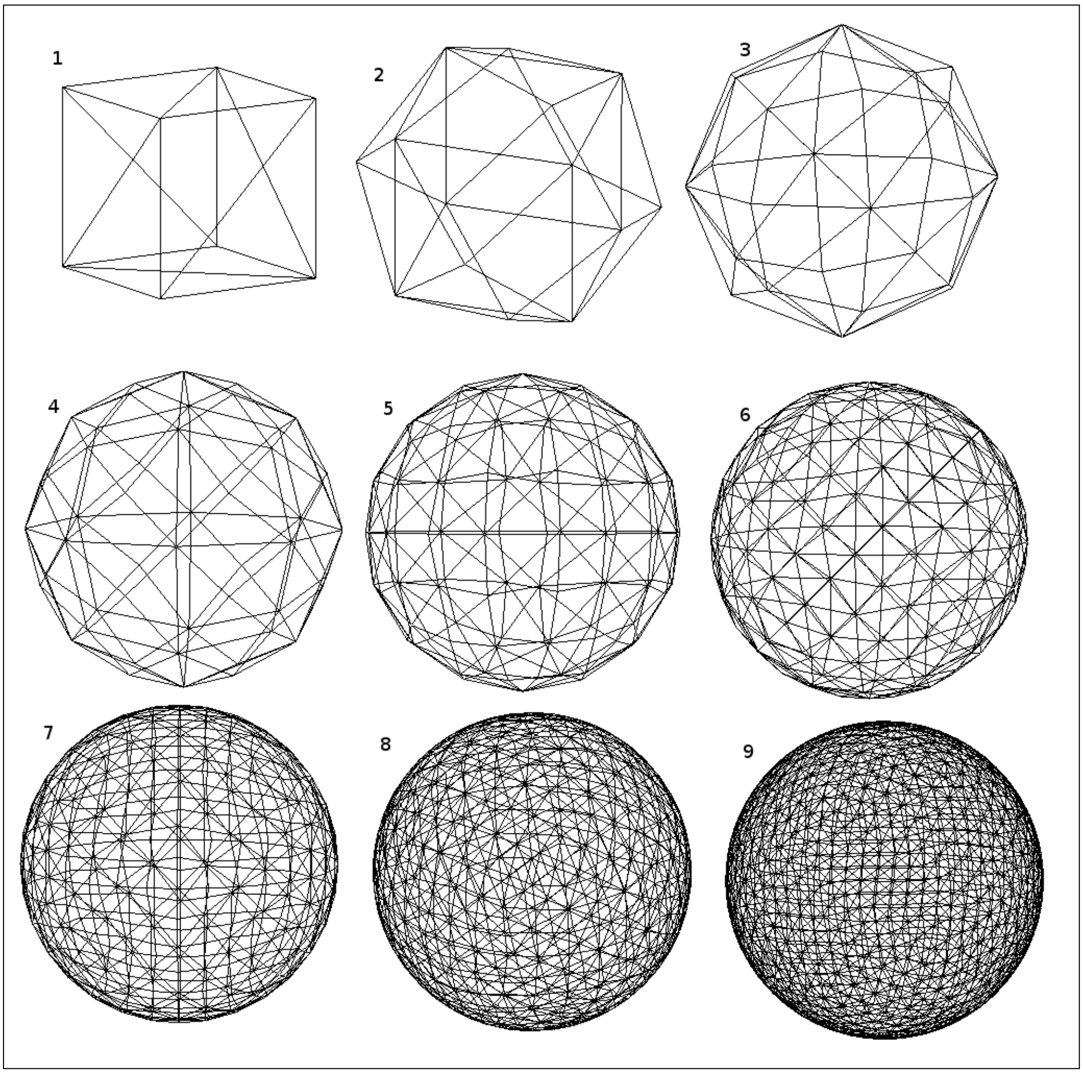


Figure 3.4: Sphere triangulation shown in every iteration from 1 to 9

and easily extensible in further studies.

In the application, everything is stored as an object or encapsulated in an object, and all objects are kept by an instance of `System` class with the composition technique, where the "Singleton Pattern" is used.

3.3.2 Optimization

Before introducing the parallelization into the codes, some code re-factoring is done for optimizing the performance in sequential solutions using reordering loop vari-

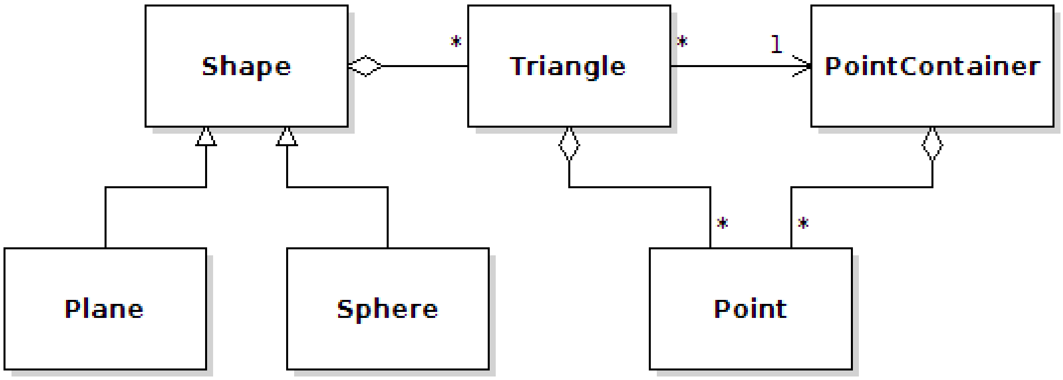


Figure 3.5: Abstract base class Shape

ables to make more cache hits [47]. For example, in integral computations, instead of keeping the fixed point A's, and traversing on varied point P's, that are given in equation-2.9, the loop variables are reordered so that outer loop fixed P's and inner loop traversed on A's. By doing this, re-computation of Gauss-points in P's triangle are eliminated.

Memory usage optimization is also done by keeping all the coordinates as a `Point` type in a big container, and using their index position in other objects, such as `Triangle`, and `Rectangle` type. For example, the triangle edges are stored as an index of three different `Point` objects, which are residing in an instance of `PointContainer`. The creation of `PointContainer` instance is an another example of using "Singleton Pattern". A `Point` object contains three floating point numbers, which can be used as a position on 3D coordinate system, or as any kind of vector representing velocity, displacement, etc. If a container were not used in keeping coordinates, to prevent redundant data, every triangle object would contain at least three `Point` objects for their edge position, which are shared at least with four other `Triangle` depending on its position. It is obvious that by re-factoring, approximately 75 percent of memory used for edge position of triangles are saved.

As another trivial optimization, the repeated calculations of the unchanged part of matrices, such as the computation of $\mathbf{G}_{00}$ and $\mathbf{H}_{00}$ given in equation 2.28 are avoided after the initial step. The activity diagram of creating $\mathbf{G}$ and $\mathbf{H}$ matrices are given in Figure 3.6.

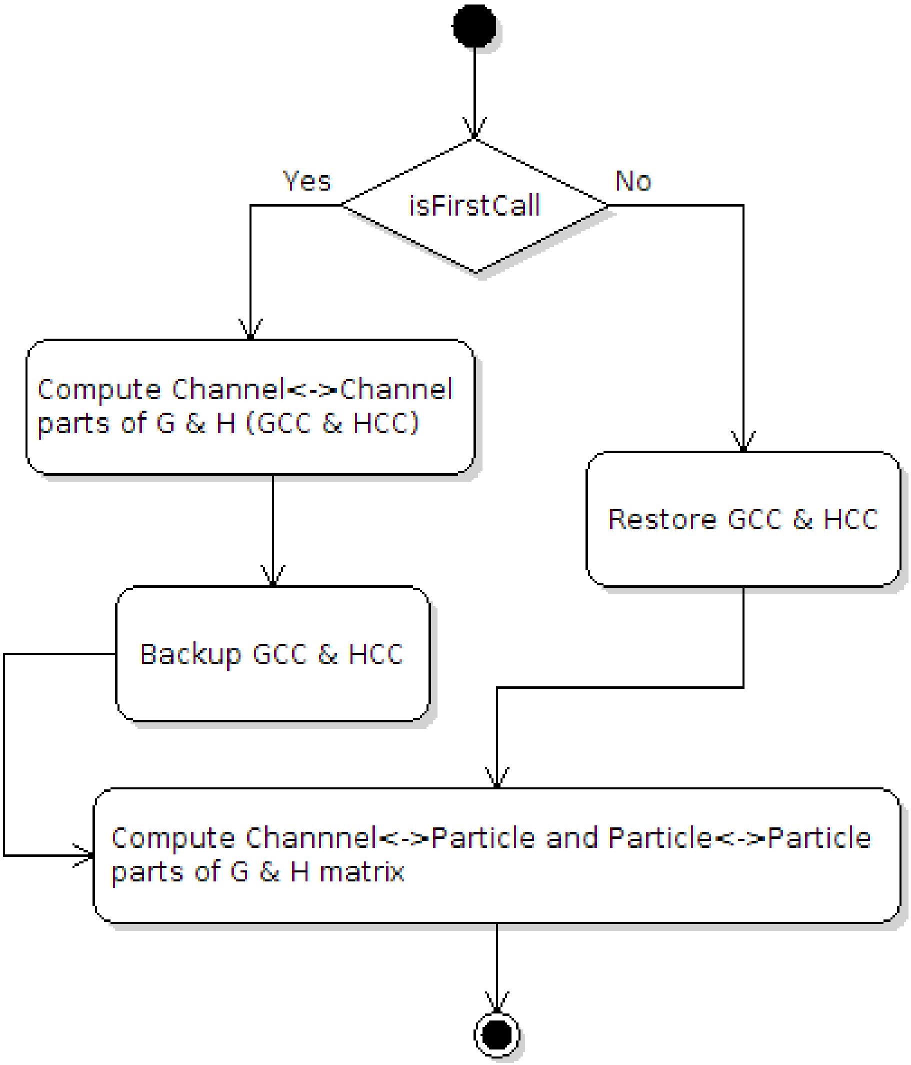

Yes
isFirstCall
No
Compute Channel<->Channel parts of G & H (GCC & HCC)
Restore GCC & HCC
Backup GCC & HCC
Compute Channnel<->Particle and Particle<->Particle parts of G & H matrix

Figure 3.6: Sequential computation of G and H matrices

3.4 Paralellization Strategy

At this point, it should be emphasized that the major objective of this study is not to implement the most efficient application, but to develop a parallel application which illustrates the efficiency of the proposed algorithm.

SMP parallelization techniques are used for making efficient use of computation power in solving the problem. In order to accomplish this, OpenMP [48] preprocessor directives are being utilised in parallel code generation.

OpenMP is an API which supports shared memory multiprocessing programming. OpenMP's SMA (Shared Memory Architecture) is based on *threading*, and gives programmer an API to use threading easily and safely. In the present problem, there is a huge amount of data to be processed and all the data is used in the subsequent steps of the computation. An advantage of SMA is that, there is no data transfer between threads, since all threads are executed on the same computer, but on different CPU cores, sharing the same memory.

An example of a OpenMP directive used in this study is given below:

```
#pragma omp parallel for private(i)
for (i=0; i < ipSz; i++){
  for (int j=0; j < tvSz; j++){
     doIntegrate(mu, internalPoints[i], *(triVector[j]),
                 intGMatrix, intHMatrix, i, j);
  }
}
```

This `pragma` directive causes the outer loop be executed in a parallel session by generating a number of threads.

As a complex example of an OpenMP directive, where the dynamic scheduling is also being utilised, is shown in the following code section:

```
#pragma omp parallel for private(r_dist, r_dist2, drdn,
    r_vect, xa, IG, IH, IGmult, IHmult,l)
    shared(t, vpArr, k, mu, IGBuffer, IHBuffer, triSize)
    schedule(dynamic)
    for (l=0; l < triSize; l++){
        // ......
    }
```

When developing a parallel version of the program, concentration is given on the parts where sequential code consumes much of the computing time.

As can be seen from the general activity diagram in Figure 3.1, after the system is created, all other operations are repeated in every iteration. Generating the system and triangulation of the mesh is done only once and these operations are relatively less time consuming and there is no need to parallelise this part.

Figure 3.7 shows how the parallelization is applied on the generation of **G** and **H** matrices, where the domain decomposition method is performed. As can be seen in eq(2.28), **G** and **H** matrices are decomposed into four sub-matrices. Since the channel does not change its shape, $\mathbf{G}_{00}$ and $\mathbf{H}_{00}$ are needed to be computed only once. In the first iteration, they are computed and stored once in the memory and retrieved many times in the subsequent iterations. Also, as discussed before, the loop reorganisation is applied to use more cache hits in order to increase the performance.

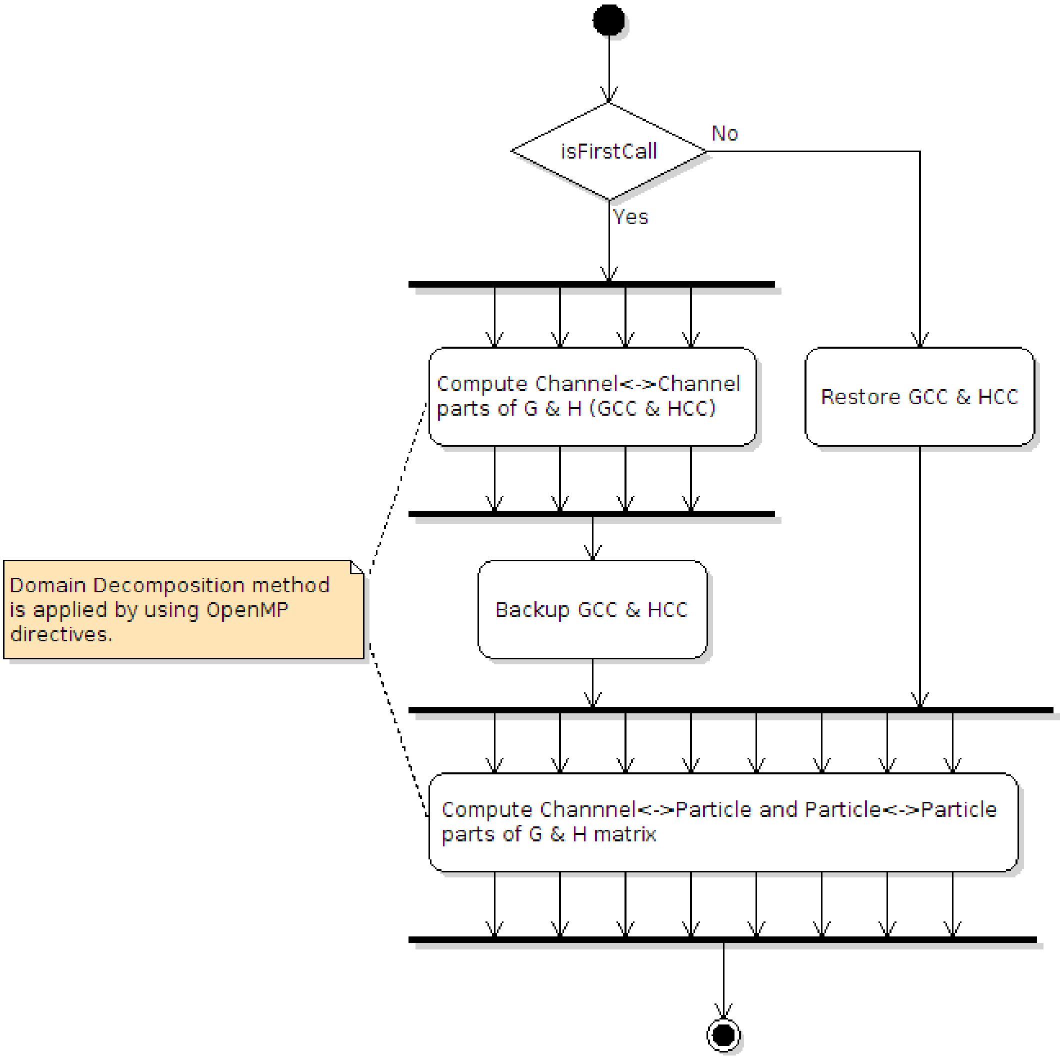

isFirstCall
No
Yes
Compute Channel<->Channel parts of G & H (GCC & HCC)
Restore GCC & HCC
Backup GCC & HCC
Domain Decomposition method is applied by using OpenMP directives.
Compute Channnel<->Particle and Particle<->Particle parts of G & H matrix

Figure 3.7: Parallel computation of G and H matrices

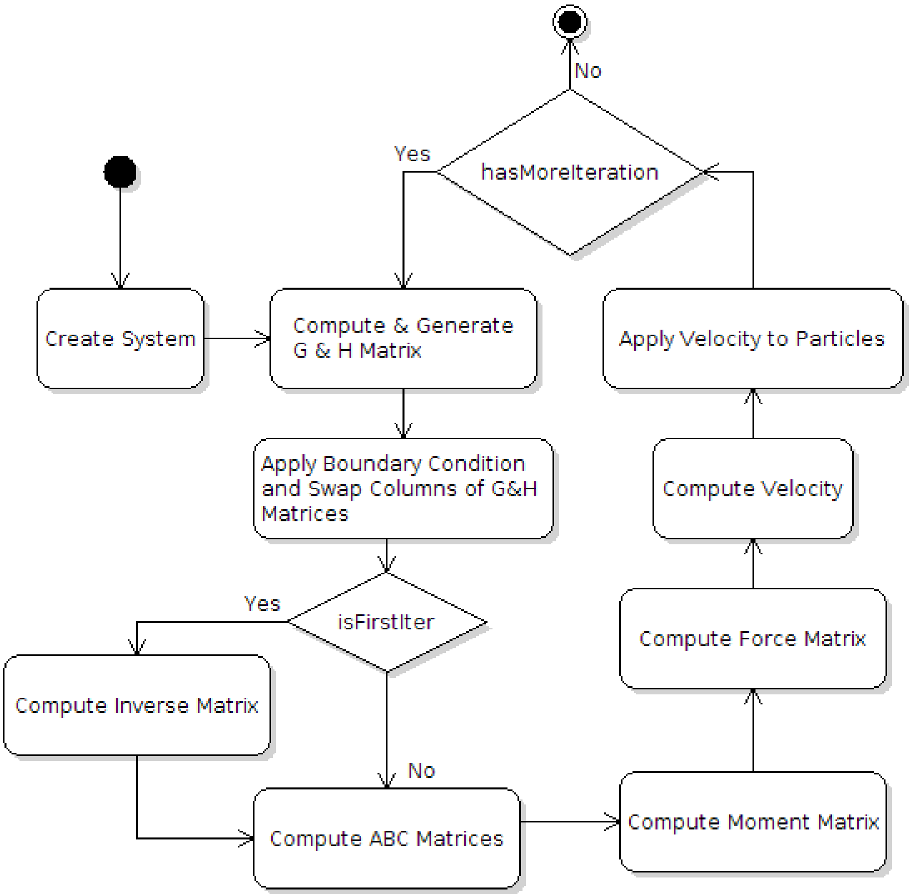


Figure 3.8: General Activity diagram of impedance formulation

Parallel performance is mainly gained by using libFLAME library, which is based on the micro-kernel operations of OpenBLAS that supports multithreading and parallelizing the generation of G and H matrices. In addition to this performance gain, application benefits from the OpenMP parallelization, especially in the computation of integrals.

3.5 Implementation of the impedance formulation

In order to show the accuracy of the new formulation with respect to the Conventional Method, impedance formulation is implemented as depicted in section 2.1.1, whose activity diagram is given in Figure 3.8, using both sequential and parallel algorithms.

The most important difference between the conventional method and the new method application is in the memory management of G and H matrices. In conventional method, both matrices are stored in contiguous memory locations, for which the memory space is taken by using a single `malloc` function call, and zeroed the content by using `memset` function call. In the new method, two new data types called `ZKMatrix` and `ZKArray` are defined to store all matrices and all vectors used in application. It should be noted here that, the memory used by these matrix and array instances are shared with the flame objects. Flame library has special object types for each construct, which reserves memory space in addition to other information, but they can also be constructed by attaching an already allocated memory space, either in row or column major order, which are reserved from somewhere else. Developed application benefits from this capability of flame objects so that flame object are linked directly to `ZKMatrix` and `ZKArray` objects.

As in the conventional method, the generation of G and H matrices are computed in each step but the G_{00} and H_{00} parts are computed once. In the solution by conventional method, G and H matrices are stored in contiguous memory as a whole block because of LU factorization and linear solver operations works on them as a whole. On the other hand, in the new method the matrices are physically divided into four, and each of them is defined as separate objects.

As stated above in the conventional method, LU operation is done on the whole G matrix, but the inverse of G_{00} is computed once only in the first iteration, where again LU decomposition is used, and is stored in another object called `G00Inv`. In subsequent iterations, the inverted matrix is retrieved from the memory and used in the rest of the calculations. As can be seen from the activity diagram, in each iteration there is a step where the A, B, and C matrices are all computed, as defined in equation 2.32.

In the proposed method, most of time spent in computing the result is used in matrix multiplications and in computing the inverse of B matrix depending on the structure of the problem to be solved.

3.6 Benchmark Environment

In the analysis, DELL R720 computer with 32 virtual Core (having 2 CPU with 16 real cores) CPU, having NUMA Architecture with 384 GB of RAM is being used to measure and compare the efficiency of both applications. Linux operating system, 64bit, is installed on this architecture and all libraries used in this study are re-compiled on this environment.

Output of Linux shell command "`lscpu`" is given in Figure 3.10, and the NUMA architecture is displayed by the output of shell command "`lstopo -no-io`" in Figure 3.9;

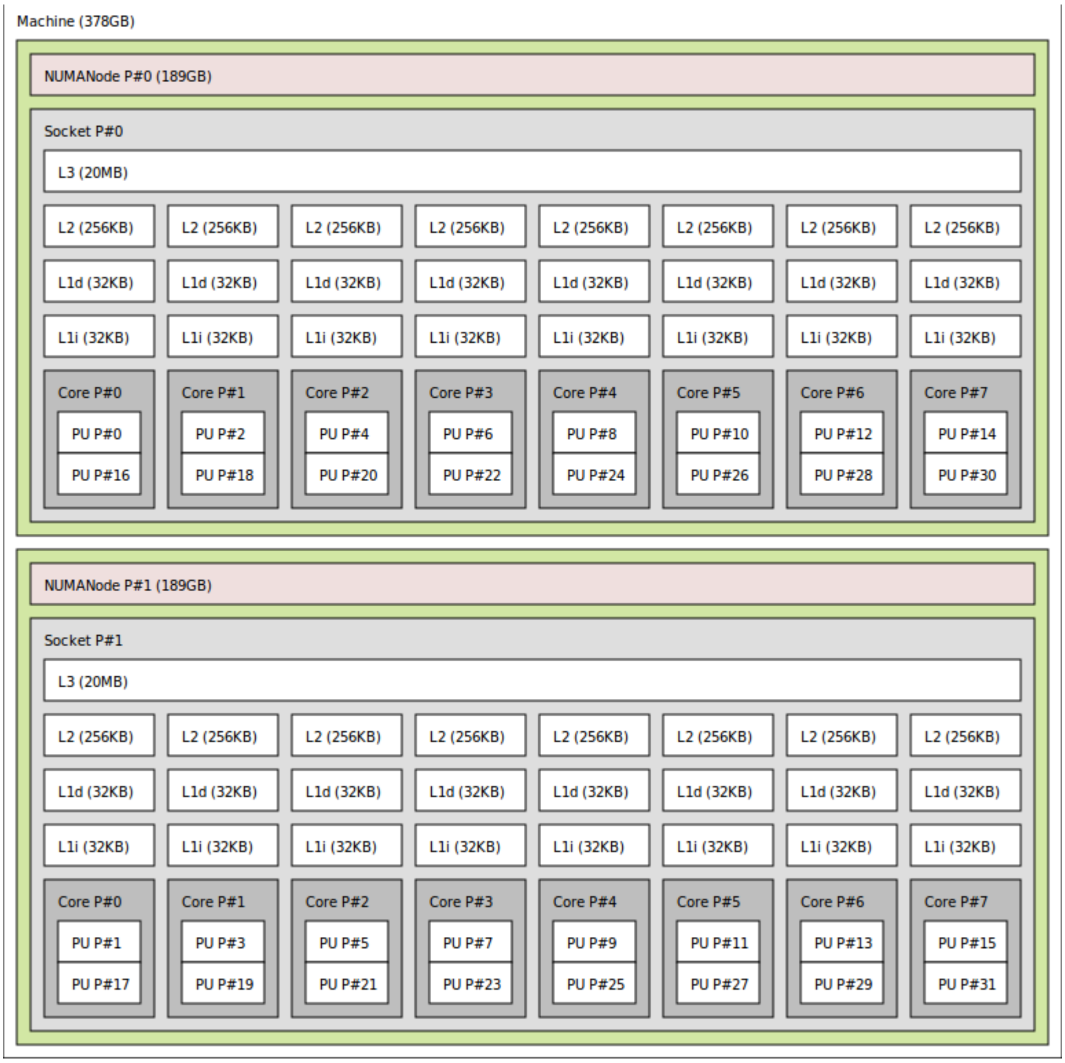
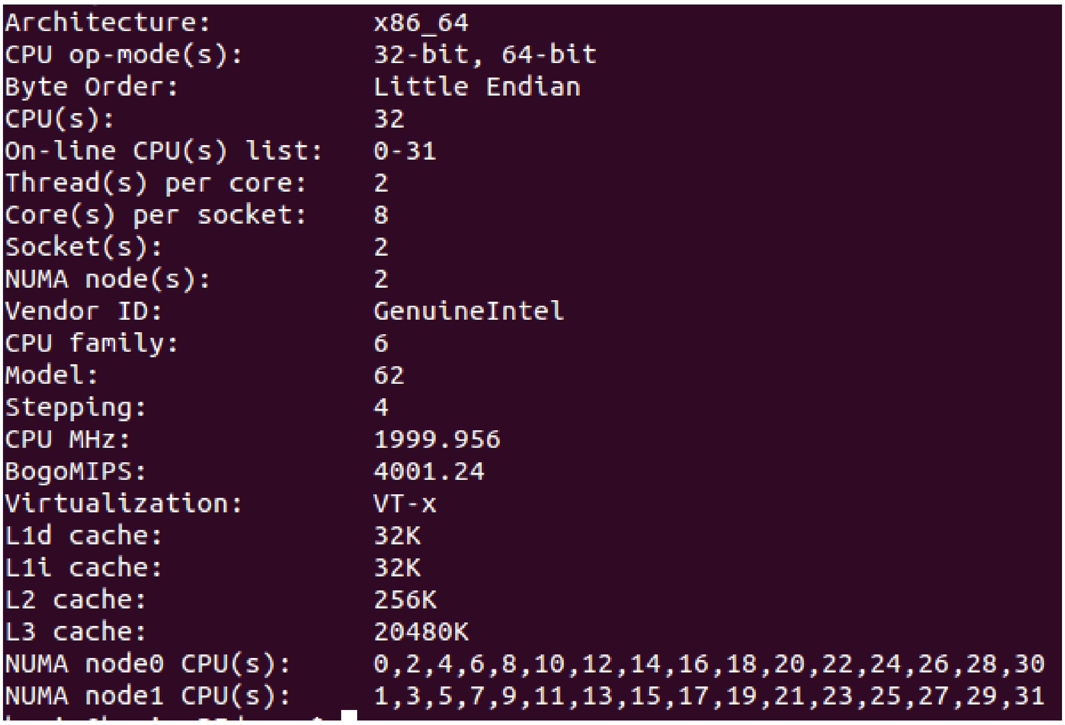


Figure 3.9: NUMA architecture of Benchmark Environment

As shown in Figure 3.10 architecture is `X86_64` of having 2 CPU sockets each of



```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 62
Stepping:              4
CPU MHz:               1999.956
BogoMIPS:              4001.24
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
```

Figure 3.10: CPU architecture of Benchmark Environment

having 8 cores are being utilised in benchmark operation. Since the architecture supports hyper-threading, two threads per core can be executed concurrently, the computer can be used to measure 32 cores.

CHAPTER 4

NUMERICAL RESULTS

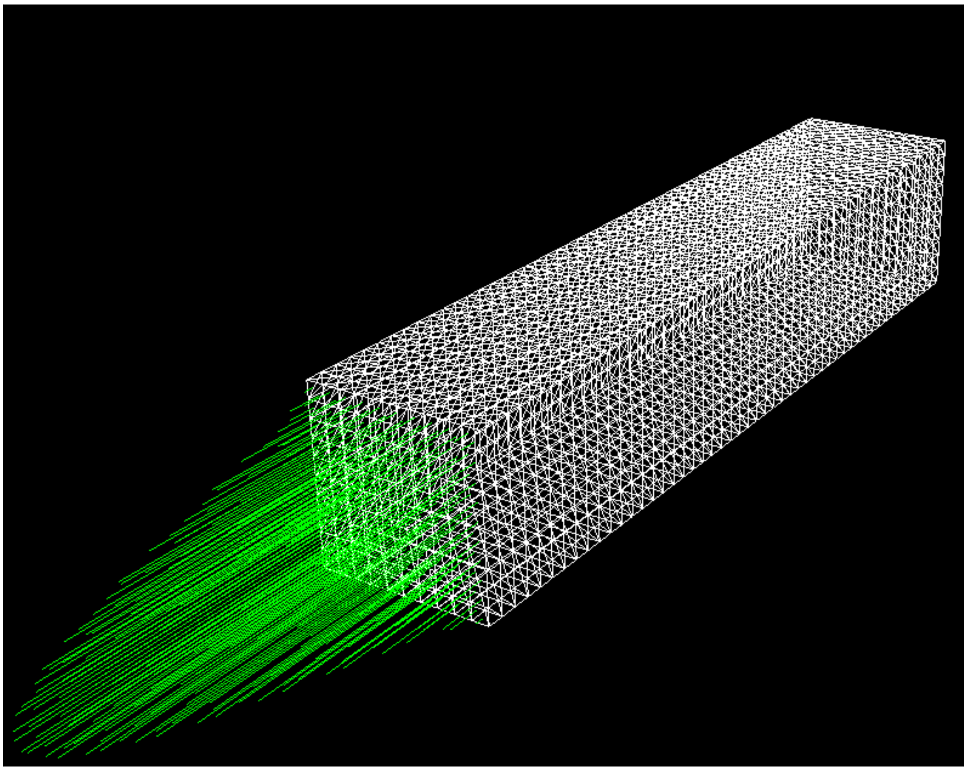
4.1 Verification of the formulation

The developed formulation has been tested for some 2D benchmark problems, and good agreement has been achieved by analytical and FEM solutions. The details of 2D verification can be found elsewhere [24]. It should be stated that, in examples given in this chapter, the length units are in μm, the time unit is s and the mass unit is kg. All units are derived from these major units, and also, since the Stoke's equation is a linear differential equation, it is possible to non-dimensionalise the respective quantities with the above mentioned units. From this perspective, all units in all figures are given in stated dimensions.

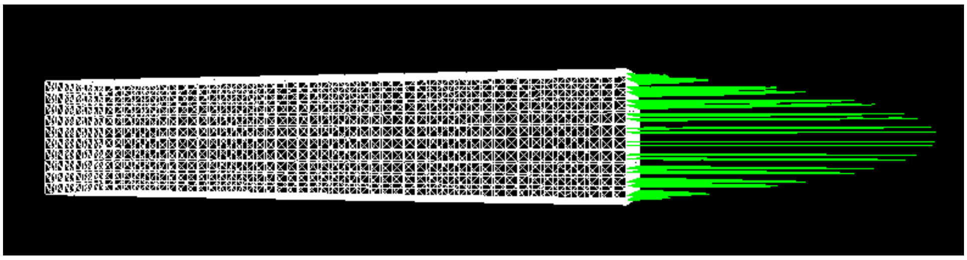
4.1.1 Flow in a square channel

For the verification of the 3D formulation, the flow in a square channel with of W and length of L is analysed as a first benchmark. At the inlet, constant velocity profile is given, and at the exit, the viscous forces are zero, leading to a no-traction condition. At the channel walls, sticking boundary conditions (zero velocity) are imposed. The flow, which has a constant velocity profile, turns to a paraboloid at each section of the square channel. The analytical solution can be obtained by using integral transform techniques as:

$$u(x,y) = \frac{16}{W^2}\frac{\Delta P}{\mu L}\sum_{m=1}^{\infty}\sum_{n=1}^{\infty}\frac{\sin(\beta_m x/W)\sin(\lambda_n y/W)}{(\beta_m^2+\lambda_n^2)\beta_m\lambda_n} \tag{4.1}$$



(a)



(b)

Figure 4.1: Two views from channel flow analysis

where β_m and λ_n's are the eigenvalues defined as:

$$\beta_m = (2m-1)\pi/W, \quad \lambda_n = (2n-1)\pi/W \tag{4.2}$$

The boundary element analysis results, presented in Figure 4.1 fits in a very good correlation to the exact data (with less than 1% error).

At this point, a short note can be added: in the case of a creeping flow, if there is a small obstacle (eg. a sphere) within the channel, the far-field solutions should not be affected. That is to say, if a spherical small particle is added to the channel, at its centroid, the flow redevelops quickly such that the solution at the exit will remain

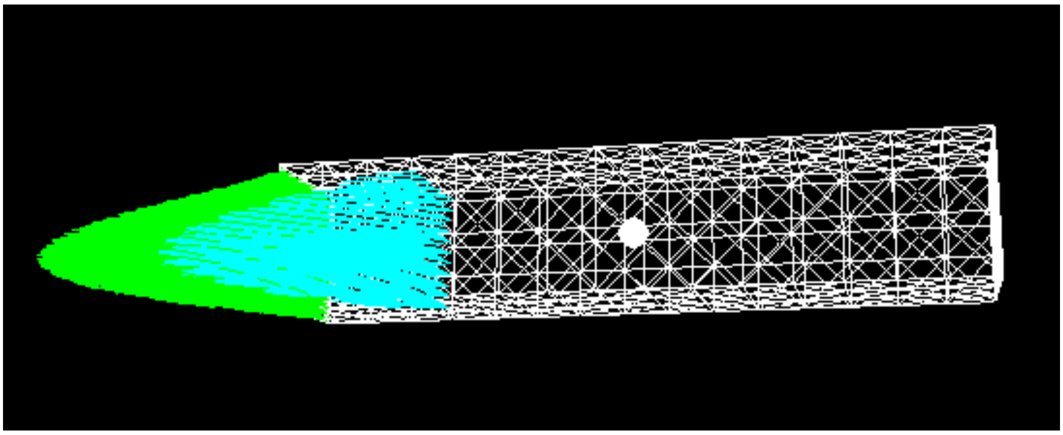


Figure 4.2: Development of the velocity profile after an obstacle in the square channel

unchanged. We demonstrate this fact in Figure 4.2.

In this figure, both exit velocities and the velocities at a vertical section within the flow domain are plotted. It can be seen that they both have parabolic shape.

4.1.2 Two sphere problem

As a second benchmark, the motion of a spherical particle over a stationary particle in an infinite medium is analysed. The schematic drawing of this problem is given in Figure 4.3.

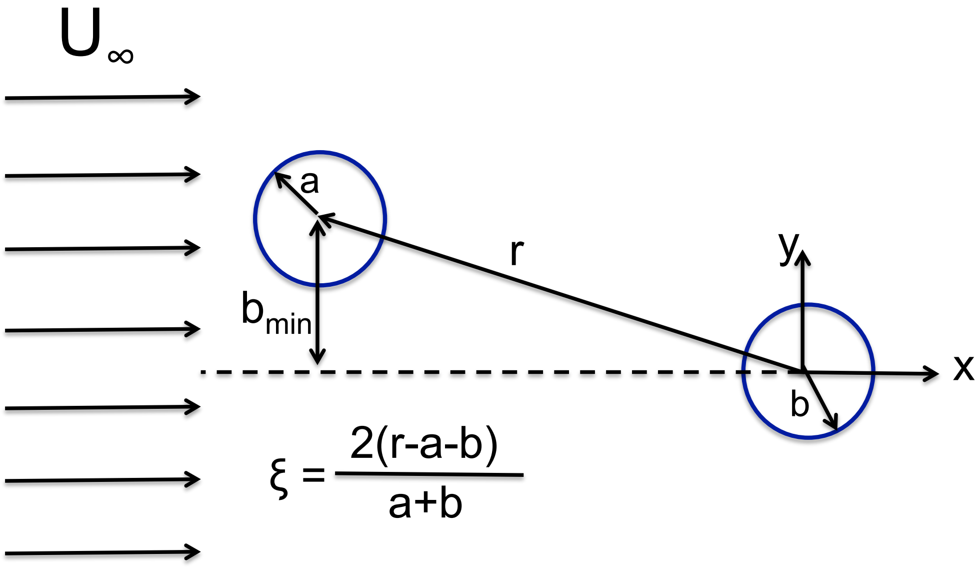


Figure 4.3: Schematic drawing of the second benchmark problem

This problem has an analytical solution for $\xi \gg 1$ (far-field solution) and for $\xi \ll 1$

(near-field solution) [49]. The comparison of the far-field is given in Figure 4.4. In the simulations 6144 elements per particle are used on the particles. As seen from the figure, BE formulation recovers the analytical solution pretty well.

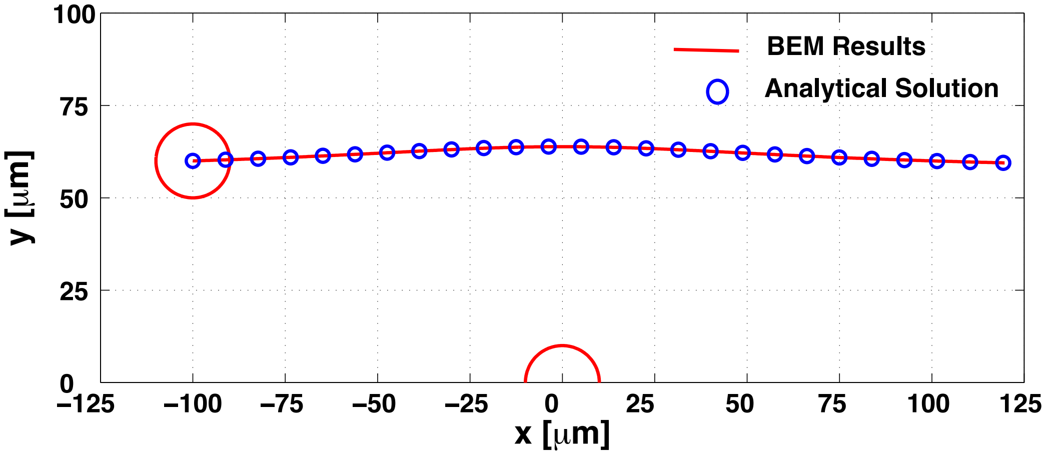
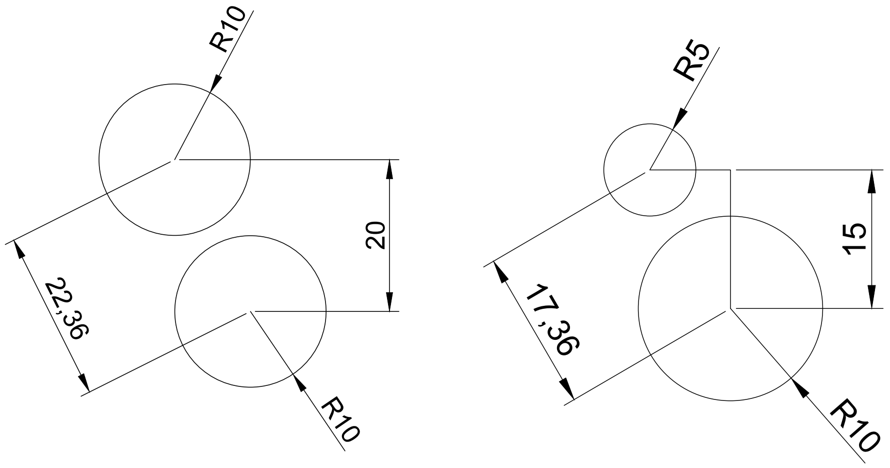


Figure 4.4: Comparison of BEM and analytical solution for far-field two sphere problem

In the near-field, however, the analytical solution is applicable only if the particle(s) are sufficiently small. From the fact that the analytical solution is symmetric, we can deduct such a conclusion: if a very small particle is moving close to the sphere, its path will be symmetrical. But if the particle size is increased, it will be affected by the velocities away from the non-moving sphere, and therefore the path will deviate from being symmetrical. This may be called, the size effect. To demonstrate this, we solve the near-field problem with two different particles: the first particle has a radius of $10\mu m$, whereas the second particle has a radius of $5\mu m$. They both start at the same distance to a non-moving sphere of $10\mu m$ radius. The drawing for the problem is given in Figure 4.5.

In Figure 4.6, the path of the center of the moving sphere is plotted. The guidelines are for referencing the symmetry along the center location of the fixed sphere. Note that, symmetry is not achieved in this case.

Decreasing the radius of the particle to $5\mu m$, the path becomes more symmetrical (Figure 4.7), as expected.



(a) (b)

Figure 4.5: Two different problems to demonstrate size effect

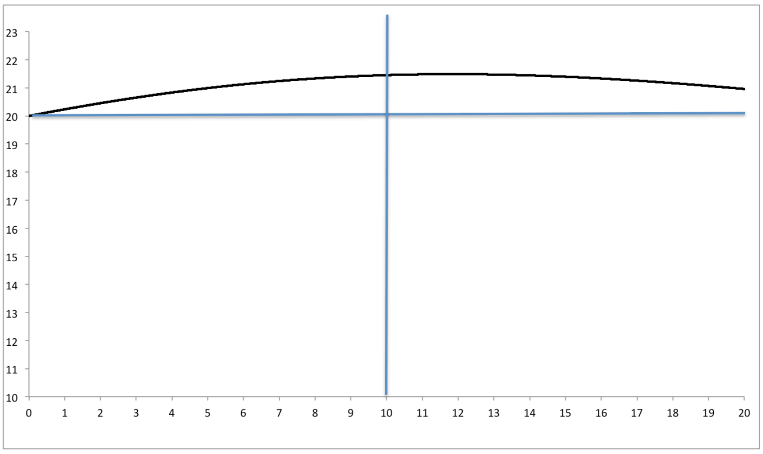


Figure 4.6: The near-field solution with BEM for large sphere

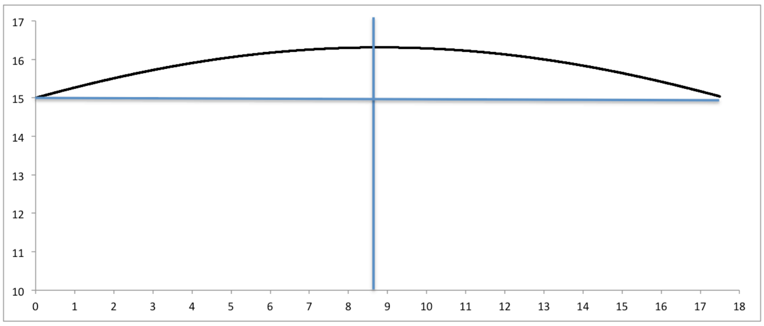


Figure 4.7: The near-field solution with BEM for small sphere

4.1.3 Three sphere problem

As a third benchmark problem, a three sphere (with identical radius) problem is considered (schematic drawing of the problem can be seen in Figure 4.8). Recently, a numerical technique has been proposed and compared against the Stokian Dynamics solution, and found that the proposed techniques are accurate up to particle separation of 0.1 radius [50]. The problem statement is simple: three spheres of same radius (a) is placed on a line with separation being $s > a$. The center sphere is forced to the right sphere with unit non-dimensional force. The output of the analysis is the velocity of the middle sphere and the velocities of the side spheres (which should be equal). The result of the study is compared with the BEM formulation presented in this study and the results are tabulated in Table 4.1 (the results from [50] was digitised for the comparison). All units in the presented table are non-dimensional. As seen from the results (Table 4.1), a very good agreement has been achieved.

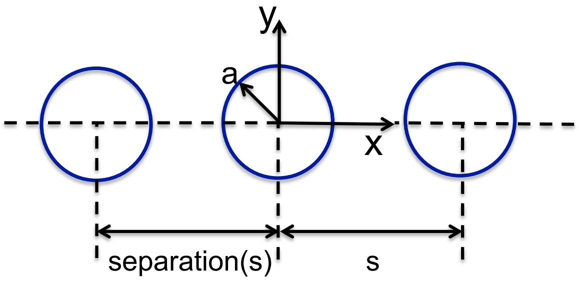


Figure 4.8: Schematic drawing of the benchmark problem

s/a	Middle Sphere			Side Sphere		
	BEM	[50]	% Error	BEM	[50]	% Error
2.01	0.666	0.659	1.08	0.644	0.645	0.21
2.05	0.703	0.694	1.28	0.629	0.630	0.10
2.10	0.733	0.727	0.89	0.615	0.614	0.10
2.20	0.780	0.775	0.69	0.589	0.589	0.01
2.40	0.847	0.838	1.02	0.548	0.547	0.13
2.60	0.886	0.878	0.89	0.513	0.512	0.25
2.80	0.913	0.906	0.80	0.483	0.483	0.06
3.00	0.933	0.925	0.90	0.457	0.456	0.15

Table 4.1: Comparison of the drag forces on the spheres for three sphere problem

4.2 Performance of the proposed algorithm

As stated above, the verification of the code is made for several problems, comparing the results with the analytical formulations or results from the previous studies. In this section, the case studies measuring the performance of the conventional method and presented algorithm are given.

For this, a very simple microchannel flow problem is defined: a channel with dimensions 100x100x500 μm with one or more particles located near to the centroid of the channel. Each particle is assumed to be spheres of $10\mu m$ radius. A representative sketch with one particle is given in Figure (4.9).

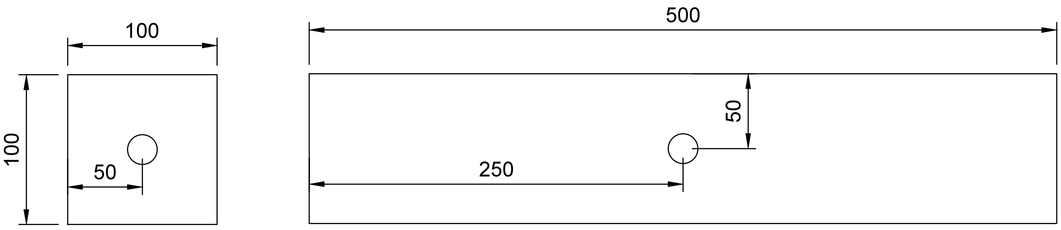


Figure 4.9: Example problem for performance analyses

The boundary conditions can be stated as follows: at the inlet, constant velocity profile with $100\mu m/s$ along the longer channel axis and zero velocity along the other two directions; at the exit no viscous forces; and on the channel walls zero velocity in all directions. From the analysis the particle velocity for single particle is found to be approximately $250\mu m/s$, which means, with a time step size of $0.001s$, the particle will move $250\mu m$ in 1000 time steps, for which the performance calculations are based on. Note that, for different problems which will have different dimensions, the number of steps to consider will differ, which will affect the following discussions.

It is clear that the performance of both methods would depend on the problem size. The main difference between the conventional method and the presented algorithm is that; in the former method the computation of LU decomposition is done on $\mathbf{G}$ and $\mathbf{H}$ matrices as a whole, but in the latter method, inverse of $\mathbf{G}_{00}$ is computed once and used in the subsequent operations.

4.2.1 Analysis 1

We begin case studies with a fixed number of elements on the particle, which is selected to be 384, and investigate the effect of changing number of elements on the channel. The number of CPU in each solution is fixed to 32 core, and the number of elements on channel is selected as 2200, 3168, 4312, 5632, 7128, 8800, 10648, and 12672.

The time to compute the results in 1000 iterations of both application, when increasing the number of elements on channel, is shown in Figure (4.10), from where it is obviously seen that proposed method gets better and better when the channel size is increased, so forth the problem size.

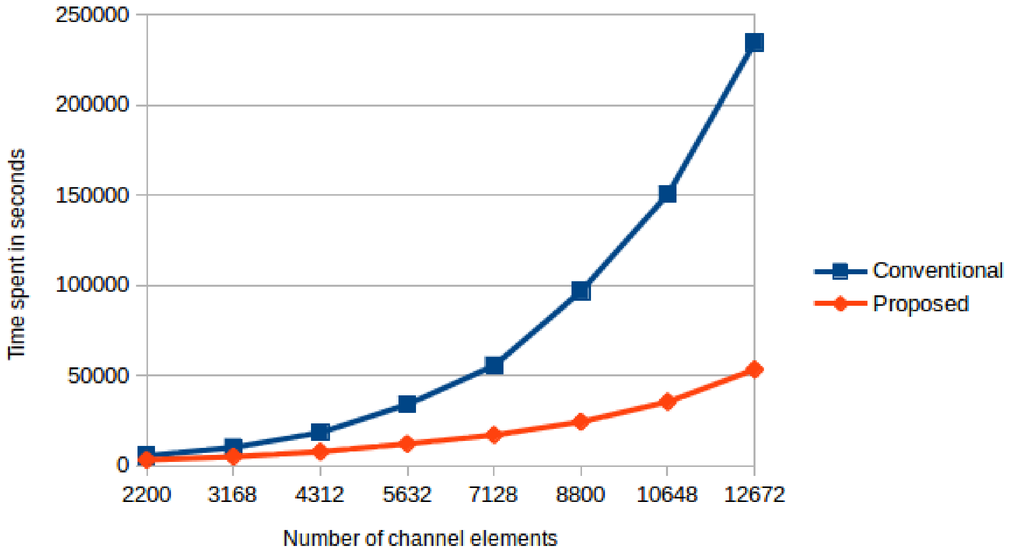


Figure 4.10: Time spent in 1000 iterations when #element on particle is fixed to 384, and #element on channel is increasing

To get the performance ratio of proposed method with respect to conventional method the formula;

$$pr = \frac{t_c}{t_p}$$

is being used, where pr represents the performance ratio, t_c is the time spent in Conventional Method, and t_p is the time spent in Proposed Method. The calculated performance ratios are sketched in Figure 4.11, from where it can be easily seen that the proposed method increases its performance when the number of elements increased on the channel. That means, when the ratio of number of elements on channel with respect to number of elements on particle increased, the proposed method gets much faster than the conventional method.

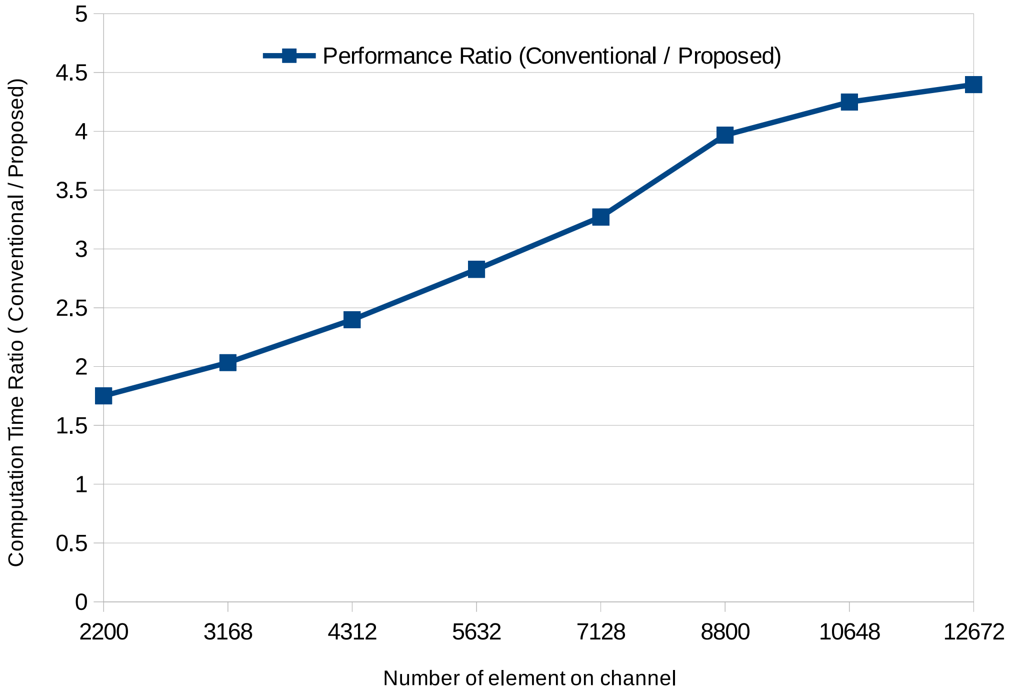


Figure 4.11: Performance ratio depending on the channel size, particle size is fixed

4.2.2 Analysis 2

A second analysis is done comparing the different numbers of particles in flow with the number of elements on the channel unchanged (Figure 4.12). Each particle is of the same size and discretised with the same number of elements. Three different channel discretisations are performed: (i) 3168, (ii) 5652, and (iii) 8800 elements on the channel. Numerical results are given in Table (4.2) where the table headings are named as;

nOfPart : number of particle
#elCh : total number of element on channel
#elParticle : total number of elements on particles
timeConven : time spend in conventional method
timeProposed : time spend in proposed method
Ratio : time in conventional / time in proposed

Table 4.2: Time spent and performance ratio

nOfPart	#elCh	#elParticle	timeConven	timeProposed	Ratio
1	3168	384	10059.264	4488.844	2.24095
2	3168	768	12850.391	8281.405	1.55172
4	3168	1536	21845.281	19271.072	1.13358
8	3168	3072	45255.747	56445.316	0.80176
16	3168	6144	127149.301	212930.510	0.59714
1	5632	384	11311.0536	31200.2420	2.75836
2	5632	768	19702.7879	38453.0132	1.95165
4	5632	1536	41996.4081	56013.6747	1.33377
8	5632	3072	106254.2890	96091.4544	0.90435
16	5632	6144	332924.0140	214216.6400	0.64344
1	8800	384	22817.1027	91541.1939	4.01196
2	8800	768	40905.0666	105616.6715	2.58199
4	8800	1536	85258.7855	130397.1705	1.52943
8	8800	3072	196739.1028	198925.9942	1.01112
16	8800	6144	623241.6868	395470.8260	0.63454

The results are also displayed in Figure 4.12 where it can be observed that, the performance of the presented algorithm depends on the ratio of number of elements on the channel to the total number of elements on the particles.

Close inspection reveals that, when this ratio is increased, the performance of the presented method increases. At this point, a short note is in place: in most particle tracking problems in microfluidic applications, the total number of elements on the channel is comparably much more than the total number of elements on the particles, which introduces a very important advantage for the presented algorithm.

4.2.3 Analysis 3

A third analysis is on the parallelisation of the procedure. For this, the problem size is fixed and the number of processors being used is changed. The number of elements on particle is selected as 768, and the number of elements on channel is fixed to 5632. For every number of processor settings, both application is executed 10 times, with the number of iterations fixed to 100. The time consumed in every execution and in each iteration is recorded and average time consumption is calculated for each iteration.

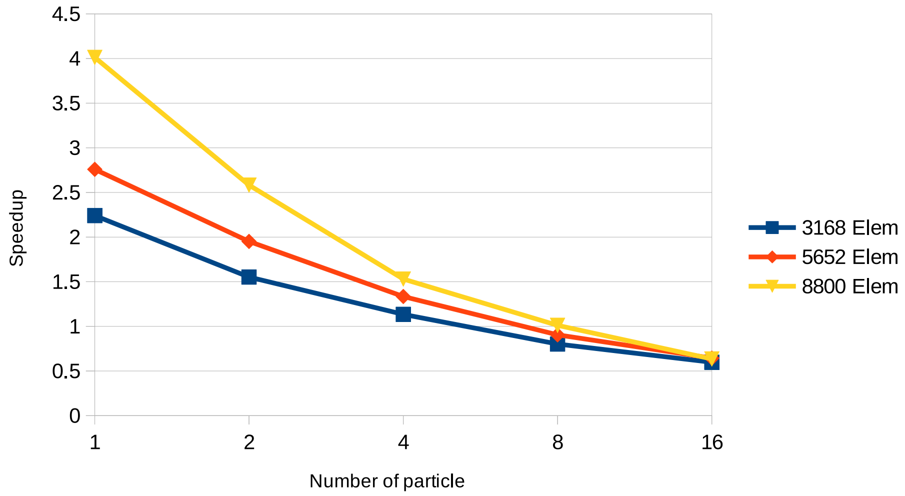


Figure 4.12: Performance ratio depending on number of particles

Since the time used in first iteration is different then the remaining iterations, its average is calculated separately. Then the results are interpolated to 1000 iterations by the formula $tt = t_{fi} + t_{ri} * 999$ (where tt states for total time, t_{fi} states for "average time consumed in first iteration", and t_{ri} represents "average time consumed in remaining iterations other than the first one").

The calculated time consumption results of both algorithms on different number of processors are given in Figure 4.13.

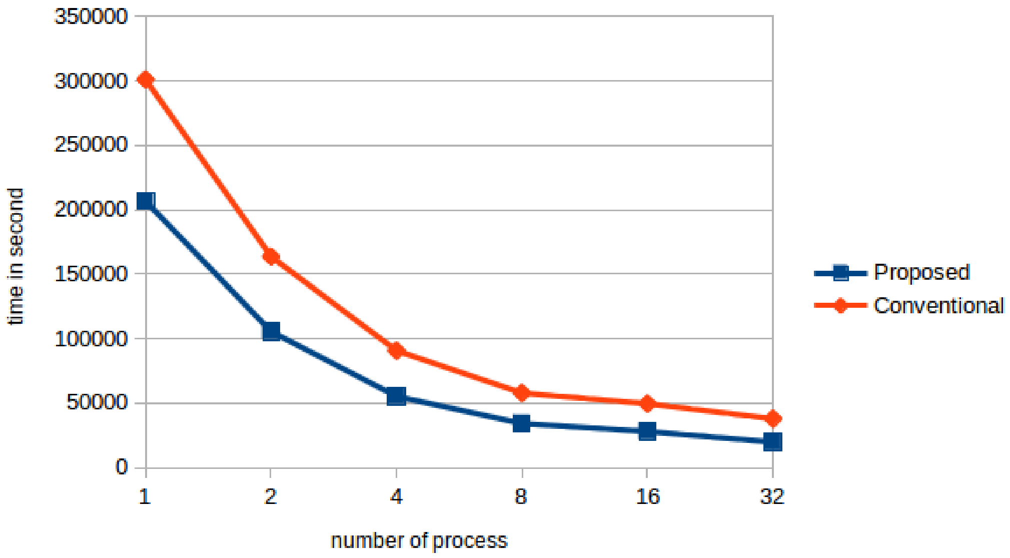


Figure 4.13: Time consumption depending on the number of processors

From the same results, the speedup ratio is calculated by the equation;

$$s = \frac{t_s}{t_p}$$

where t_s is the time spent in sequential, and t_p is the time spent in parallel algorithms. As the speedups illustrate in Figure 4.14, the proposed method shows better spedups from the conventional method, which proves itself as more parallellizable than the conventional method.

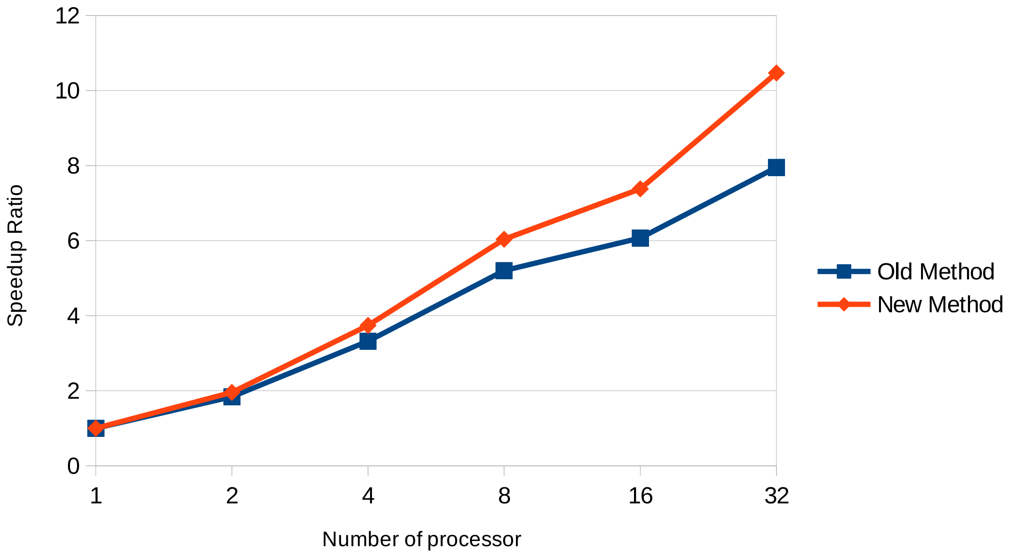


Figure 4.14: Speedup ratio depending on number of processor

4.3 Separation problem

Up to this point, the verification and performance of the proposed method is presented. Now, a very common application of microchannel flow in LOC will be shown: the hydrodynamic separation. The main objective of the hydrodynamic separation is to separate large and small particles in a particulate flow - the large particles may resemble the red blood cells in a blood sample with small particles being the bacteria, so the designed microchannel may be used to separate bacteria in a blood sample from the red blood cells.

The hydrodynamic separation is mainly achieved by geometry changes in a microchannel which alters the flow lines so that different sized particles with the same initial starting position end up at different exit locations. With the exit locations, the particles can be extracted to separate places.

In the presented example problem, separation of two spherical particles, one with $10\mu m$ and the second with $4\mu m$ diameters, will be studied. The channel geometry is given in Figure 4.15.

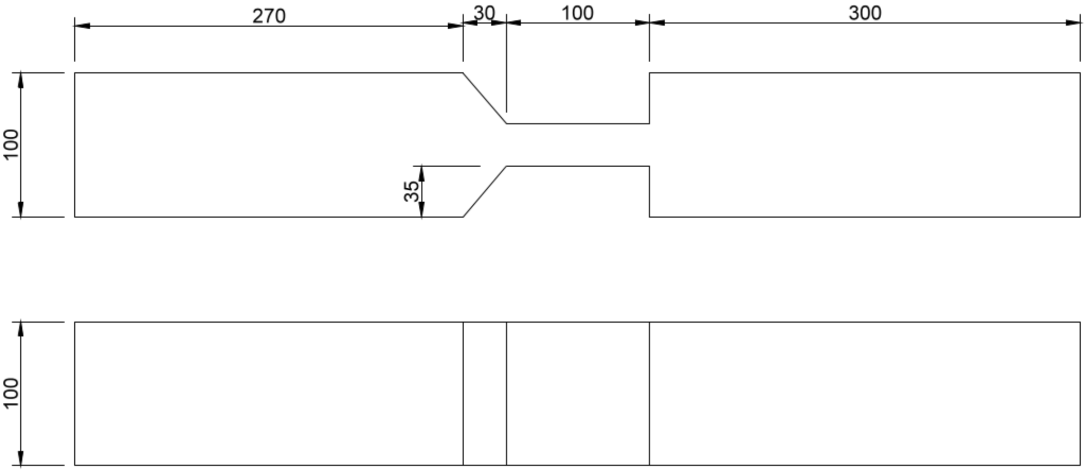


Figure 4.15: The channel geometry for separation problem

The particles are released at symmetric positions from the channel wall, with the same distance $10\mu m$ from the wall. It is obvious that, since the assumed motion of the fluid obeys Stoke's equation, regarding the distance of the particles, the effect of one particle to the other will be negligible. This is to say that, the particle trajectories could be obtained by releasing one particle at a time, then combining the analysis. In this study, however, two particles are released at the same time as displayed in Figure 4.16.

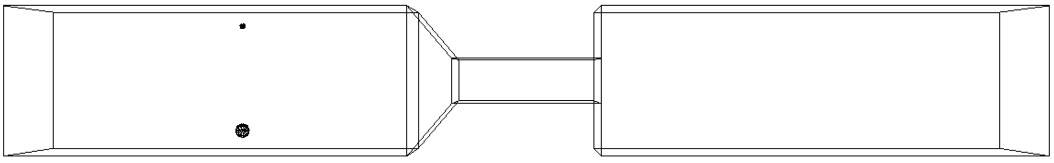


Figure 4.16: Starting position of the two particles in separation problem

The trajectory of the particles are displayed in Figure 4.17. It can be seen that, the particle with smaller diameter appears to end up at a closer location to the symmetry axis of the channel when compared with the particle with the larger diameter. This way, it can be concluded that these two particles are *separated*.

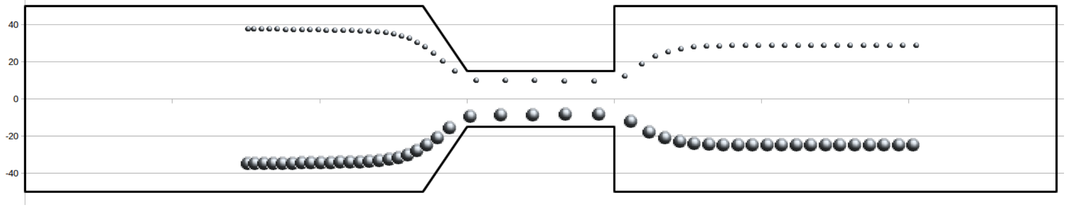


Figure 4.17: Trajectory of the two particles in separation problem

4.4 Cytometry problem

Another practical problem in microfluidic applications is to *count* the number of particles of importance. An example would be to count the number of bacteria in urine samples, revealing whether the subject is infected or not. A very practical way to achieve this is to force the particulate fluid to cross a *hurdle* (a smaller channel or an obstacle in the channel) where this cross over will be forced to be one-particle at a time. This way, using a laser beam or a marker, each particle can be counted. This is a design problem where many simulations with lots of particles should done. In this study, a simple example with 27 particles of $5\mu m$ radius each, will be presented. The particles are placed at an equal grid of 3×3 and relased within the flow. The channel geometry is the same as given in Figure 4.15.

The results of the analysis is presented in Figure 4.18. It can easily be seen that, the method simulates the flow of 27 particles in a long channel with hurdle at the middle in a sufficiently good manner. Even when the particles get very close to each other (when crossing the hurdle) it has been seen that hydrodynamic effects keep them from touching each other's boundary. It appears, on the other hand, that the selected channel geometry is not a good geometry for cytometry applications. It can be observed that two particles pass at the same vertical position at a time. This is a situation that is not in favour of counting the particles.

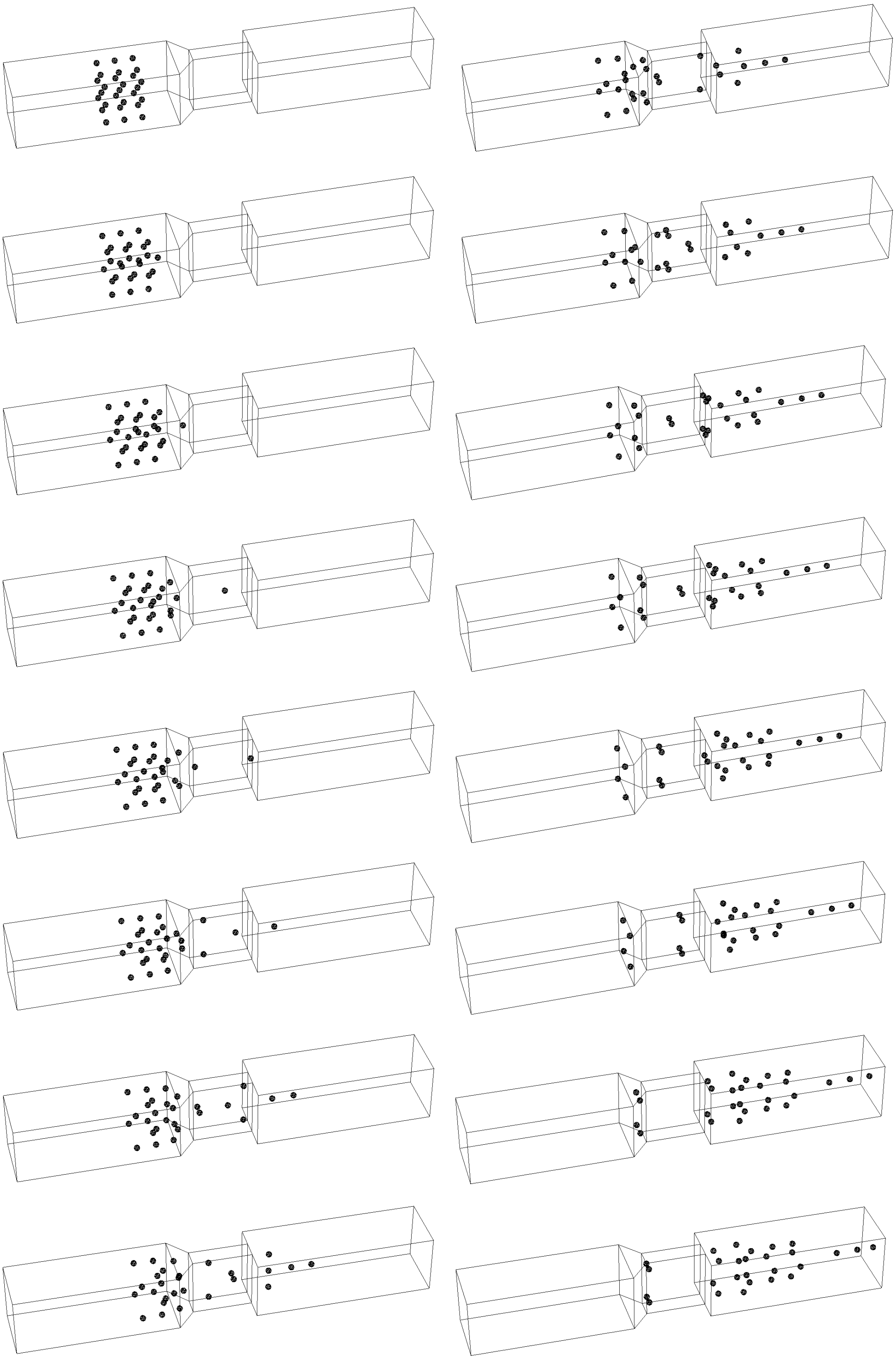


Figure 4.18: Screen captures of 27 particle moving inside the channel

CHAPTER 5

CONCLUSIONS

A new method for particle trajectory tracking is presented for Stoke's flow problems, especially applicable to the field of microchannel flows. The method depends on matrix multiplications using the system matrices of the boundary element method, resulting into a system of equations relating the particle velocities (linear and angular) to the external forces and moments applied on the particle (in case of freely moving particle(s), these forces and moments are zero). The presented method would prove its use in multiphysics applications where the external force is due to another physical problem, such as electrical, magnetic and/or acoustic field, etc.

LU factorisation with partial pivoting, although being one of the most effective methods for solving linear system of equations, is very difficult to parallelise, due to the pivoting step [46]. Incremental pivoting is proposed [46] for speeding up the LU factorisation. However, it was mentioned in the same article that, this approach may cause numerical instability.

Matrix multiplication operations, on the other hand, has been proven to have high parallel performance, especially on the new infrastructures (such as GPUs) are introduced. Also, with these new infrastructures, new algorithms that give higher speed-ups in parallel processing are being introduced [51, 52]. Accordingly, the formulation presented in this study is expected to be much more useful in the near future. Note that, the speed-up of the presented algorithm compared with the conventional solution is increasing when the number of CPU cores increase (recall Figure 4.14 for a demonstration up to 32 cores) - which is an indication of high-scalability of the proposed method. Considering the number of cores in GPUs available for floating point operations, the proposed algorithm would be much more effective.

In the presented algorithm, the matrices $\mathbf{G}_{00}$ and $\mathbf{H}_{00}$ are computed only once, and the inverse of the matrix $\mathbf{G}_{00}$ is obtained at the beginning of the process. A possible application to utilise this property of the present procedure would be as: computing the $\mathbf{G}_{00}$, $\mathbf{H}_{00}$ and $\mathbf{G}_{00}^{-1}$ in a powerful computer that has sufficient memory and storing it for use in a common computer where only the matrix multiplications are performed. Also, with the above matrices evaluated once, different problems (e.g. with different number of particles at different locations) can be solved simultaneously in different machines.

A major drawback of the presented algorithm would be the requirement of large memory space. This, however, is not a major issue when the increasing trend of computer architectures with larger memory installed. Also, with the note above in place, the static matrices $\mathbf{G}_{00}$ and $\mathbf{H}_{00}$ can be computed in a computer with sufficient memory for later use in computer(s) with less memory. Therefore, although there appears a disadvantage in memory usage of the proposed method when compared with the conventional method, this can be disregarded.

In the case study presented, it can be seen that, as the ratio of $\mathbf{G}_{00}$ over $\mathbf{G}_{PP}$ increases, the presented method gets better. When general applications of particle flow in microchannels is considered, this ratio is very high. This makes the presented algorithm a powerful tool for particle tracking problems in microfluidic applications.

Aside from the performance of the proposed method, in this study, the effectiveness of the BEM in solving particle flow problems is demonstrated using several examples. In each example, it has been seen that the BEM presents a very powerful tool for solution of Stoke's flow problems as well as particle motion - especially when the particle is close to the non-moving boundary.

5.1 Further Study

The parallelization techniques and the underlying architecture plays a very important role on the performance of both method. It is stated, in this study, that there are many parallel matrix multiplication methods in the literature. Especially GPU parallelization may be implemented to improve the performance. There are also many research on hybrid GPU-CPU matrix operations in the literature, which may also be studied for proposed method. It is also recommended for researchers to work on distributed parallel processing techniques, such as MPI.

The other important issue in solving BE problem is the amount of memory required to solve the dense matrix systems. Further research may also be concentrated on this issue, and solution such as out-of-core (OOC) could be applied.

A third, and very important future perspective is to employ deformable particles within the study. In this study all particles are assumed to be rigid; yet in real life applications, especially when blood samples or similar particulate flows are considered, the particles are deformable. The inclusion of deformable particles in the formulation would be most beneficial.

Also, the proposed formulation presents it usefulness in multiphysics applications. These applications can be listed as;

- Electric field (AC or DC) application to certain areas of the fluid medium, which would affect different particles (with different electrical properties) in different amounts resulting in different total net force on the particle.
- Magnetic field application at certain locations, affecting the total net force on the particle.
- Acoustical field application, resulting in a similar net force change on the particles
- Optical field, like laser beam application, would also affect particles with different properties in applied net force.

The present study makes it possible to impose the net external force (which might come from a different physics problem as stated above) directly into the resulting set of linear algebraic equations. This external force can be computed via a different simulation - in that case, a coupled analysis is possible. Another possible application would be the reverse engineering problems, where the external force application is not know directly, but the path of the particle may be extracted from experimental analysis. With this experimental results, the net total force on the particle due to the application of the external field can be computed using the proposed formulation.

REFERENCES

[1] L. C. Wrobel, *The Boundary Element Method, Volume 1, Applications in Thermo-Fluids and Acoustics*, vol. 1. John Wiley and Sons, Ltd., 2002.

[2] G. D. Manolis and D. E. Beskos, *Boundary element methods in elastodynamics*. Unwin-Hyman, 1988.

[3] C. Pozrikidis, *A Practical Guide to Boundary-Element Methods A practical guide to boundary-element methods with the software library BEMLIB*. Chapman and Hall, CRC, 2002.

[4] I. Benedetti and M. H. Aliabadi, "A fast hierarchical dual boundary element method for three-dimensional elastodynamic crack problems," *International Journal for Numerical Methods in Engineering*, vol. 84, pp. 1038–1067, NOV 26 2010.

[5] Y. Mengi, A. H. Tanrıkulu, and A. K. Tanrıkulu, "Boundary element method for elastic media an introduction," tech. rep., METU, September 1992.

[6] G. E. Karniadakis and A. Beskok, *Micro Flows Fundamentals and Simulatioon*. Springer, 2001.

[7] W. Jung, J. Han, J.-W. Choi, and C. H. Ahn, "Point-of-care testing (poct) diagnostic systems using microfluidic lab-on-a-chip technologies," *Microelectronic Engineering*, vol. 132, no. 0, pp. 46 – 57, 2015. Micro and Nanofabrication Breakthroughs for Electronics, {MEMS} and Life Sciences.

[8] B. Cetin, M. B. Ozer, and M. E. Solmaz, "Microfluidic bio-particle manipulation for biotechnology," *Biochem. Eng. J.*, vol. 92, pp. 63–82, 2014.

[9] B. Cetin and D. Li, "Dielectrophoresis in microfluidics technology," *Electrophoresis, Special Issue on Dielectrophoresis*, vol. 32, pp. 2410–2427, 2011.

[10] Y. Ai, S. W. Joo, Y. Jiang, X. Xuan, and S. Qian, "Pressure-driven transport of particles through a converging-diverging microchannel," *Biomicrofluidics*, vol. 3, no. 022404, pp. 1–14, 2009.

[11] Y. Ai, B. Mauroy, A. Sharma, and S. Qian, "Electrokinetic motion of a deformable particle: Dielectrophoretic effect," *Electrophoresis*, vol. 32, pp. 2282–2291, 2011.

[12] A. L. Fogelson and C. S. Peskin, "A fast numerical method for solving the three-dimensional stokes equations in the presence of suspended particle," *J. Comput. Phys.*, vol. 79, pp. 50–69, 1988.

[13] R. Glowinski, T. W. Pan, T. I. Hesla, D. D. Joseph, and J. Periaux, "A fictitious domain approach to the direct numerical simulation of incompressible viscous flow past moving rigid bodies: Application to particulate flow," *J. Comput. Phys.*, vol. 169, pp. 363–426, 2001.

[14] D. L. House and H. Luo, "Electrophoretic mobility of a colloidal cylinder between two parallel walls," *Engineering Analysis with Boundary Elements*, vol. 34, pp. 471–476, 2010.

[15] D. L. House and H. Luo, "Effect of direct current dielectrophoresis on the trajectory of a non- conducting colloidal sphere in a bent pore," *Electrophoresis*, vol. 32, pp. 3277–3285, 2011.

[16] G. K. Youngreen and A. Acrivos, "Stokes flow past a particle of arbitrary shape: a numerical method of solutions," *J. Fluid Mechanics*, vol. 69, pp. 377–403, 1975.

[17] G. K. Youngreen and A. Acrivos, "On the shape of a gas bubble in a viscous extensional flow," *J. Fluid Mechanics*, vol. 76, pp. 433–442, 1976.

[18] J. Rallison and A. Acrivos, "A numerical study of the deformation and burst of a viscous drop in an extensional flow," *J. Fluid Mechanics*, vol. 89, pp. 191–200, 1978.

[19] C. Pozrikidis, "A spectral-element method for particulate stokes flow," *J. Comput. Phys.*, vol. 156, pp. 360–381, 1999.

[20] D. Womble, D. Greenberg, S. Wheat, R. Benner, M. Ingber, G. Henry, and S. Gutpa, "Application of Boundary Element Methods on the Intel Paragon," in *Supercomputing '94, Proceedings*, Supercomputing Proceedings, (10662 Los Vaqueros Circle, Los Alamitos, Ca 90720), pp. 680–684, IEEE, Comp Soc; Assoc Comp Machinery; Soc Ind & Appl Math, IEEE, Computer Soc Press, 1994. Supercomputing 94, Washington, DC, Nov 14-18, 1994.

[21] S. Weinbaum, P. Ganatos, and Z. Yan, "Numerical Multipole and Boundary Integral-Equation Techniques in Stokes-flow," *Annual Reiew of Fluid Mechanics*, vol. 22, pp. 275–316, 1990.

[22] J. Gomez and H. Power, "A multipole direct and indirect BEM for 2D cavity flow at low Reynolds number," *Enginering Analysis with Boundary Elements*, vol. 19, pp. 17–31, Jan 1997.

[23] J. Gomez and H. Power, "A parallel multipolar indirect boundary element method for the Neumann interior Stokes flow problem," *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 523–543, JUN 10 2000.

[24] B. Baranoglu and B. Cetin, “A particle flow specific boundary element formulation for microfluidic applications,” in *4^{th} Micro and Nano Flow Conference, 6–10 September 2014* (T. Karayiannis, C. S. Konig, and S. Balabani, eds.), no. 206, Brunel University, 2014.

[25] S. Chaillat and M. Bonnet, “Recent advances on the fast multipole accelerated boundary element method for 3D time-harmonic elastodynamics,” *WAVE MOTION*, vol. 50, pp. 1090–1104, NOV 2013.

[26] H. Wu, Y. Liu, and W. Jiang, “A low-frequency fast multipole boundary element method based on analytical integration of the hypersingular integral for 3D acoustic problems,” *Enginerring Analysis with Boundary Elements*, vol. 37, pp. 309–318, FEB 2013.

[27] Z. Chen, H. Xiao, X. Yang, and C. Su, “Taylor series multipole boundary element-mathematical programming method for 3D multi-bodies elastic contact problems,” *International Journal for Numerical Methods in Engineering*, vol. 83, pp. 135–173, JUL 9 2010.

[28] L. Ying, G. Biros, and D. Zorin, “A kernel-independent adaptive fast multipole algorithm in two and three dimensions,” *Journal of Computational Physics*, vol. 196, no. 2, pp. 591 – 626, 2004.

[29] H. Argeso and Y. Mengi, “A frequency domain boundary element formulation for dynamic interaction problems in poroviscoelastic media,” *Computational Mechanics*, vol. 1, pp. 1–3, 2013.

[30] O. F. Yalcin and Y. Mengi, “A new boundary element formulation for wave load analysis,” *Computational Mechanics*, vol. 52, pp. 815–826, 2013.

[31] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, “Basic linear algebra subprograms for fortran usage,” *ACM Trans. Math. Soft.*, vol. 5, pp. 308–323, Sept. 1979.

[32] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson, “An extended set of fortran basic linear algebra subprograms.,” *ACM Trans. Math. Soft.*, vol. 14, pp. 1–17, March 1998.

[33] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duf, “A set of level 3 basic linear algebra subprograms,” *ACM Trans. Math. Soft.*, vol. 16, pp. 1–17, March 1990.

[34] R. C. Whaley and J. J. Dongarra, “Automatically tuned linear algebra software.,” in *Proceedings of SC 98.*, 1998.

[35] Intel, “Math kernel library.” https://software.intel.com/en-us/intel-mkl, December 2014.

[36] AMD, "Amd core math library." http://developer.amd.com/tools-and-sdks/cpu-development/amd-core-math-library-acml/, December 2014.

[37] IBM, "Engineering and scientific subroutine library." http://www.ibm.com/systems/power/software/essl/, December 2014.

[38] OpenBLAS, "Optimized blas library based on gotoblas2." http://www.openblas.net/, December 2014.

[39] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. D. Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' guide (3rd ed.)*. SIAM, January 1999.

[40] libFLAME, "High performance dense linaer algebra library." http://www.cs.utexas.edu/ flame/web/libFLAME.html, December 2014.

[41] F. Van Zee, E. Chan, R. van de Geijn, E. Quintana, and G. Quintana-Orti, "Introducing: The libflame library for dense matrix computations," *Computing in Science Engineering*, vol. PP, no. 99, pp. 1–1, 2009.

[42] MPI, "Message passing interface." http://www.mpi-forum.org/, December 2014.

[43] A. Raltson and A. P.Rabinowitz, *A first course in Numerical Analysis*. McGraw-Hill, 1978.

[44] C. Gerald and P.O.Wheatley, *Applied Numerical Analysis*. Addison-Wesley, 1989.

[45] G. Engeln-Müllges and F. Uhlig, *Numerical Algorithms with C*. Springer, January 2014.

[46] E. Chan, R. van de Geijn, and A. Chapman, "Managing the complexity of lookahead for lu factorization with pivoting.," in *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, pp. 200–208, 2010.

[47] M. Kowarschik and C. Weiß, "An overview of cache optimization techniques and cache-aware numerical algorithms," in *Algorithms for Memory Hierarchies* (U. Meyer, P. Sanders, and J. Sibeyn, eds.), vol. 2625 of *Lecture Notes in Computer Science*, pp. 213–232, Springer Berlin Heidelberg, 2003.

[48] OpenMP, "Openmp specification for parallel programming." http://openmp.org/, December 2014.

[49] S. R. Risbud and G. Drazer, "Trajectory and distribution of suspended non-brownian particles moving past a fixed spherical or cylindrical obstacle," *J. Fluid Mech.*, vol. 714, pp. 213–237, 2013.

[50] H. J. Wilson, "Stokes flow past three sphere," *J. Comp. Phys.*, vol. 245, pp. 302–316, 2013.

[51] Z. A. Alqadi, M. Aqel, and I. M. M. E. Emary, "Performance analysis and evaluation of parallel matrix multiplication algorithms," *World Applied Sciences Journal*, vol. 5, no. 2, pp. 211–214, 2008.

[52] T. M. Smith, R. v. d. Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. V. Zee, "Anatomy of high-performance many-threaded matrix multiplication," in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, (Washington, DC, USA), pp. 1049–1059, IEEE Computer Society, 2014.

APPENDIX A

APPENDIX

The fundamental solutions of the Stokes' flow are given as (adopted from [1]):

$$u_{ij}^{*}(\mathbf{A},\mathbf{P}) = \frac{1}{8\pi\mu r}\left(\delta_{ij} + r_i r_j\right) \tag{A.1}$$

$$t_{ij}^{*}(\mathbf{A},\mathbf{P}) = \frac{3 r_i r_j}{4\pi r^2}\frac{\partial r}{\partial n} \tag{A.2}$$

$$p_{i}^{*}(\mathbf{A},\mathbf{P}) = \frac{r_j}{4\pi r^2} \tag{A.3}$$

$$q_{i}^{*}(\mathbf{A},\mathbf{P}) = \frac{1}{2\pi r^3}\left[n_j - 3 r_j \frac{\partial r}{\partial n}\right] \tag{A.4}$$

for 3D. Also 2D fundamental solutions are given below for completeness:

$$u_{ij}^{*}(\mathbf{A},\mathbf{P}) = \frac{1}{4\pi\mu}\left[\ln(\frac{1}{r})\delta_{ij} + r_i r_j\right] \tag{A.5}$$

$$t_{ij}^{*}(\mathbf{A},\mathbf{P}) = \frac{r_i r_j}{\pi r}\frac{\partial r}{\partial n} \tag{A.6}$$

$$p_{j}^{*}(\mathbf{A},\mathbf{P}) = \frac{r_j}{2\pi r} \tag{A.7}$$

$$q_j^*(\mathbf{A},\mathbf{P}) = \frac{1}{\pi r^2}\left[n_j - 2r_j\frac{\partial r}{\partial n}\right] \tag{A.8}$$

All these equations refer to the definitions below (with Fig-A.1)

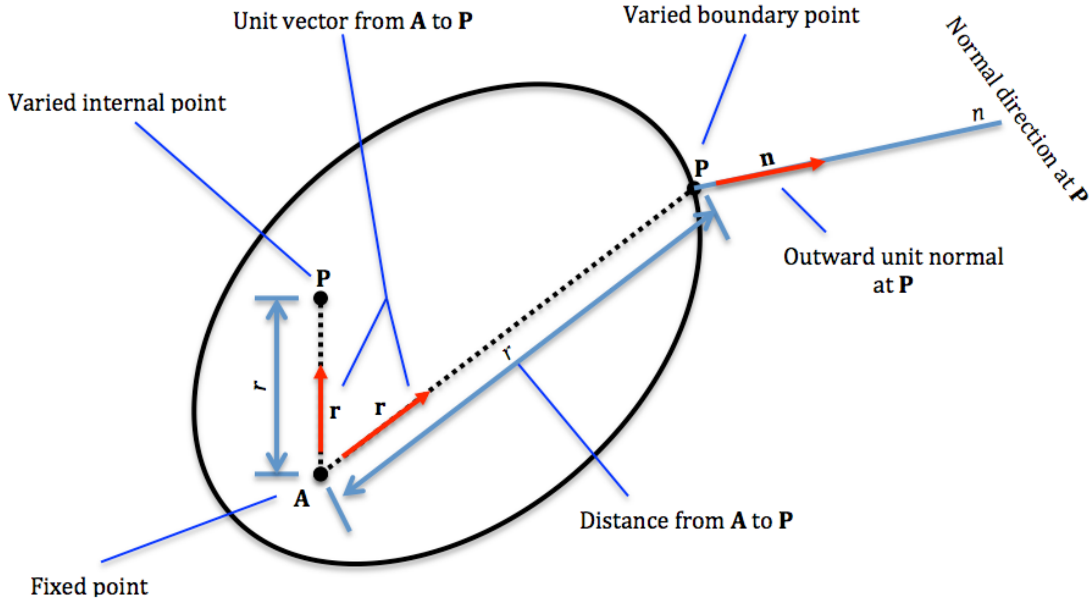

Unit vector from **A** to **P**
Varied boundary point
Normal direction at **P**
n
Varied internal point
P
P
n
Outward unit normal at **P**
r
r
r
r
A
Distance from **A** to **P**
Fixed point

Figure A.1: Development of the velocity profile after an obstacle in the square channel

$\mathbf{A}$	Fixed point
$\mathbf{P}$	Varied point
r	Distance between the fixed point and the varied point
$\mathbf{r}$	The unit vector in the $\overrightarrow{\mathbf{AP}}$ direction
r_i	Components of the vector $\mathbf{r}$
$\mathbf{n}$	The unit normal vector at the point $\mathbf{P}$ when the point is on the boundary
n_i	Components of the vector $\mathbf{n}$
$\frac{\partial r}{\partial n}$	The directional derivative of the position function $\mathbf{r}(\mathbf{A},\mathbf{P})$ in the normal direction at the point $\mathbf{P}$

With these definitions, the following identities are in place:

$$r_i = \frac{\partial r}{\partial x_i} \tag{A.9}$$

$$\frac{\partial r}{\partial n} = \mathbf{r} \cdot \mathbf{n} = r_i n_i \tag{A.10}$$

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Karakaya, Ziya
Nationality: Turkish (TC)
Date and Place of Birth: 07.04.1965, Sivas
Marital Status: Married
Phone: 0 312 5868345
Fax: 0 312 5869000

EDUCATION

Degree	Institution	Year of Graduation
M.S.	METU-CEIT	2001
B.S.	METU Department of Mathematics	1992
High School	Çetinkaya Lisesi	1982

PROFESSIONAL EXPERIENCE

Enrollment	Place	Year
Student Advisor	METU Computer Center, Ankara	1987-1988
High Level Student Advisor	METU Computer Center, Ankara	1988-1992
Part Time Specialist	The Scientific and Technical Research Council of Turkey, Ankara	1991-1992
Part Time Specialist	METU, Printing House, Ankara	1991-1992
Teaching Assistant	METU, Dept. of Mathematics, Ankara	1992-1998
System Manager	METU, Faculty of Art and Science, Ankara	1996-1998
Advisor to the President	The Scientific and Technical Research Council of Turkey, Ankara	1994-1998
Work Group Member	Relationship with Turkish Republics in IT, Prime Ministry, Ankara	1998-1999
IT Projects Coordinator	The Scientific and Technical Research Foundation, Ankara	1998-2001
Part Time Instructor	METU, Department of Mathematics	1998-2003
Part Time Instructor	Department of Computer Engineering, Cankaya University, Ankara	2000-2001
Director	Atılım University, Computer Center	2003-2008
Yön. Kur. Üyesi	Türkiye Bilişim Derneği, Kamu Bilgi İşlem Yöneticileri Birliği	2004-2009
Instructor	Dept. of Computer Engineering, Atılım University, Ankara	2001-Cont

PUBLICATIONS

1. Ziya Karakaya, “Development and Implementation of an On-line Exam for a Programming Language Course”, Ms. Thesis, METU-CEIT, September 2001.
2. Ziya Karakaya, “Design, Development and Implementation of On-Line Exam System”, International Open and Distance Education Symposium, Anadolu University, Eskişehir, May 2002. (in Turkish)

3. Ziya Karakaya, Ali Yazıcı, “Developing Sharable Contents for Online Systems”, Academic Informatics Conference, Çukurova University, February 2003. (in Turkish)

4. Ziya Karakaya, “Data Interchange Between Agencies in e-Government”, Informatics Congress, Istanbul, May 2003. (in Turkish)

5. Ziya Karakaya, et al. “Principles of Data Interchange Between Agencies in e-Government”, e-Government Report, TBD 2003. (in Turkish)

6. Ziya Karakaya, “Internet/Web Based Distance Education Systems and the Future”, Informatics Congress, Istanbul, May 2003. (in Turkish)

7. A. Yazıcı, Ziya Karakaya, Irfan Altas & Barney Dalgarno, “Changing Our Educational Institutions: Transition From Traditional to E-Learning Programs”, Submitted to ITHET 2004, 5.th Int. Conf. on Information Technology Based Higher Education and Training, Boğaziçi University, Istanbul, Turkey, May 31-Jun 2 2004.

8. A.Yazıcı, Irfan Altas, Ziya Karakaya & Barney Dalgarno, “Transition from Traditional to e-Learning Programs: A Dynamic of Changing our Educational Institutions“, Proceedings of the 4th Int. Conference on Knowledge, Culture, and Change in Organizations, University of Greenwich, London, 2-4 August, 2004.

SCI & SSCI Papers

1. A.Yazıcı & Z. Karakaya, “Normalizing Database Schemas using Mathematica”, Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, Proceedings, Part II, LNCS 3992, Eds. Vassil N. Alexandrov, G. Dick van Albada, Peter M. A. Sloot, Jack Dongarra, pp.375-382, ISBN:3-540-34381-4, 2006

Reviewed Papers

1. A. Yazıcı & Z. Karakaya, “JMathNorm: A Database Normalization Tool using Mathematica”, Proceedings, Computational Science - ICCS 2007, Lecture Notes in Computer Science (LNCS), Eds. Shi, Y., van Albada, D., Sloot, P.M.A, and Dongarra, J., Springer-Verlag, Vol.4488, pp., 185-192, 2007.

BOOKS

1. Dr. Badrul H. Khan, Ziya Karakaya, “Web Based Learning, Measurement and Evaluations”, Printed by TBD, May 2003, ISBN : 975-96888-9-1. (in Turkish)