PhD-FSTM-2020-082
The Faculty of Science, Technology and Medicine

DISSERTATION

Presented on 18/12/2020 in Esch-Sur-Alzette

to obtain the degree of

DOCTEUR DE L’UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Abdoul Wahid MAINASSARA CHEKARAOU
Born on 03 June 1991 in Niamey (Niger)

LARGE SCALE PARALLEL SIMULATION FOR THE EXTENDED DISCRETE ELEMENT METHOD (XDEM)

Dissertation defense committee:

Chairman: Prof. Dr. Pascal BOUVRY
University of Luxembourg, Luxembourg

Vice-Chairman: Prof. Dr. Miriam MEHL
IPVS, University of Stuttgart, Germany

Jury Member: Dr. Emmanuel JEANNOT
LaBRI, INRIA, France

Jury Member: Dr. Sebastien VARRETTE
University of Luxembourg, Luxembourg

Ph.D. Supervisor: Prof. Dr. Ing. Bernhard PETERS
University of Luxembourg, Luxembourg

Ph.D. Advisor: Dr. Xavier BESSERON
University of Luxembourg, Luxembourg

DECLARATION OF AUTHORSHIP

I, Abdoul Wahid MAINASSARA CHEKARAOU, declare that this thesis titled, “Large Scale Parallel Simulation for Extend Discrete Element Method (XDEM)” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at University of Luxembourg.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Fifty years ago Kurt Gödel... proved that the world of pure mathematics is inexhaustible. No finite set of axioms and rules of inference can ever encompass the whole of mathematics. Given any finite set of axioms, we can find meaningful mathematical questions which the axioms leave unanswered. This discovery... came at first as an unwelcome shock to many mathematicians. It destroyed... the hope that they could solve the problem of deciding by a systematic procedure the truth or falsehood of any mathematical statement. ...Gödel's theorem, in denying ...the possibility of a universal algorithm to settle all questions, gave... instead, a guarantee that mathematics can never die. ...there will always be, thanks to Gödel, fresh questions to ask and fresh ideas to discover.."

Freeman Dyson, Infinite in All Directions (1988)

ABSTRACT

Abdoul Wahid MAINASSARA CHEKARAOU

Large Scale Parallel Simulation for Extend Discrete Element
Method (XDEM)

Numerical models are commonly used to simulate or model physical processes such as weather forecasts, fluid action, rocket trajectory, building designs, or biomass combustion. These simulations are immensely complex and require a hefty amount of time and computation, making it impossible to run on a standard modern laptop in a reasonable and fair period. This research work targets large scale and parallel simulations of DEM and DEM-CFD couplings using high-performance computing techniques and optimizations. This thesis aims to analyze, contribute, and apply the DEM approach using the XDEM multi-Physics toolbox to physical processes that have been reluctant to be used due to their required computational resources and time.

The first step of this work is to analyze and investigate the performance bottlenecks of the XDEM software. Therefore, the latter has been profiled, and some critical parts as the contact detection were identified as the main bottlenecks of the software. A benchmark has also been set up to assess each bottleneck part's performance using a baseline case. This step is crucial as it defines the general guidelines to follow in optimizing any application in general.

A complete framework has been developed from scratch and aims to test and compare several contact detection algorithms and implementations. The framework, which also has a parallel version, has been used to select an appropriate algorithm

and implementation for the XDEM software. The link-cell approach, combined with a new Verlet list concept, proved to be the best option for significantly reducing the contact detection part's computational time. The Verlet buffer concept developed during this thesis takes the particle flow regime into account when selecting the skin margin to enhance the algorithm's efficiency further.

In order to target the high-performance computers for large-scale simulations, a full hybrid *distributed-shared memory* parallelization has been introduced by adding a *fine-grain OpenMP* implementation layer to the existing *MPI* approach. A shared memory parallelization allows taking full advantage of personal workstations with modern CPU architecture. On the other hand, a hybrid approach is one of the best ways to fully exploit the computing nodes capacities of our modern CPU clusters that mainly have a *NUMA* architecture. Macro-benchmarking performance analysis showed that we could entirely exploit 80% (speed-up) of 85 computing nodes representing 2380 cores on the ULHPC supercomputer.

Finally, a life-size biomass combustion furnace is developed and used as an application test to demonstrate the complex and heavy cases that the XDEM software can accommodate at this time. The furnace is the combustion chamber of a 16 MW geothermal steam super-heater, part of the Enel Green Power "Cornia 2." power plant located in Italy. It proves that DEM, in general, and XDEM in particular, can be used for real-case applications that discourage users due to their complexity and especially the time required to deliver the outcome results.

ACKNOWLEDGEMENTS

Foremost, I would like to express my deep gratefulness to my supervisor Prof. Bernhard PETERS for his continuous support of my Ph.D study and research.

Besides my supervisor, I would like to thank Dr Xavier BESSERON and Alban ROUSSET for their motivation, enthusiasm and immense knowledge. Their guidance helped me in all the time of research and writing of the thesis. I could have not imagined better advisors and mentors for my Ph.D study.

My sincere thanks goes also to all my colleagues at the LUXDEM team.

Last but not least, I would like to thank my family and specially my parents for all their support since day one. I could have not achieve all of this without their unconditional love, support and advise.

CONTENTS

Declaration of Authorship **3**

Abstract **7**

Acknowledgements **9**

Contents **11**

List of Figures **17**

List of Tables **25**

List of Abbreviations **27**

List of Symbols **29**

1 Introduction **1**

1.1 Granular materials modeling and particle simulations 2
1.1.1 Granular materials 2
1.1.2 Modeling Methods 3
1.1.2.1 Continuum approach 3
1.1.2.2 Discrete approach 4
1.1.3 Particle simulations 5
1.1.4 The eXtended Discrete Element Method(XDEM) 6
1.2 High performance computing 10
1.3 DEM software: A state of art 12
1.4 Motivation and Objectives 15
1.5 Contributions 15
1.6 Structure of the thesis 16

I Methodology 19

2 Performance assessments 21

2.1 Introduction 22
2.2 Baseline 23
2.2.1 XDEM version 24
2.2.2 Metrics of performance 24
2.2.2.1 Characteristics of performance metrics 25
2.2.2.2 Type of performance metrics 25
2.2.3 Real test case 27
2.3 Profiling 27
2.3.1 Tracing profilers 28
2.3.2 Statistical profilers 29
2.4 Benchmarking 30
2.4.1 Macro and micro benchmarks 30
2.4.2 Evaluation criteria 31
2.5 Performance models 31
2.5.1 Roofline model 31
2.5.2 Execution-Cache-Memory 34
2.5.2.1 In-core model 34
2.5.2.2 Data transfer model 35
2.6 Application 36
2.6.1 Hotspots analysis 36
2.6.2 Memory footprint analysis 39
2.6.2.1 Heap and stack profiling 39
2.6.2.2 Memory leaks detection and cache profiling 41
2.7 Summary 43

3 Contact Detection in the eXtended Discrete Method (XDEM) 45

3.1 Abstract 46
3.2 Introduction 47
3.3 Broad-phase collision detection 49
3.3.1 Axis-Aligned Bounding Boxes (AABBs) 50

3.3.2 Oriented Bounding Boxes (OBBs) 51
3.3.3 Bounding Spheres 51
3.4 Broad-Phase algorithms 52
3.4.1 Spatial partitioning 53
3.4.2 The Brute Force approach 54
3.4.3 The Octree and k-d tree 54
3.4.4 Loose Octree 56
3.4.5 Grid-based spatial partitioning algorithms 58
3.4.6 Sweep and prune 60
3.4.7 Bounding volume hierarchy (BVH) 60
3.4.8 Framework and C++ library for collision 61
3.5 Benchmark 62
3.5.1 Sequential runs 62
3.5.2 Parallel runs 65
3.5.3 Adaptive approach 67
3.6 Conclusion 68

4 Local Verlet buffer approach for broad-phase interaction in DEM 71

4.1 Abstract 72
4.2 Introduction 73
4.3 Related work 75
4.4 Background 77
4.4.1 XDEM flow chart 78
4.4.2 Collision detection in XDEM 79
4.5 Local Verlet buffer approach 82
4.5.1 The local skin parameter 85
4.5.2 Automatic update and validity of the Verlet list 87
4.6 Performance Evaluation 90
4.6.1 Methodology 90
4.6.2 Test-cases 91
4.6.3 Experimental settings 96
4.6.4 Results 97

4.7 Conclusion 101

5 Predicting near-optimal skin distance in Verlet buffer for DEM 105

5.1 Abstract 106
5.2 Introduction 107
5.3 Background 109
    5.3.1 The Extended Discrete Element Method 109
    5.3.2 Verlet buffer method for XDEM 110
    5.3.3 Dakota Software Package 112
5.4 Skin distance optimisation problem 114
    5.4.1 Definition 114
    5.4.2 Evolution optimisation of the skin distance parameter 115
5.5 Near-optimal skin distance characterisation 117
    5.5.1 Design of experiments 117
    5.5.2 Methodology 118
    5.5.3 Results 121
5.6 Parallel Execution of DAKOTA SOFTWARE on HPC cluster 124
5.7 Conclusion 127

6 Hybrid MPI+OpenMP Implementation of XDEM 129

6.1 Abstract 130
6.2 Introduction 131
6.3 Challenges and implementation for the OpenMP parallelization of XDEM 133
    6.3.1 Data structures and concurrent accesses 134
    6.3.2 Memory allocation 138
6.4 Experimental Results 140
    6.4.1 Case for experimental evaluation 140
    6.4.2 Experimental settings 140
    6.4.3 Impact of data structures 140
    6.4.4 Impact of memory allocator 142
    6.4.5 OpenMP and MPI scalability 142
    6.4.6 Hybrid execution 145

6.5 Conlusion 146

II Performance analysis and Application 149

7 Performance analysis 151

7.1 Introduction 152

7.2 New vs old XDEM versions performance 152

7.2.1 Biomass furnace combustion 153

7.2.2 Sequential performance 154

7.2.3 Parallel performance 157

7.3 Large scale XDEM-OpenFOAM coupling performance 162

7.3.1 Dam break test case 162

7.3.1.1 Configuration 162

7.3.1.2 Parallel set-up 166

7.3.2 Strong scalability and Speed Up 167

7.4 Conclusion 174

8 Numerical Analysis of a Grate Firing Combustion Process 177

8.1 Abstract 178

8.2 Introduction 179

8.3 Numerical model and simulation conditions 180

8.4 Results and performance analysis 187

8.5 Conclusion 192

III Conclusion 195

9 Conclusion 197

9.1 Summary 197

9.2 Future tasks 200

IV Bibliography 203

Bibliography 205

LIST OF FIGURES

1.1 Examples of granular materials in industry: Tablets pills medicine, Silo for storing corn, and Crusher with conveyor 2
1.2 Common approaches in computational mechanics. 5
1.3 Discrete element method loop's scheme. 7
1.4 Classic MPI domain decomposition layer. Process 0 and 1 exchange particles from ghost layers. 9
1.5 University of Luxembourg's Iris cluster Infiniband interconnection. . . . 10
1.6 June 2020 Ranking of the Top 5 super computers in the world (*June 2020 top500 poster*). 11

2.1 Moore'S law 23
2.2 The roofline model for the naive brute force algorithm in collision detection. The roofline was generated using the `Intel advisor` tool. . 33
2.3 Schönauer vector triad (left column = bandwidth of data path, black arrow = cache line transfer, red arrow = write- allocate cache line transfer) (Erlangen Regional Computing Center, 2019) 36
2.4 XDEM call-graph 38
2.5 GRAPH 39

3.1 Workflow of an XDEM simulation: the *Broad-phase Collision Detection* finds the pairs of particles that may interact with each other by replacing the particles real shape with bounding volume and it returns a list. The latter is used in the *Narrow-phase Collision Detection* to perform a collision detection with particles real shape. 48

3.2 Type of bounding volumes used in broad-phase: sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull (Ericson, 2004). 49
3.3 Check object one against all other objects, then object two against all other objects (except object one). 54
3.4 Left: Recursive subdivision of a cube into octants. Right: The corresponding octree (Truong, Arikatla, and Enquobahrie, 2019). 55
3.5 A quadtree node with the first level of subdivision shown in black dotted lines, and the following level of subdivision in gray dashed lines. Dark gray objects overlap the first-level dividing planes and become stuck at the current level. Medium gray objects propagate one level down before becoming stuck. Here, only the white objects descend two levels (Ericson, 2004). 57
3.6 Regular domain decomposition as a quadtree. 57
3.7 Loose quadtree representation. The nodes have been expanded by half the node width in the two directions. 57
3.8 Representation of a regular quadtree and loose quadtree (Ericson, 2004). 57
3.9 (A) 58
3.10 (B) 58
3.11 Uniform grid approach. (A) Uniform grid subdivision in equal sized cells. Each particle is assigned to a cell where its center is located. (B) The dark blue particle is checked with the light blue particle contained in the same cell. It is also checked against green particles located in the direct neighboring cells in light green. 58
3.12 Hierarchical grid level 0. 59
3.13 Hierarchical grid level 1. 59
3.14 Hierarchical grid level 2. 59
3.15 Hierarchical grid level 3. 59

3.16 Two dimensional hierarchical grid example. The level 0 corresponds to the domain and contains all the particles. The first level is the coarser grid (in red) and contains the biggest particle in red. The second level encapsulated the yellow particles without the biggest red particle. The third and last level contains the blue and green particles, that inserted in all the three grids. 59
3.17 An example of a bounding volume hierarchy using rectangles as bounding volumes (Schreiberx, 2020). 61
3.18 Comparison results for *Slow Broad-Phase Algorithms* from 1*k* to 100*k* of particles. 63
3.19 Comparison results for *Fast Broad-Phase Algorithms* from 1*k* to 1*M* of particles. 64
3.20 OpenMP scalable results for *Fast Broad-Phase Algorithms* from 1 to 24 of cores. 66

4.1 Flow chart of XDEM software detailing the main different steps in an iterative simulation. 80
4.2 Collision detection (broad-phase and narrow-phase) process workload in XDEM. The broad-phase is the main computational time consumer. 81
4.3 Illustration of the cell linked method. For the particles in green cell, a collision is checked with particles in the same cell (green particles), and also within the immediate neighbour cells (blue). 82
4.4 Initial configuration and update of the Verlet list. 84
4.5 New flow chart of XDEM software. A construction of the Verlet list is added. If the conditions are satisfied, the list is kept and the broad-phase is skipped and the simulation continues directly to the narrow-phase. 86
4.6 The test case used for the performance evaluation simulates the hopper discharge of 125k, 250k and half a million particles. It shows any overview of the set-up with the particles coloured according to their size. 92

4.7 Granular flows on roughed inclined plane. The rough plane has particles vibrating at $50Hz$ frequency with $200mm$ amplitude. The free flow particles are coloured according to their velocity 93

4.8 Simulation of an avalanche at the top of a habitable valley. The particles bed represents a cohesive snow model. 94

4.9 Furnace of the combustion of Biomass. The particle bed is arranged on four (4) moving grates. The bed is heated up in the combustion chamber by inlets located below the grates. The particles are colored according to their surface temperature. 95

4.10 Powder levelling for Selective Laser Melting. 96

4.11 Dependence of broad-phase, narrow-phase and interactions models on skin K factor. The vertical blue dashed lines show the optimum K for each simulations corresponding to the lowest overall simulation time. The orange horizontal line represents the simulation time for a constant skin equal to the particles radius. The skin distance is capped by the cell size in all simulations. 97

4.12 Dependence of executed broad-phase in percentage upon the skin K factor. The percentage correspond to the number of executed broad-phase over the total number of steps in the simulation. 100

4.13 Simulation time overhead compared to the optimum for each K value for all test cases. 101

5.1 XDEM user specifications. 108

5.2 In the Verlet buffer approach, the bounding sphere of the particles is extended by the skin distance such as $R_{NL} = R_C + skin$. 111

5.3 The loosely-coupled or "black-box" interface between DAKOTA SOFTWARE and a user-supplied simulation code (Adams et al., 2019). 113

5.4 Dependence of broad-phase, narrow-phase and interactions models on skin distance factor K. 115

5.5 The SOGA solver convergence with the biomass case. Three populations with different initial sizes have been considered: 10, 20, 50. 117

5.6 Design of experiment steps. S_f: Solid fraction, C_s: Ratio cell to particle size, Vel: Velocity, n: Number of particles, T: XDEM computational time. 119

5.7 Figure of a homogeneous particle system for near-optimal skin distance determination. 120

5.8 Response surfaces for the near-optimal skin distance factor K for solid fraction and ratio cell to particle size. 123

5.9 Response surfaces for the near-optimal skin distance factor K for solid fraction and velocity. 123

5.10 Response surfaces for the near-optimal skin distance factor K for solid fraction and number of particles. 123

5.11 Response surfaces for the near-optimal skin distance factor K for ratio cell to particle size and velocity. 123

5.12 Response surfaces for the near-optimal skin distance factor K for ratio cell to particle size and number of particles. 123

5.13 Response surfaces for the near-optimal skin distance factor K for velocity and number of particles. 123

5.14 The DAKOTA SOFTWARE parallelism and scheduling scheme. 126

6.1 The different phases of an XDEM iteration loop. 135

6.2 The test case used for the performance evaluation simulates the hopper discharge of 1 million particles. The left part shows an overview of the setup with the particles colored according to their size. The right side displays the middle slice allowing us to see the particle velocity distribution during the discharge process. 139

6.3 Impact of data structure in sequential for Hopper test (from 250k left to 1M right). 141

6.4 Impact of different memory allocators on one iteration time with OpenMP version on one node (from 1 to 28 cores) for Hopper test case (1M particles). 143

6.5 SpeedUp of MPI and OpenMP versions on one node *i.e.* up to 28 cores. 144

6.6 Speedup of hybrid MPI+OpenMP executions for different number of threads per process on the Hopper Discharge case with 1 million particles. 146

7.1 The Enel Green Power "Cornia 2" biomass combustion power plant. . . 153
7.2 Sequential simulation time comparison between 03/2017 and 10/2020 XDEM versions. 155
7.3 XDEM main parts simulation time comparison between 03/2017 and 10/2020 XDEM versions. The y-axis is represented in log time. 156
7.4 Strong scaling of the main loop’s simulation time. The number of MPI processes is on the x-axis, and the log simulation time on the y-axis. The execution was made on one node with 28 cores. 157
7.5 Strong scaling of the main loop’s simulation time of three hybrid MPI/OpenMP parallel configurations. The number of cores is on the x-axis, and the simulation time on the y-axis. The executions was made on one node with 28 cores (exclusive reservation). 159
7.6 Speed up comparison between 03/2017 XDEM version in Full MPI configuration and 10/2020 XDEM version in Full OpenMP configuration. The x-axis represents the number of processes and the speed up on the y-axis. The executed was made on one node with 28 cores. . . . 160
7.7 Improvements comparison between 03/2017 XDEM version in Full MPI configuration and 10/2020 XDEM version in Full OpenMP configuration. The comparison is made between sequential and parallel simulations using 28 cores. We compare the improvements of some critical parts of the XDEM code: the broad-phase, the narrow-phase, the integration (dynamic and conversion), and the dynamic models. The x-axis represents the critical parts of the XDEM code and the improvement in percentage (compared to sequential) on the y-axis. . . 161
7.8 Dam break initial configuration. Light particles (bottom) in yellow and heavy particles in red (upper) are initially positioned within a column of water. 163

7.9 Different length scales in high-Stokes three-phase flows: bulk (coarse) scale and fluid fine scale (left figure). Schematic of the solution procedure for the bulk and fine length-scale in the simulation. The two boxes represent the different models adopted, while the arrows show the communication between the scales schematically. A coarse grid (top) is used to perform the volume averaging and to solve the fluid-particle interaction. Particle-related fields are mapped to the supporting domain (bottom) then, a finer grid is used to solve the fluid equations (Pozzetti and Peters, 2018). 164
7.10 Three-phase dam-break multi-scale strategies. 165
7.11 XDEM and OPENFOAM parallelization strategies. XDEM uses an hybrid MPI+OpenMP approach while OPENFOAM is partitioned using only MPI. 166
7.12 The Iris cluster computing nodes. 167
7.13 Speed up of hybrid MPI/OpenMP runs with different process/thread configurations. The x-axis represents the number of processes and the speed up on the y-axis. 169
7.14 XDEM and OPENFOAM simulation time proportion comparison. The left figure compares the proportions for a Full MPI configuration for XDEM. The right figure compares the proportions for a hybrid 2 MPI processes per node and 14 OpenMP threads per MPI process for XDEM.171
7.15 XDEM and OPENFOAM load-imbalances for the dam-break domain decomposition in the Full MPI parallel configuration. 172
7.16 XDEM load-imbalances for the dam-break domain decomposition in the hybrid MPI/OpenMP parallel configuration. 173

8.1 Biomass combustion chamber 2D design. 181
8.2 Biomass combustion chamber 3D views. 183
8.3 Gas phase temperature distribution at different simulation time: $5s$ (upper left), $50s$ (upper right), $250s$ (lower left), $600s$ (lower right). . . . 188
8.4 Surface bed temperature distribution. 189
8.5 Particle composition in the inlet, outlet and along the grate length. . . 190

8.6 Particle average composition in the inlet, outlet and along the grate. . . 191

8.7 Time proportion of XDEM and OpenFoam in the coupling simulation. 192

LIST OF TABLES

1.1 Table to compare DEM software specifications. 15

2.1 Hotspots in XDEM using INTEL VTUNE. 37
2.2 Time table memory allocation details. 40
2.3 Cache data access summary. 42

3.1 Algorithms that have been implemented and parallelized in the C++ framework. The brute force, the grid and tree algorithms were implemented from scratch. The CGAL and Bullet libraries use spatial partitioning algorithms while FCL was used with a AABB tree based algorithm. 53

4.1 Summary of the performance results of the Verlet buffer method over the different testcases. 102

5.1 SOGA parameters 116
5.2 Simulation designs for near-optimal skin distance. 121
5.3 Overview of the Verlet buffer method results with the surrogate model over the different test-cases. 124
5.4 Scheduled job statistics. 127

6.1 Characteristics of the containers used in XDEM before and after our optimizations. Only the operations used in XDEM are considered. . . 138

7.1 Table to compare XDEM versions simulation times in sequential. . . . 154
7.2 Table to compare XDEM versions simulation times in parallel, full MPI configuration. 158
7.3 Gas and liquid phases properties. 165

8.1 Characteristics and operating conditions of the super-heater. PA = primary air, SA = secondary air 181
8.2 Biomass analysis. ar = as received, daf = dry ash free 182

LIST OF ABBREVIATIONS

DEM	**D**iscrete **E**lement **M**ethod
XDEM	e**X**tended **D**iscrete **E**lement **M**ethod
LuXDEM	**Lu**xembourg e**X**tended **D**iscrete **E**lement **M**ethod Centre
MD	**M**olecular **D**ynamic
CFD	**C**omputational **F**luid **D**ynamic
HPC	**H**igh **P**erformance **C**omputing
BP	**B**road **P**hase
NP	**N**arrow **P**hase
AABB	**A**xis **A**lign **B**ounding **B**ox
OBB	**O**riented **B**ounding **B**ox
DOP	**D**iscrete **O**roented **P**olytope
CGAL	**C**omputational **G**eometry **A**lgorithms **L**ibrary
FCL	**F**ast **C**ollision **L**ibrary
BVH	**B**ounding **V**olume **H**ierarchy
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphic **P**rocessing **U**nit
NUMA	**N**on **U**niform **M**emory **A**ccess
SMP	**S**ymmetric **M**ulti **P**rocessing
EDR	**E**ighteen **D**ata **R**ate
GPFS	**G**eneral **P**arallel **F**ile **S**ystem
DAKOTA	**D**esign **A**nalysis **K**it for **O**ptimisation and **T**erascale **A**pplications
DACE	**D**ace **A**nalysis **C**omputer **E**xperiments
SOGA	**S**ingle **O**bjective **G**enetic **A**lgorithm
MOGA	**M**ultiple **O**bjective **G**enetic **A**lgorithm
PA	**P**rimary **A**ir
SA	**S**econdary **A**ir
FGR	**F**lue **G**as **R**ecirculation
PaSR	**P**artially **S**tirred **R**eactor
FEM	**F**inite **E**lement **M**ethod
FDM	**F**inite **D**ifference **M**ethod
FVM	**F**inite **V**olume **M**ethod

LIST OF SYMBOLS

a	distance	m
W	power	W (J s^{-1})
ω	angular frequency	rad
W	angular frequency	rad

To my dear family…

Chapter 1

Introduction

1.1 Granular materials modeling and particle simulations

1.1.1 Granular materials

Granular materials are an aggregation of macroscopic particles and are omnipresent, and their behavior is of great importance in pharmaceutical, mining, food processing, iron making industries, avalanches, cereal storage, powder mixing, and frictional materials (concrete). They can be classified into two categories: *powders* and *granulars*. Particles above 100 µm are considered granular, and their dynamics are strongly influenced by their interaction with their neighbors and/or surrounding fluid through frictional contact. On the other hand, powders are composed of tiny particles (1 −100 µm) that easily float in a gas. The behavior of granular materials can range from solid-like material (cohesive soil (Peters and Džiugys, 2002)) to fluid-like due to their variety and usually involves multi-scale phenomena. Medicine tablets, coffee, coal, wood chips, sand, snow, rocks, planets, or even galaxies are few examples considered granular materials.

Granulates are distinguished from molecules and fine particles by the size of the particles that compose them. They must be large enough so that their motion is not vulnerable to thermal fluctuations.

FIGURE 1.1: Examples of granular materials in industry: Tablets pills medicine, Silo for storing corn, and Crusher with conveyor

Granulates are the most widely manipulated material on the planet after water (Richard et al., 2005). However, despite their far-reaching importance in countless fields, efforts to understand and predict the specific granular behavior remain under-researched despite a need for a better understanding. Even today, the design of simple structures handling granular materials such as static grain silos often fail. Therefore, there is a need and demand for new predictive tools such as numerical simulations or analytical procedures to understand granular materials behaviors in industrial environments.

Simulations have become great tools that have been used to design and optimize industrial processes. The granular materials are no exceptions, and simulations are used to deepen the understanding of their behaviors. These simulations use physical and mathematical modeling as a basis to reproduce, predict, and set-up the physical process.

1.1.2 Modeling Methods

When it comes to modeling and simulation of physics process in general and particulate systems, there are two widely used approaches: continuum or Eulerian (Liu and WALkINGTON, 2001) and discrete or Lagrangian (Lagrange, 1853). The continuum approach assumes the materials to be continued in their internal structure representation. It has been successfully applied to different domains and materials, such as fluids, metals, and most homogeneous materials. On the other hand, the discrete approach considers materials as a system of independent and interacting particles.

1.1.2.1 Continuum approach

In the continuum method, granular matter's constitutive behavior is defined by constitutive laws, commonly expressed in the form of differential equations that relate to mechanical field variables (e.g., Stress and Strain). The simulation of material with this approach assumes that it is continuous and fills the space it occupies. As a consequence, the behavior of individual particles is ignored. The resulting constitutive equations are solved numerically (e.g., Finite Element Method). The crucial issues involved in using continuum methods for the granular material simulation are the proper formulation of constitutive behavior. Relevant stress-strain laws for materials often do not apply or are unnecessarily complicated. Particle system processes are often strongly dependent on particle level behavior.

The most commonly used continuum methods are the Finite Element Method (FEM), Finite Difference Method (FDM), and the Finite Volume Method (FVM).

- The most straightforward approach to discretizing partial differential equations is the finite-difference method. You consider a point in space where you take the equation's continuum representation and substitute it with a series of discrete equations called equations of finite-difference. On a regular grid, which can be used for very effective solution methods, the finite-difference method is usually defined. Therefore for unconventional geometries, the approach is not commonly used, but most frequently for rectangular or block-shaped versions.
- The finite-element method is a method that subdivides a material into geometrically simple shapes of small but finite-sized elements. The so-called finite element mesh constitutes the collection of all these simple shapes.
- The finite-volume method is similar to the finite-element method in that the model is first divided into small but finite-sized elements of geometrically simple shapes. Apart from this, the finite-volume method is very different from the finite-element method, beginning with the concept of elements, which are instead referred to as cells.

1.1.2.2 Discrete approach

In comparison to the continuum approach, discrete methods model every particle as a discrete object and portray granular material as a bulk system of particles. The overall (macroscopic) system behavior results from individual particle interactions. It makes the discrete solution very useful for analyzing phenomena at the particle length scale and gives a better simulation of the particles' mass behavior. As granular material micromechanics can be more accurately modeled with discrete methods, they are best suited for modeling the flow and massive displacements of discontinuous material (Kabore et al., 2018).

The discrete methods or particle-based models are numerical models that consider materials into individual and independent particle system (Nambu and Jona-Lasinio, 1961). Therefore, they are perfect at modeling granular materials and can also be used to model fluids, metals, and most homogeneous materials. The idea that matter is made of discrete elements dates back over 2000 years, with the Ancient

Greek atomists Leucippus, Democritus, and Epicurus (Berryman, 2004b; Berryman, 2004a; Furley, 1967) arguing that nature was composed of *atomos* or *indivisible individuals*. Numerical modeling of materials behavior considers different scales from atomic to macro scale. These different scale levels are captured in different methods such as the Molecular Dynamics (MD), the Particle In Cell (PIC), and the Discrete Element Method (DEM) (see Fig. **??**).

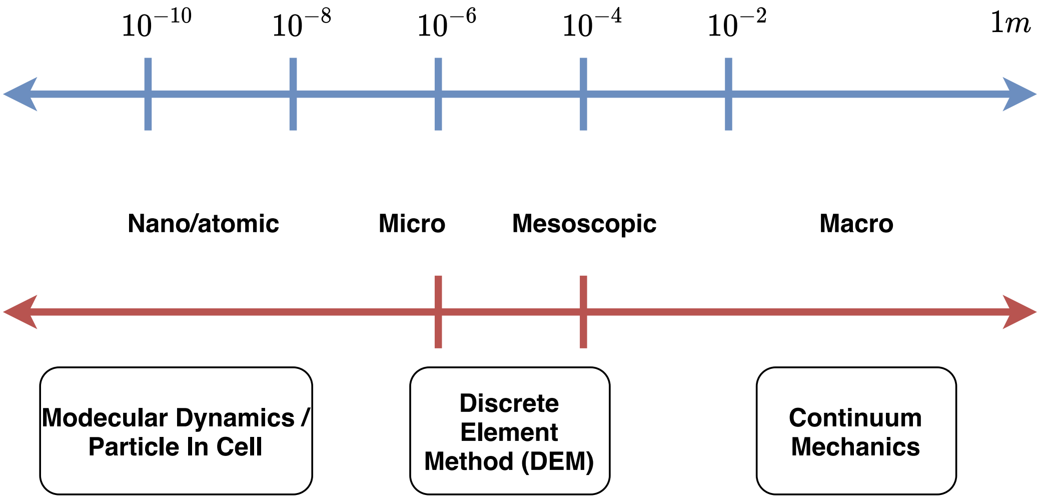


FIGURE 1.2: Common approaches in computational mechanics.

The computational mechanics' field can be divided into different scales, as presented in Fig. 1.2: macro, mesoscopic, microscopic, and nanoscales. Molecular Dynamics can be used at micro, meso, and nano scales structures. The DEM approach can be used at micro, meso, and even macro-scales structures, making the DEM a perfect method to model granular materials in its broad diversity. The continuum mechanics is commonly used to model the macro-scale structures, where the finite element method can be used.

1.1.3 Particle simulations

Simulations of particulate matter started with the invention of molecular dynamics in the late 1950s, which models the physical motion of atoms and molecules in a multi-body simulation. The atoms' motion is determined by solving Newton's second law of motion for a system of interacting particles, where forces between the particles and potential energy are defined by inter-atomic potentials or molecular mechanics force fields.

Particle-based methods introduced new simulations that were hardly possible with the continuum approach but come with additional difficulties, extra computing time, and programming effort. Indeed, while such Lagrangian approaches can have substantial advantages over conventional mesh-based methods, their accurate and effective implementation often poses several challenges.

The Discrete Element Method (DEM) is a family of numerical methods for computing discrete particles' motions. The global behavior of the system is gauge from the individual motion and mutual interactions of the particles. Typically, each DEM particle represents a separate and independent element with a calculated momentum and energy field. The DEM approach can be schematized as described in the Fig. 1.3 below. Cundall (Cundall and Strack, 1979) first proposed it in 1971. While very similar to MD, the DEM method is characterized by the addition of rotational degrees of freedom and elastic contact, and complicated geometry. These additions make the method ideally suited for modeling the bulk behavior of granular materials.

It has become possible to simulate millions of particles on a single processor numerically with developments in computing power and numerical algorithms for nearest neighbor sorting. DEM simulations, however, are relatively costly, intensive, and challenging in terms of calculation, reducing either the length of a simulation or the number of particles. Additional challenges are also faced when coupling DEM simulations to continuum solvers such as Computational Fluid Dynamics (CFD).

1.1.4 The eXtended Discrete Element Method(XDEM)

The XDEM software is a numerical multi-physics simulation framework (Samiei and Peters, 2010) based on the dynamics of granular material or particles described by the classical DEM (Cundall and Strack, 1979; Allen and Tildesley, 1990). It is extended by additional properties such as the thermodynamic state, stress/strain or electro-magnetic field for each particle (Peters, 2013; Peters and Pozzetti, 2017; Mahmoudi et al., 2016a). It is organized as a C++ library composed of a set of

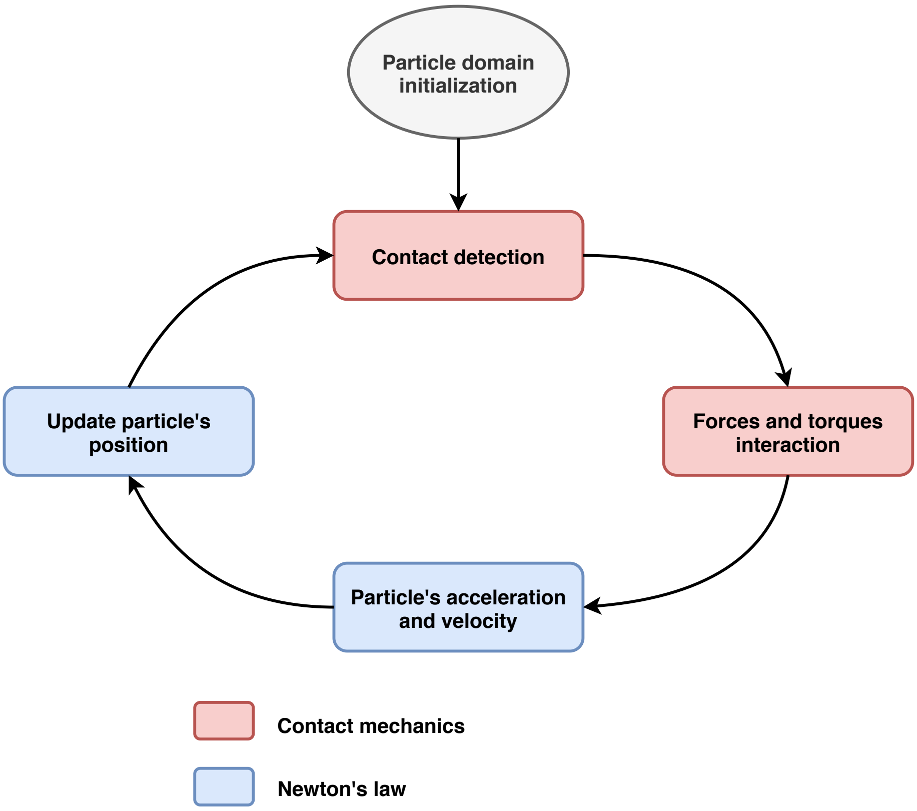

Particle domain initialization
Contact detection
Forces and torques interaction
Particle's acceleration and velocity
Update particle's position
Contact mechanics
Newton's law

FIGURE 1.3: Discrete element method loop's scheme.

modules: *Dynamics* for the pure DEM part, *Conversion* for the chemical conversion and thermodynamics, *CFD coupling* for the coupling through an external CFD library such as OpenFOAM (Jasak, Jemcov, and Tukovic, 2007).

An XDEM simulation is an *iterative* time loop which contains the following main phases:

- **Prediction**: initiates and prepares the quantity of particle motion for the upcoming calculation. It is an optional phase used only with some specific integration models;
- **Broad-Phase**: uses a fast but approximate collision detection to build a list of particle pairs that can potentially interact. During this phase, the particles are represented by bounding spheres with an appropriate radius (to express the interaction range);
- **Narrow-Phase**: processes the list of potentially interacting particle pairs and performs a precise contact detection using the actual shape of the particle (*e.g.* sphere, cube, disk, cylinder, triangle, etc.). It calculates the overlap/distance, the contact point, and the direction between the two particles;
- **Apply Physics Model**: this phase selects the physics models defined in the particle properties (*e.g.* for impact, bonding, rolling, conduction, radiation, chemical reaction) and calculates the contribution of the interaction to each particle involved (in terms of force, torque, heat flux, chemical specie mass fraction, Etc.).
- **Integration**: updates the state of the particles after accumulating all the interactions' contributions. Different integration models are available for the different components of the particles' state (*e.g.* position/orientation, temperature, chemical composition).

Each of the simulation modules can be enabled separately and have specific time settings. The XDEM simulation driver is responsible for executing at each iteration the required phases for the activated modules.

To benefit from larger-scale As described in previous work (Besseron et al., 2013), XDEM parallelization is based on a classical domain decomposition approach. It relies mainly on three concepts:

- the simulation *domain* containing all the particles, is split in regular sub-division called *cells*;
- cells are grouped to form a *sub-domain* or *partition* which will be assigned to processes participating in the simulation;
- a partitioning algorithm is responsible for the creation of these partitions (Rousset, Besseron, and Peters, 2017).

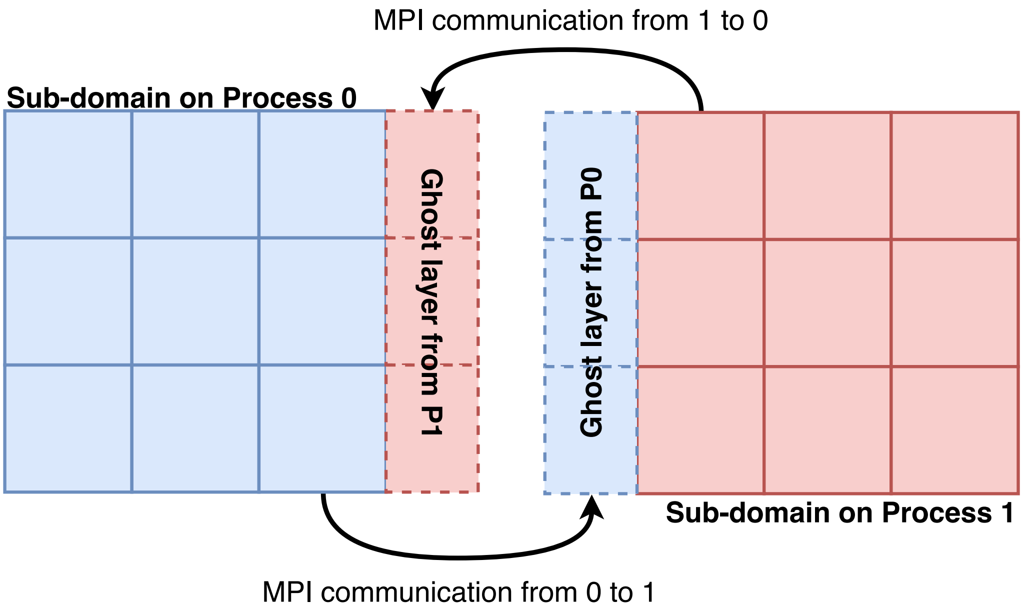


FIGURE 1.4: Classic MPI domain decomposition layer. Process 0 and 1 exchange particles from ghost layers.

The parallel simulation driver executes the different partitions' evaluation while computations inside a partition or a sub-domain are performed sequentially. Because of parallel execution, additional communications are required to exchange particle data between neighboring cells located on different partitions. A layer of *ghost cells* (or *ghost layer* as shown on Fig. 1.4) is added at the boundaries of each partition for this purpose. These layers are thus used to represent cells located on the other processes.

1.2 High performance computing

High-performance computing (HPC) is a set of techniques for performing billions of billions of operations on large amounts of data using super-powered computers. It is also defined as the "use of parallel processing for running advanced application programs efficiently, reliably, and quickly." The supercomputers are an aggregation of computing power involving thousands of processors performing billions of computations on a massive amount of data in parallel using fast network infrastructure (Fig. 1.5). It is today an essential tool for researchers and engineers to solve cutting-edge problems.

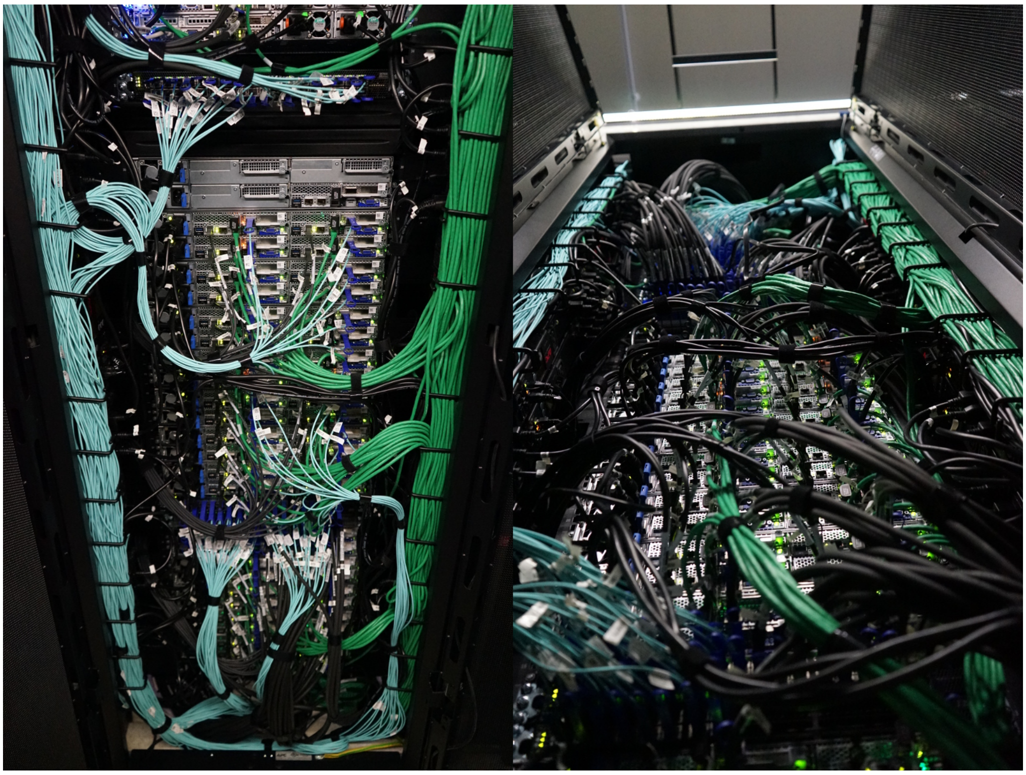
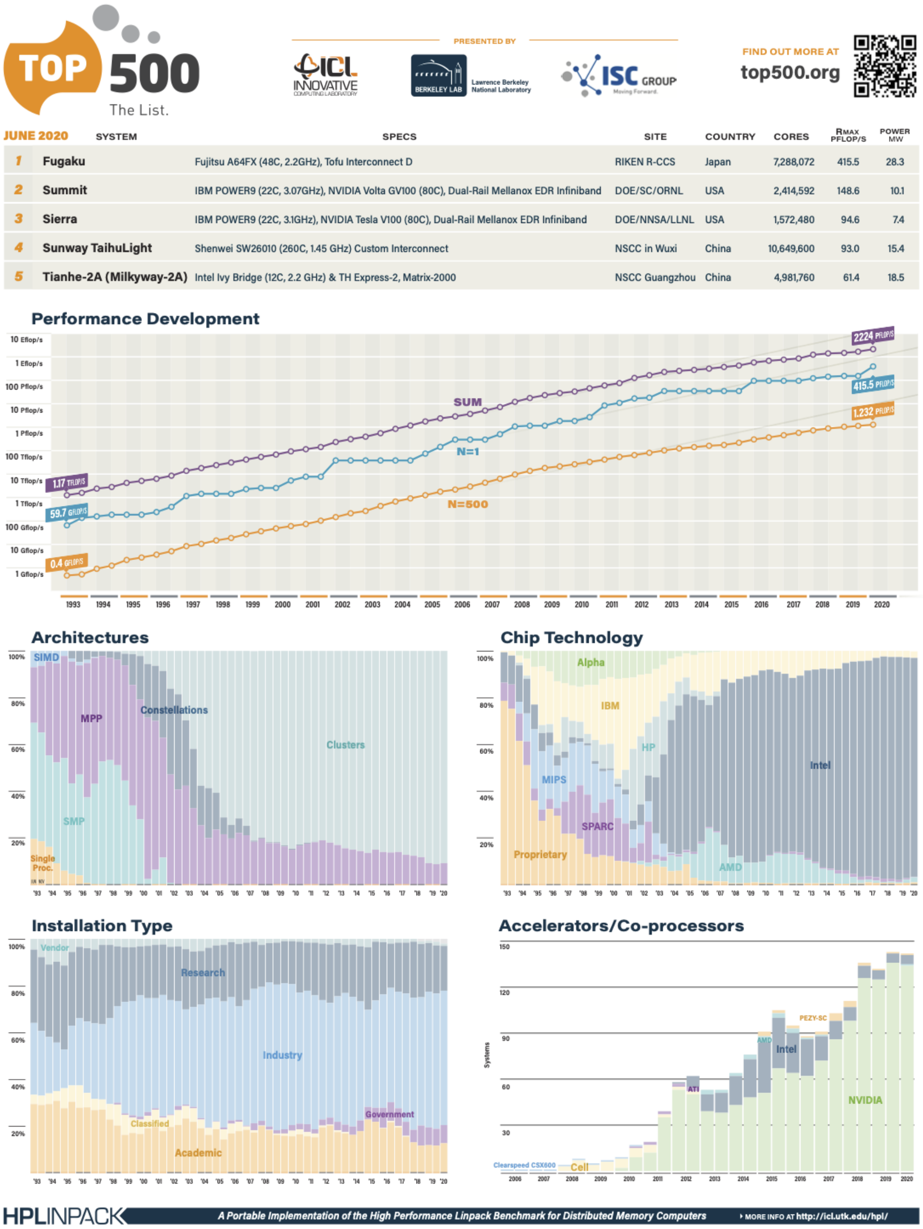


FIGURE 1.5: University of Luxembourg's Iris cluster Infiniband interconnection.

HPC has enabled tremendous applications in weather prediction, nuclear, environment, fluid mechanics, aerospace, astrophysics, data science, artificial intelligence, and hydrology. It is possible to simulate large scale and parallel systems that require high computing power and are data intensive. There is currently a development race for the most powerful supercomputer between Western nations (Europe and the US) and China. **TOP500** is a world ranking project of supercomputers that ranks the first 500 supercomputers in the world based on the number of floating-point operations per second (FLOPS/s) they can perform. In 2020, the rank is dominated by the Japanese machine Fugaku (see Fig. 1.6) which is installed at the RIKEN Center for Computational Science (R-CCS) in Kobe, Japan and has a theoretical peak performance of 415.5 petaflops with more than 7 million cores.



TOP 500 The List.

PRESENTED BY

FIND OUT MORE AT top500.org

JUNE 2020	SYSTEM	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	**Fugaku**	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,288,072	415.5	28.3
2	**Summit**	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	10.1
3	**Sierra**	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	94.6	7.4
4	**Sunway TaihuLight**	Shenwei SW26010 (260C, 1.45 GHz) Custom Interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
5	**Tianhe-2A (Milkyway-2A)**	Intel Ivy Bridge (12C, 2.2 GHz) & TH Express-2, Matrix-2000	NSCC Guangzhou	China	4,981,760	61.4	18.5

Performance Development

Architectures

Chip Technology

Installation Type

Accelerators/Co-processors

HPLINPACK A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers ‣ MORE INFO AT http://icl.utk.edu/hpl/

FIGURE 1.6: June 2020 Ranking of the Top 5 super computers in the world (*June 2020 top500 poster*).

The high computational demand is one of the key disadvantages of the DEM method, despite numerous advantages. Modern computers have provided high-performance computing with powerful hardware platforms, but many current DEM codes are usually serially coded, which entirely prevents them from using modern computing capability. Adapting DEM codes to run on supercomputers enables to run large scale DEM simulations that will accelerate research outcomes by dramatically reducing the computational time for these studies while providing high-resolution representation of physical experiments.

1.3 DEM software: A state of art

A bibliographical search allowed us to establish a list of several existing platforms or projects of parallel DEM platforms. This list is not exhaustive, but it is intended to be as complete as possible.

LIGGGHTS (Kloss et al., 2012) is a well known DEM software developed by DCS Computing GmbH, Linz, in Austria. It is constructed on top of LAMMPS (Plimpton, Crozier, and Thompson, 2007), a classical molecular dynamics simulator. LIGGGHTS is an open-source software package for DEM simulations, including granular materials and heat transfer. It is inherited from LAMMPS, and it is parallelized through MPI but with a dynamic domain decomposition (unlike LAMMPS). XDEM and LIGGGHTS decompose the simulation domain in a very similar manner. However, unlike LIGGGHTS (in its free version), XDEM does not produce a Cartesian grid of the subdomain (it also has RCB decomposition in the commercial license). XDEM on the other hand uses topological, geometric, and Hypergraph partitionners from the ZOLTAN toolkit (Devine et al., 2002). It gives us an edge over a better load balance among the MPI processes and more flexibility. LIGGGHTS has, in its commercial license, a lock-free OpenMP parallelization (Berger et al., 2015) on top of the MPI decomposition and can therefore perform hybrid simulations. In (Berger et al., 2015), a speedup of 64 over 128 cores for a Hopper discharge (silo) test case with 1.5 million particles was shown, as well as a speedup 55 on 128 cores with a mixing process test case with 770000 particles.

YADE (Kozicki and Donze, 2009) is an open-source C++ and Python framework for discrete numerical models, focused on Discrete Element Method. The code has a generic design to provide extreme flexibility in order to add new features. It can also be coupled to other software or import data from a third software. YADE also directly incorporates Lattice Geometrical Models (LGM) and FEM, making it a complete software. A parallel version using shared memory (OpenMP) of the software was released in 2013 and consisted of a brute force parallelization of loops in addition to a parallel collision module. It presents a speedup of 12 on 20 threads for a simulation of one million particles (DEM8 open-source presentation). In 2018, a distributed memory (OpenMPI) working alpha version was released with no substantial change to the existing code. The MPI parallelization of the code is divided between Python and C++ as follows: 90% of Python parallelization with **mpi4py** and 10% of C++ using **OpenMPI**. An ETA "beta" was released in 2019. Unlike XDEM, YADE does not offer additional thermodynamics properties for particles with a conversion module. It, therefore, offers less possibility of simulating complex and complete DEM cases such as a fluidized bed.

MERCURYDPM is a scientific software for discrete particle simulations. It is a very adaptable, object-oriented C++ and Fortran code working on Linux distributions, Mac OS, and Windows 10 and released under the BSD 3-clause license (Weinhart et al., 2020). It was initially designed for granular chute flow but can now be used for more granular problems, including geophysical modeling of cinder cone creation (Weinhart et al., 2020). **MERCURYDPM** supports polydiverse particles, curved walls, and coarse-graining analysis for extracting continuum fields as density, momentum, and stress. Many contact force models are implemented, including elastic or dissipative normal forces and tangential friction. Flat or polyhedral walls are modeled, as well as fixed-particle walls. The code support parallel processing through a distributed approach using MPI. The domain decomposition is classical and Cartesian, where the subdomains have equal size and are associated with an MPI process. It does, therefore, not support complex domain decomposition, unlike XDEM. The code is claimed to have 40% speedup with hyper-threading and above 60% speedup without a rotating drum with varying width.

ROCKY DEM (Granular Dynamics International and Software, 2020) is commercial and industry-oriented software developed by *ESSS* and is claimed by authors to be one of the complete DEM software available on the market. It simulates the flow behavior of bulk and granular materials with complex shapes and size distributions (polydiverse particles). It supports 3D shapes, 2D shells, and fibers that are flexible or rigid. The main feature that distinguishes the ROCKY DEM software is its support of breakage models to mimic the breakage behavior of particle shape. It also fully integrates the ANSYS package for CFD. The code can be executed mainly on GPU but also CPU. Since release 4 of Rocky, the software allows execution with multiple GPUs through MPI. **ROCKY DEM** can therefore run on many GPU cards.

EDEM ("2.4 Theory Reference Guide, 2011") is also a commercial software developed by DEM-Solutions. It is aimed to simulate and analyze the behavior of bulk materials such as coal, mined ores, soil, tablet, and powders. It supports a large range of complex shapes and thousands of pre-calibrated material models for rocks and ores. The EDEM software can be coupled with Finite Element Analysis (FEA), Multi-body Dynamics (MBD), and Computational Fluid Dynamics (CFD). EDEM is parallelized for both shared and distributed memory and offers the way to target CPU and GPU architectures. It also supports multi-GPU capacity through MPI.

Finally, **BLAZE DEM** (Govender, Wilke, and Kok, 2016) is A GPU Based Polyhedral DEM particle transport code and specifically targeted for NVIDIA GPU platforms.

In Tab. 1.1, we compare the DEM software mentioned above according to five criteria: *license, target node, parallelization approach, programming language, and coupling capabilities*.

Without going into individual comparisons, all the above software tools were mainly developed with a particular purpose in mind. As such, all of them are useful tools that serve very well in their field of applicability. However, there are no universal tools that could be used for everything. For this reason, each of these platforms is briefly presented, focusing mainly on three main criteria: the parallelization strategies used such as distributed or shared memory (MPI, OpenMP), the targeted architecture

DEM software specifications					
Software	License	Target node	Parallel approach	Language	CFD coupling
LIGGGHTS	Open Source	CPU	MPI	C++	YES
YADE	Open source	CPU	MPI	C++	YES
MERCURYDPM	Open source	CPU	MPI/OpenMP	C++	NO
ROCKYDEM	Commercial	GPU/CPU	Cuda/MPI	Cuda C++	YES
EDEM	Commercial	GPU/CPU	Cuda/MPI	Cuda C++	YES
BLAZEGPU	Open source	GPU	Cuda MPI	Cuda C++	NO
XDEM	Commercial	CPU	MPI	C++	YES

TABLE 1.1: Table to compare DEM software specifications.

(CPU or GPU), and some scalability or speedup information when it is available (speedups are challenging to interpret because it is very test case dependent).

1.4 Motivation and Objectives

The main objective of this doctoral research is to run large scale systems with the XDEM software on HPC platforms. The first motivation behind this research is to understand the behavior, limitations, and bottlenecks (communications overhead, load balance, scalability limits) of the XDEM software by performing a complete profile of the code. The second objective is to develop and apply several *HPC* techniques, algorithms, and parallelization methods to overcome the software limitations. The final objective is to be able to use XDEM to perform highly scalable simulations of applications such are *Blastfurnace* for iron making simulation, *Biomass combustion* for green energy production, and particle flow (*Avalanche*, *landslides*. To achieve this goal, it is fundamental to adopt computing and memory-efficient implementation techniques combined with different parallelization taxonomies with the use of *HPC* resources.

1.5 Contributions

The main benefits and contributions of the current thesis are summarized as follows:

- A complete review and development of collision detection algorithms and their implementation in a shared memory approach. A `C++` framework has been developed to implement, study and compare most popular collision detection algorithms. The best approaches were implemented within the XDEM software.

- An original Verlet list implementation for DEM that takes the particle flow regime into account when selecting the skin margin to further enhance the efficiency of the algorithm is presented with performance comparison. An optimization study has been conducted to determine which optimum skin margin gives the best computing performance depending on particle local flow regime parameters. Therefore, a polynomial function expressing the optimum skin margin as a function of the simulation parameters was proposed.
- A full *OpenMP* parallelization layer has been added to the XDEM software. The implementation enables the possibility of running hybrid *MPI/OpenMP* simulations taking advantages of modern supercomputer NUMA architecture.
- Development of an entire large scale biomass combustion case. The application is an *XDEM-OpenFOAM* (DEM-CFD) coupling approach, with thousands of particles interacting with the surrounding gas phase.

1.6 Structure of the thesis

The current thesis is partially a collection of published and submitted scientific papers to peer-reviewed conferences and journals. The different papers give an overview of the optimization process of a large, complex, and legacy software such as the XDEM software.

The third chapter's content is an extended version of an article published in the 2017 Particles conference and is intended to be published in a journal as an extended version. The fourth chapter is a journal article currently under review and submitted to the Advances in Engineering Software. The fifth chapter has been presented in the 10^{th} IEEE Workshop Parallel/Distributed Combinatorics and Optimization (PDCO 2020) and published in the IEEE International Parallel and Distributed Processing Symposium (IPDPS2020). The eighth chapter has been published at the 12th European Conference on Industrial Furnaces and Boilers (INFUB12). The sixth chapter is an article presented in the 9^{th} Workshop on Applications for Multi-Core Architectures (WAMCA) and published in the 30^{th} International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD). The remaining chapters will be presented in paper format and are intended for publication.

Chapter 2 describes in-depth the process of profiling a software code to find bottlenecks. Different approaches are presented with a different tool to perform the profiling alongside performance metrics. The hotspots and memory footprint of the XDEM software are analyzed and presented, and the chapter outcomes' are used as steps to follow throughout the thesis.

The third chapter 3 is a consequence of the second chapter where the contact detection broad-phase has been identified as a hotspot. Therefore, it presents a general review of the broad-phase algorithms found in the literature and the implementation of a few of them in a framework for comparative benchmarking. An *OpenMP* parallel version of most of the algorithms is also proposed and their implementation and integration in the XDEM software.

Chapter 4 introduces a new Verlet list approach called the Local Verlet buffer approach. The method extends the classical Verlet list by taking into account the local flow regimes of each particle for the choice of the optimum skin margin. The method has been tested and compared against the classical Verlet list and different approaches (skin values) to assess its performance over the classical approach.

The fifth chapter 5 studies the different parameters that influence the optimum skin margin in the local Verlet buffer method. For this purpose, an optimization problem that required hundreds of simulations is solved using the DAKOTA software.

In chapter 6, we present the *OpenMP* implementation of the XDEM software in order to target the different high-performance computers. The different obstacles and challenges of parallelizing an existing and legacy code are presented with different solutions on algorithms and data structures. A performance test on hundreds of cores has also been conducted, and the results are analyzed through the scalability and speed up performance.

Chapter 7 focus on studying the XDEM parallel performance using two test cases: dam-break and biomass. We first compared the 03/2017 and 10/2020 XDEM versions to highlight the gains brought by our optimizations. We then study the dam-break test case's performance by running parallel simulations on 85 nodes on the HPC cluster. The scalability, speed up, and load imbalance are the different methods used to perform the parallel performance study in this chapter.

The eighth chapter 8 presents an application test case: a full and extensive scale biomass combustion process. It demonstrates the new capabilities of the XDEM software to run large scale, parallel, and coupled *DEM-CFD* simulations with millions of particles in a very reasonable time and computing resources.

Part I

Methodology

Chapter 2

PERFORMANCE ASSESSMENTS

"Measurements is the first step that leads to control and eventually to improvement."

H. James Harrington

"The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet."

Michael A. Jackson

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%"

Donald Knuth

2.1 Introduction

Measuring the performance and the efficiency of a computer program during or after software development is a path that any developer should follow. More often than not, especially in the research community, numerous software properties such as correctness, functionality, reliability, robustness, or portability are more valuable than performance. The latter statement is reinforced by the increase in the power of modern computing resources. As stated by Moore's law (Moore, 1965), the number of transistors inside a single chip has been doubling almost every two years since the 70s though the cost of computers is halved. Moore's law also resulted in the doubling of the processor frequency with that of transistors. It means that every two years, the new generation of CPUs is twice faster than the previous generation and costs half as much. From this perspective, programmers can only wait two years for new generation processors to run their programs twice faster than before without doing anything more (Sutter, 2005).

Nevertheless, Moore's law is an empirical and observational law that has to come to an end (Schaller, 1997). Some say it has already been happening, or we are no longer at the same rate. *"It is over. This year that became clear,"* says Charles Leiserson in 2019. Since 2005, although the number of transistors per processor has continued to increase at the rate predicted by Moore's law, we have experienced a decrease of the processor frequency, as shown in Fig. 2.1. Indeed, although single-core CPUs became more and more complex, that complexity is not translated into more performance, but rather, it turns into the failure of most programs to take advantage of this complexity. It also raised another difficulty to processor vendors: the power density (W/cm^2) inside a chip is increased by approximately $25 \sim 30\%$ per year (Gelsinger, 2004) generating an overheating problem inside chips. Overheating inside processors has become a serious concern, and the cooling systems are becoming more and more expensive.

One solution brought by the industry to overcome the issue of power dissipation is to make processors with many cores: *multi-core processors*. It has brought in new programming paradigms, multi-threaded programs, which come at an overhead price

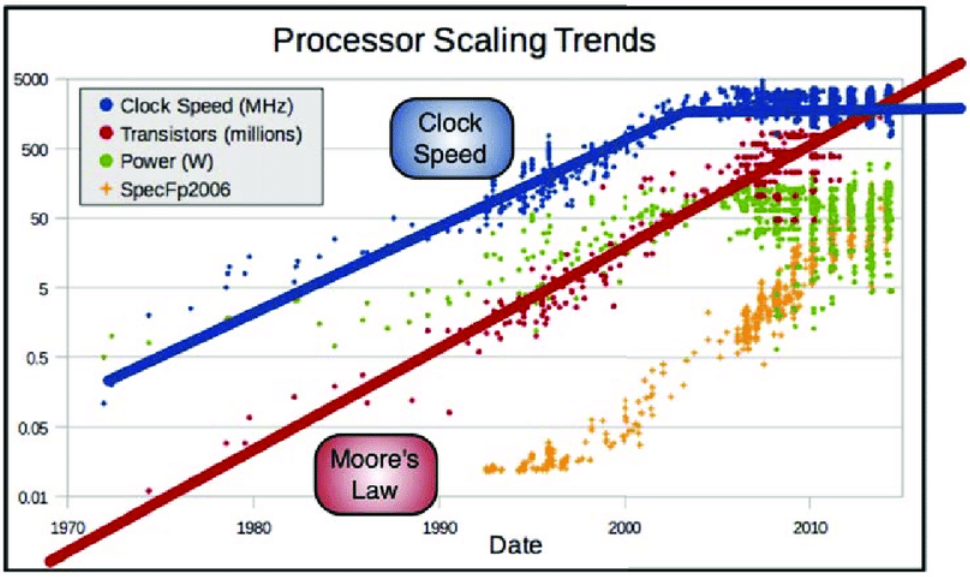


FIGURE 2.1: CPU transistor densities, clock speeds power and performance from 1970-2015 (Stewart and Lampl, 2017).

cost, and since programmers have to pay attention to the design and behavior of their code to get the most performance out of the CPUs. *Multi-core processors* come with complex memory design and hierarchy shared by the cores, and it has become difficult to understand how to take full benefit from such structure. It is, in summary, how performance becomes nowadays a widely used property for software requirements. However, what is performance, and what makes a program performant? How do we measure and analyze the performance of a computer program?

Sections 2.3 and 2.4 introduce two common methods for assessing a computer program performance and applied to our XDEM software. The results are discussed in section 2.6 and gave the path to follow during this research work.

2.2 Baseline

It is essential to have a starting point for the measure when carrying out a given program's performance tests. The starting point is usually the program's performance results at a given state of the period before making modifications or applying whatever optimization. It is then used as a reference state for upcoming performance tests. In this section, we will define our baseline code used later on to compare with all optimizations we have made during the research. We will also introduce the metrics we have used to characterize our performance results and what makes them an excellent metric to measure a specific performance count. Finally, we present the *real-life* test case used to assess the progress achieved with all our optimizations improvements.

2.2.1 XDEM version

XDEM as introduced in section 1.1.4 of chapter 1 is a C++ legacy software that has been developed inside the **LuxDEM** team for more than *fifteen years*. Several researchers/developers are continuously working and using it in a continuous integration manner. Therefore, it was necessary to define a baseline version to be used as a starting point for any optimization and comparison.

As we started the doctoral research on 1st of March 2017, it was then obvious to take an XDEM version from March 2017 (git hash: *18a22cbfdadf7fe2afd8bbc9ba02744d75b775d0*). It is a straightforward and perfect choice to begin with as a significant update on the partitioner has been made at the beginning of the year, and no other extensive updates were expected in the following months. The results presented in section 2.6 has been collected using the baseline XDEM version.

2.2.2 Metrics of performance

Measuring a computer program's performance involves measuring a metric that can be a count of occurring events, a period, or an amount of a defined parameter. However, it is usually convenient and interesting to normalize event counts to a common time basis to provide a speed metric such as instructions or operations executed per second. This metric is called a rate metric or throughput and is calculated by dividing the number of events that occurred in a given interval by the time interval over which the events occurred (Lilja, 2005). Since a rate metric is normalized to a common time basis, such as seconds, it is useful for comparing different measurements made over different time intervals (Lilja, 2005).

2.2.2.1 Characteristics of performance metrics

The choice of the metric only depends on the needs and the cost of measuring that metric. However, metrics need to fulfill some characteristics to be considered good metrics (Lilja, 2005).

- A good metric is certainly **consistent**, meaning that its units and definition remain the same across different systems.
- A metric is **reliable** when system or configuration **A** always gives better performance metrics than system or configuration **B**. The comparison outcome should always give the same result no matter how many times the test is executed.
- A performance metric should also be **repeatable** by having the same value measured every time the same test is executed.
- The last property but not least, that a useful performance metric should have is the **easiness of its measurement**. The easiest a metric is to be measured, the better chance it has to be correctly measured, and there is nothing worse than a lousy metric that is incorrectly measured. It should be noted that most of the widely used performance metrics do not satisfy altogether the characteristics mentioned in this section.

2.2.2.2 Type of performance metrics

Most of the performance metrics we have been using are processor-related metrics. The **MFLOPS** performance metric count the number of operations that has been performed by/in a computer program when being executed (Smith, 1988). It is a throughput or arithmetic operation rate defined as the *millions of floating-point operations executed per second* giving by the formula $MFLOPS = \frac{f_n}{T_n \times 10^6}$ where f_n is the count of floating-point operations executed in T_n seconds. This metric does have a good fit with a DEM application that performs a substantial amount of floating-point operations. However, it does certainly not consider any part of the program that does not perform floating-point operations but does affect the performance. The **MFLOPS** performance metric may differ from different systems (they may not perform the same floating-point operations) and is therefore considered as **unreliable** and **inconsistent**.

The **MBytes memory bandwith** is a memory performance metric that expressed the rate at which data is read and stored to and from caches and the main memory. It is a throughput or memory rate defined as the *millions of bytes read/stored per second* giving by the formula $MBytes = \frac{B_n}{T_n \times 10^6}$ where B_n is the count of bytes memory read or stored in T_n seconds. It is a handy metric to detect whether a computer program's performances are bounded by the memory accesses (it is usually the case rather than bounded by computation).

The **execution time** is one of the most commonly used performance metrics as programmers are mostly interested in how fast a program can be executed on a system (Stewart, 2001). Version **B** of a computer program performs better than its version **A** if version **B** execution time is lesser than **A**'s. It is essential to be aware of the precision and accuracy of the time measurement method and to distinguish the difference between *wall clock* time (including system execution overhead as the time waiting for memory to be un/loaded) and the *CPU* time, which does not incorporate the time the program is context switched out while running other applications. The **execution time** can considerably fluctuate between different runs due to random events such as the operating system tasks and the cache mappings, and it is a non-deterministic metric. Nonetheless, the **execution time** is **reliable, repeatable, easy to measure, consistent, natural** and fulfill all the characteristics listed in section 2.2.2.1 and can thus be considered as a good metric.

The **speedup** is a normalized performance metric that can be derived from **execution time**. The **speedup** measures the relative performances between two systems or different versions of the same computer program. It is usually used to show the improvement in speed of execution for a parallel program but can be used more generally to illustrate the performance effect between two systems or program versions after optimization or update (Sun and Gustafson, 1991). Considering program A and program B, the speedup of program B with respect to program A is defined as $S_{B->A} = \frac{T_B}{T_A}$, where T_A and T_B are respectively the global execution time for program A and program B. Thus, if S_{B-A} is greater than 1, then program B is $S_{B->A}$ faster than program A, if not, program B is $\frac{1}{S_{B->A}}$ slower than program A.

2.2.3 Real test case

Three different test cases have been used to assess and conduct our performance study. They have been chosen to cover many aspects: large scale, application, dynamic, conversion, coupling, and industry use. The cases are:

1. **Biomass furnace**. The test case simulates the behavior of a combustion chamber of a 16 MW geothermal steam super-heater. It is a coupling XDEM-OPENFOAM case where particles are treated as discrete elements coupled by heat, mass, and momentum transfer to the surrounding gas continuous phase. The particles are taken into consideration via XDEM (Dynamic and conversion modules are active), while the gaseous phase is described by Computational Fluid Dynamics (CFD) with OPENFOAM. The case is used in section 7.2 as a baseline case to compare the performance gains between the 03/2017 and 10/2020 XDEM versions.
2. **Dam-break**. The Dam break is a famous case for two-phase flow simulations. The entire case comprises 2.35 million particles interacting with the column water in an XDEM-CFD coupling approach. It uses a multi-scale DEM-VOF method that adopts a dual-grid multi-scale approach with a coarse grid that performs the coupling between CFD and DEM code at a bulk scale, while a finer and non-uniform grid is adopted to discretize the CFD equations. The case is used in section 7.3 to study the XDEM-OPENFOAM coupling performance in a large scale simulation.

2.3 Profiling

As discussed in the introduction section 2.1, we need to use an accurate technique and procedure to measure and quantify the performance of a computer program. Profiling is a complex software analysis that measures memory utilization, the use of explicit instructions, and the frequency and length of function calls during program execution.

It is necessary to identify computational bottlenecks, and it helps developers focus their optimization efforts on the program's bottlenecks by spotting the critical sections of code.

There exist many PROFILERS that can help to identify performance bottlenecks. We have been using many of them, such as SCALASCA, ARM MAP, INTEL VTUNE, PERF, VALGRIND, GOOGLE GPERFTOOLS, PAPI, and LIKWID because they sometimes offer different performance aspects. Performance profilers can be classified into two main categories: **tracing and sampling** profilers. Some, as SCALASCA, are instrumenting profilers that are code or executable intrusive, which require modifying the source code and the compilation process. The majority of the remaining profilers are sampling profilers that let applications run without any run-time modifications, and the order of execution is not affected, and all the profiling work is done outside the application's process.

In the end, they all answer the question "How often is any method called in my code?" and "How much time does each method take?" It is then easier to identify which methods are on the top of the list in CPU or/and memory usages and then find a way to improve them. We have constantly used MAP and INTEL VTUNE in the pursuit of everyday improvement because what makes a difference is a continuous improvement over time.

2.3.1 Tracing profilers

A tracing or event-based profiler tracks and collects data from a set of predefined events during the program's execution. The events can be defined as entering or leaving a function, process communication, memory allocation for an object, or throwing an exception (Wadleigh and Crawford, 2000). They are usually trace-based, meaning that the compiler keeps track of the collected data based on events.

It is recommended to use event-based profilers when it is crucial to track specifics events. For example, one may want to track all the *return* statements occurring in a program. They are, therefore, instrumental in profoundly understanding the performance issue. However, the set of events to be tracked can be large, and so will be the generated output data (Doglio, 2015). They can also have a considerable overhead $(100 - 1000\%)$ that slows down the program's execution.

SCALASCA, EXTRAE and VALGRIND in a certain way are tracing profilers. They provide accurate call stacks, functions call (except inline functions), and the number of calls but require more time to run. A VALGRIND profiling can be 10 times slower than the normal simulation time, and EXTRAE can generate hundreds of gigabytes of trace files.

2.3.2 Statistical profilers

A statistical or sampling profiler tracks and collects data by probing the program's call stack at regular intervals using the operating system interrupts. At each interruption, the profiler determines which function is currently being executed (by using the program stack) and increases the sample count for that function (*An introduction to profiling mechanisms and Linux profilers*). The output generated by the profilers is a collection of functions and the number of times they were found being executed during the execution of the program. Each function's execution time can then be approximated by multiplying the number of occurrences by the interruption time period.

The sampling method has the advantage of having very little overhead $(5 - 15\%)$ compared to tracing profilers and produces fewer data to analyze. It allows the program to be executed at almost the usual execution time. It is recommended to use sampling profilers at first to get a glimpse overview of the program execution and detect the hotspots. In return, it gives less accuracy in the output information as it uses statistical approximation (Mytkowicz et al., 2010).

ARM MAP, INTEL VTUNE, PERF, GPERFTOOLS and LIKWID are sampling profiling tools that also support the code instrumentation. They have a low memory footprint, small trace files, and do not change the application time. The call stacks and the function calls are inaccurate, as not all of them are captured (depend on the sampling period).

2.4 Benchmarking

To solve a given problem, we are sometimes faced with different algorithms and implementations as a solution. For example, there are more than fifty algorithms for solving collisions' detection in a DEM simulation. We may also use different data structures to implement those algorithms on different computer systems. Therefore, a given problem must compare the performances (using performance metrics) of some of the available methods to make a suitable choice.

Benchmarking the performance of different methods or different computer systems is the procedure that consists of comparing their performance against a standard method or computer system using a range of performance metrics and evaluation criteria. It is defined to be the *"systemic measurement of some aspect of a computer system's performance"* (Berry, Cybenko, and Larson, 1991).

2.4.1 Macro and micro benchmarks

In our current work, we have been conducting two kinds of benchmarking: **micro** and **macro** benchmarks. A **micro** benchmark aimed to focus on the performance of a particular and specific computer program section. As it is introduced in chapter 3, we developed a benchmark on the broad-phase (see Fig. 3.3) of the collision detection process of the XDEM software. It is a neutral benchmark as it only compares existing algorithms with enhanced implementations. In a general manner, we benchmark the hotspots functions detected during profiling and presented in section 2.6.1. The goal is to assess the following statement: *"This particular implementation of this function benchmark of a given size on this given computing node executes in this particular time using the compiler with this level of optimization."* (Hockney, 1996).

A **macro** benchmark, on the other hand, tests the performance of the whole application or system. In chapter 7, we present a **macro** benchmark of XDEM applied to a real test case. We compared the performance of two versions or states of XDEM: the baseline version (03/2017) presented in section 2.2.1 and the very last version (10/2020) defined as the version having all the optimizations.

2.4.2 Evaluation criteria

Comparing methods in a benchmark is based on at least one performance metric (Weber et al., 2019). The latter requires at least one of the characteristics detailed in section 2.2.2.1 to make it a good performance metrics. Unfortunately, measuring such metrics leads to uncertainties that are considered as errors or noise. There are many sources of errors introduced during measurements, such as precision and accuracy, or errors due to experimental mistakes. They are classified into two categories: systemic (experimental) and random errors. It is, therefore, important to understand and take them into account before drawing any conclusion. Even though it is almost impossible to quantify the systemic errors since it is a function of bias, it is essential to use a model for random errors (Gaussian) to quantify the precision and the repeatability of the measurements.

2.5 Performance models

The performance model generates knowledge about software-hardware interaction. Its main purpose is to come up with a quantitative estimate for expected performance. Without an expected performance estimate, it is impossible to decide on performance optimizations as there is no clear knowledge of what aspect of software/hardware interaction limits the performance and what could be the optimal performance. You formulate a model to estimate expected performance and compare this to application benchmarking. Additionally, performance profiling may be used to validate model predictions. In case the validation fails, either the profiling or performance measurement is wrong, the model assumptions are not met, or the model inputs are wrong.

2.5.1 Roofline model

It is important to consider the computer system architecture when evaluating the performance of a computer program. As it can perform differently depending on the architectural characteristics. Therefore, there is a need for programmers to have a performance model that helps them understand which performance they can expect from a given architecture. Statistical and stochastic models are available models (Tikir et al., 2007; Boyd et al., 1994) that can be used to precisely foresee a program performance on multi-core architectures. However, these models do not provide any understanding of the reasons for an underperforming program.

The roofline model is a visual performance model used to provide perceptive performance evaluations of a given compute kernel (Williams, 2009). It gives an insightful and visual representation of the program's intrinsic bounds and possible optimizations on multi-core CPU or GPU architectures. The roofline model is unique to each architecture and integrates in-core performance, memory bandwidth, and locality into a single, easy-to-understand performance figure.

The roofline model uses the **operational intensity** to measure the traffic between the caches and the **DRAM** to include memory optimizations into the model's bound and bottleneck. It is defined as the ratio of the **Work** to the memory traffic **Q** and express the number of operations per byte of memory traffic:

$$I = \frac{W}{Q} \tag{2.1}$$

, where W is the work defined as the number of operations performed by the compute node and Q, denotes the number of bytes of memory transferred by the compute node during an execution. The operation defined in the model can be one of the metrics introduced in section 2.2.2 or any operation, such as the number of integer or floating-point operations. The roofline model can capture other performance ceilings other than simple peak bandwidth and performance, such as *instruction level parallelism (ILP)*, *single instruction multiple data (SIMD)* and *Balance floating-point operation mix* peaks that give hints for the programmer on which optimization to focus on.

The Fig. 2.2 is a roofline graphic representation of the brute force algorithm used in collision detection. The x-axis shows the **arithmetic or operational intensity** measured in the number of floating-point operations (FLOPs) or/and integer operations (INTOPs) per byte, and y-axis shows **performance** measured in billions of floating-point operations per second (GFLOPS) or/and billions of integer operations per second (GINTOPS). The diagonal chart lines indicate memory bandwidth limitations preventing loops/functions from achieving better performance without some form of optimization. The L1 diagonal chart line indicates the L1's bandwidth maximum amount of work that can get done at a given arithmetic intensity if it always hit the L1 cache. The horizontal chart lines indicate the compute capacity limitations preventing loops/functions from achieving better performance without optimizing. The *Scalar Add Peak* represents the peak number of addition instructions that can be performed by the scalar loop under these circumstances. The *Vector Add Peak* represents the peak number of addition instructions that can be performed by the vectorized loop under these circumstances.

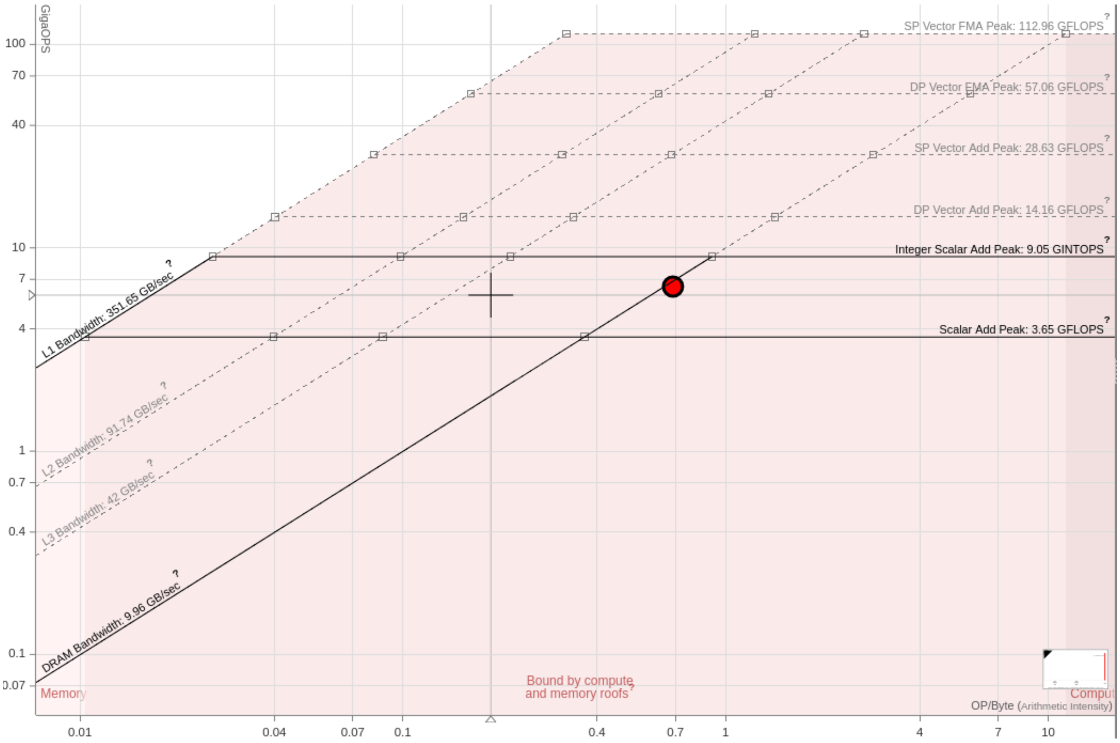


FIGURE 2.2: The roofline model for the naive brute force algorithm in collision detection. The roofline was generated using the `Intel advisor` tool.

From Fig. 2.2, we can state that our brute force approach is a memory-bound algorithm as the red dot is positioned below the **DRAM** diagonal chart line. It indicates that our program misses too much often the cache lines when fetching data. It can be expected as the objects are stored without any particular order, which favors a lot of memory jump. A possible solution to improve the algorithm's performance is to apply a space-filling curve approach to store the objects based on their position in the space. Objects spatially close to each other should be stored close to each other in the memory. The red dot is also positioned below the *DP Vector Add Peak* but above the *Scalar Add peak* horizontal line, indicating that the loops are vectorized but bound by the *DRAM* memory accesses.

2.5.2 Execution-Cache-Memory

The ECM (Execution-Cache-Memory) performance model is a resource-based analytic performance model. It can predict the runtime of serial straight-line code (usually an innermost loop body) on a specific processor chip. Runtime predictions are based on maximum throughput assumptions for instruction execution and data transfers, but refinements can be added. The model, in its simplest form, can be set up with pen and paper.

The model decomposes the overall runtime into several contributions, which are then put together according to a machine model. A processor cycle is the only time unit used. All runtime contributions are provided for several instructions required to process a certain number of (source) loop iterations; we typically choose one cache line length (e.g., eight iterations for a double-precision code), which makes sense because the smallest unit of data that can be transferred between memory hierarchy levels is a cache line. For simple calculations, bandwidths and performance are consistently specified in “cycles per cache line.” However, this choice is essentially arbitrary, and one could just as well use “cycles per iteration.” Unless otherwise specified, an “iteration” is one iteration in the high-level code.

2.5.2.1 In-core model

The primary resource provided by a CPU core is instruction execution. Since instructions can only be executed when their operands are available, the ECM model’s in-core part assumes that all data resides in the innermost cache. It further assumes out-of-order scheduling and speculative execution to work correctly so that all the instruction-level parallelism available in the code can be provided by the hardware, resources permitting (on a given microarchitecture) (Stengel et al., 2015).

In practice, the first step is to ignore all influences preventing maximum instruction throughput, such as:

- Out-of-order limitations
- Instruction fetch issues
- Instruction decoding issues
- Complex register dependency chains

2.5.2.2 Data transfer model

Data transfers are a secondary resource required by code execution. Modeling data transfers starts with analyzing the data volume transferred over the various data paths in the memory hierarchy. Some knowledge about the CPU architecture is required for this in order to know what path cache lines take to get from their initial location into L1 cache and back. It is assumed that latencies can be perfectly hidden by prefetching, so all transfers are bandwidth limited. Additional data transfers from cache conflicts are neglected (it is still possible to extend the model to account for additional transfers). With known (or measured) maximum bandwidths, the data transfer analysis results in additional runtime contributions from every data path 2.3. How to put together these contributions and with the in-core execution time is part of the machine model. The two extreme cases are:

- Full overlap: The predicted execution time is the maximum of all contributions.
- No overlap: The predicted execution time is the sum of all contributions

There is a large gray zone in between these extremes. On most modern Intel Xeon CPUs, data transfer times must be added up, and everything that pertains to “work” (i.e., arithmetic, loop mechanics, etc.) overlaps with data transfers. It yields the most accurate predictions on these CPUs, but other architectures behave differently.

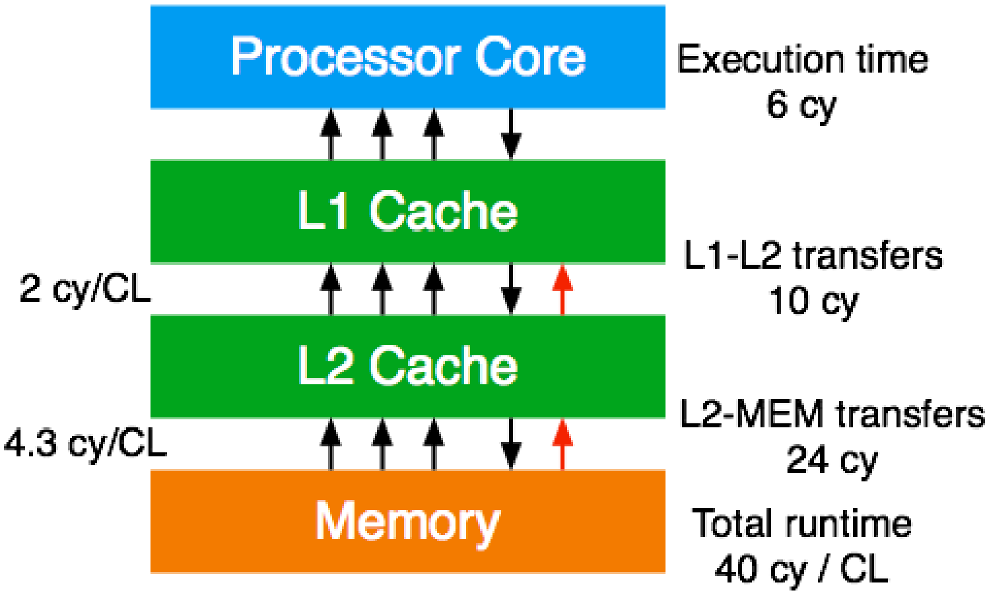


FIGURE 2.3: Schönauer vector triad (left column = bandwidth of data path, black arrow = cache line transfer, red arrow = write- allocate cache line transfer) (Erlangen Regional Computing Center, 2019)

2.6 Application

In this section we evaluate the performance of our baseline version (03/2017) defined in section 2.2 by profiling the reference test case presented in section 2.2.3. Our preeminent objective is to identify the parts of the code that need to be optimized and redesigned by improving the algorithms, using the appropriate data structures and parallelization techniques.

2.6.1 Hotspots analysis

The **hotspots** analysis provides a deep understanding of an application flow and identifies the functions where the code is mostly executed: **bottlenecks**. We used available profiling tools evoked in section 2.3 to generate a sampling profile as at this stage, we are only interested in quantifying how *fast* is the XDEM program. The results in tab 2.1 presents the seventh most time consuming functions and was gathered using INTEL VTUNE's user-mode sampling method on an INTEL XEON CPU at 3.40GHz. The execution time were collected running with a pure dynamic XDEM **optimized** version compiled with debug info $-g$ and the $-O3$ level optimization in sequential.

TABLE 2.1: Hotspots in XDEM using INTEL VTUNE.

Index	% Total	Self (s)	Children (s)	Function name	Parent index
1	99.66	0.0	835.032	*Run_simulation*	1
2	91.50	0.0	773.460	*Collision_detection*	1
3	30.33	2.64	392.671	*Interaction_models*	2
4	27.36	0.77	168.773	*Narrow_phase*	2
5	15.44	5.69	186.354	*Broad_phase*	2
6	14.23	4.75	71.865	*Reset_pair_interaction_list*	2
7	13.97	0.19	154.509	*Save_interactions_history*	3

The first column shows the function's index, and the second column shows the proportion of the CPU run time of each function to the total CPU run time of the entire simulation. It is fair to say that most of the time ($\sim$ 92%) is spent in the particle interactions process. It is no surprise as we were running a pure dynamic version with particle-particle and particle-wall interactions.

The third column shows the running time of the function executing itself, and we can see that most of the functions spend very little time doing so. Furthermore, the reason is clarified in **CPU Time:Children** column, which indicates the time spent by the function requesting others' functions. The function *Collision_detection* does not perform any computation but rather calls other functions that themselves mostly call some core functions not displayed in tab 2.1.

The last column **Parent index** specifies the **caller** and **callee** relationships between functions. Among the **callee** functions of *Collision_detection*, three of them appears to be the most time-consuming and are executed in the code in this order *Broad_phase* → *Narrow_phase* → *Interaction_models*. They are closely tied as the first step of detecting collisions in a DEM simulation is the *broad-phase*, which finds out the closest pairs of particles that are possibly in contact. The *Narrow_phase*, on the other hand, returns the pairs of particles that actually are in contact using the previous phase results. The *Interaction_models* applies all the defined models between pair of particles as the **contact, attraction, impact, bond, and rolling** models defining how the particles behave when in contact with each other or the walls. Actually, this function may not be the most time-consuming since it is not necessary to have all the interaction models defined; most of the time, only the **impact** and **contact** models are defined. Those three DEM procedures are explained in more detail in chapter 3.

A dynamic call-graph, as represented in Fig. 2.4, is a profiling graph (tree view) that visualizes the calling relationships between the functions during the execution of a computer program. In Fig. 2.4, each node represents a function of XDEM and each edge between any node A and B indicates a **caller** and **callee** relationships. Surely not all function relationships are shown in the call-graph but only the most important one that provides hints on where and what to look at.

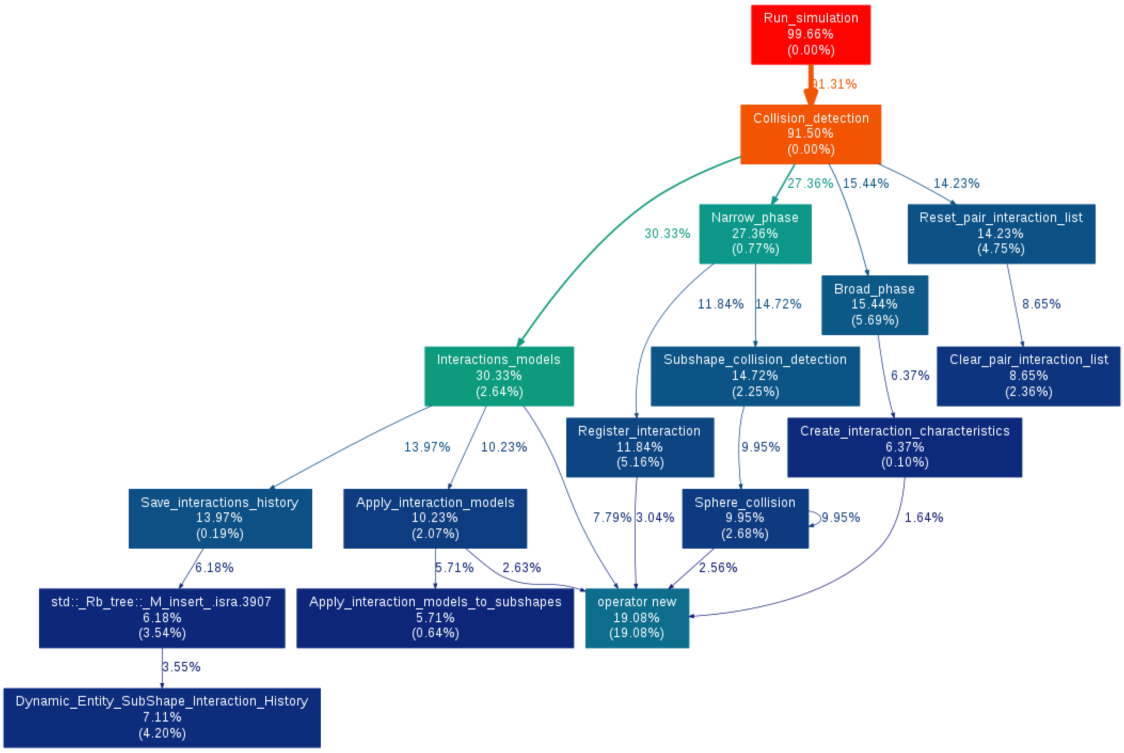


FIGURE 2.4: Call-graph profile of XDEM. The redder the node is, the more computational time it consumes

In the Fig. 2.4, the redder the node is, the more computational time it consumes. We can then identify a *hot path* connecting the redder nodes going from top to down the tree. As a result, we can spot what is probably an abusive use of `map`'s **insert** function or even more as an improper usage of C++ `map` data structure in *Save_interactions_history* function. The program also spent 10% of the running time dynamically allocating memory in the heap with the `C++ new operator`. It overall indicates the reason why the *Interaction_models* function is time-consuming is mostly an implementation and memory issues.

We are aware of the limits of such a call-graph approach on a complex program alike XDEM that uses dynamic method binding (Spivey, 2004). In dynamic binding, the compiler is not able to resolve the call at compile-time, and the binding is known at run-time such that the profiler is unable to capture the call.

2.6.2 Memory footprint analysis

As we have spotted in the previous section 2.6.1, memory management can be a major issue in computer program performance. Usually, a **hotspots** or **call-graph** are not suitable for detecting bad memory usage or management and is therefore necessary to profile the memory itself using a memory profiler.

2.6.2.1 Heap and stack profiling

We have been using `valgring` and `massif` tools to measure how much heap memory the XDEM program uses. It provides information about heap blocks and stack sizes, which are very useful to track memory usage or leaks (when memory is allocated but not used).

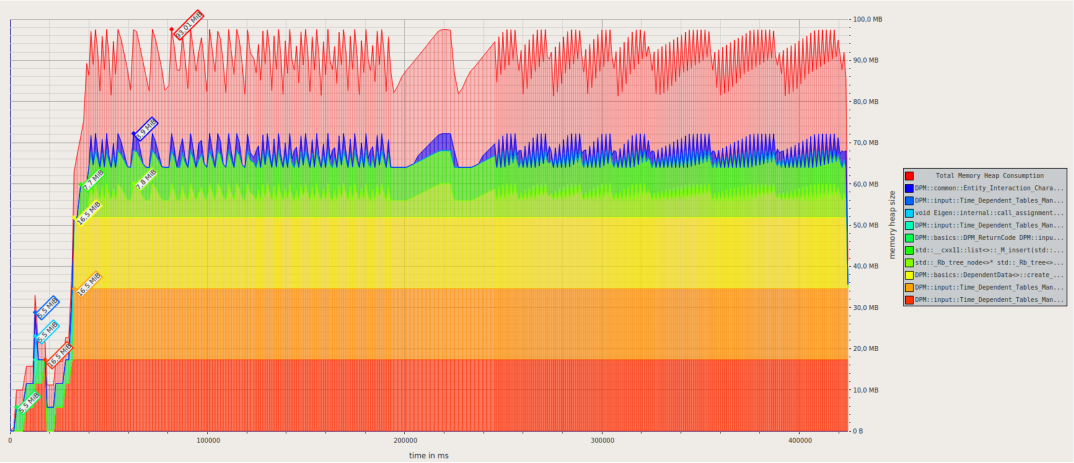


FIGURE 2.5: Heap profile of XDEM generated by `valgrind`'s `massif` tool.

Fig. 2.5 shows the different memory allocations in the heap during an XDEM simulation using our baseline. Part of the graph's fluctuations is due to the sampling period where `valgrind` collects the data and finds out which functions call the memory allocator. The other reason is that the XDEM application allocates and releases memory at each iteration, calling the memory allocator very often. We back-traced the allocation/deallocation process to the function, which updates the interaction history list (it is used for some integration model) after every iteration: the previous interaction history list is freed, and a new one is created.

Fig. 2.5 shows how our baseline case allocates memory in the heap at the peak of 93MiB after initialization. It is by any means, not an excessive use of the heap. However, what is surprising in this figure is that the application allocates three times the same size, 16.5MiB of memory, and it appears from the timeline that the latter two are copies of the primary allocation. The first allocation was backtraced to initialization, where a time table that governs the grate motion is loaded from the input file. The two copies were made in functions using the time table data without passing it by reference.

The tab. 2.2 presents the memory call-graph at the end of an XDEM simulation and shows the functions that load the timetable for boundaries motion. The **useful-heap** is defined as the memory that is allocated for the time table and, the **extra-heap** is defined as the memory allocated for the book-keeping.

Percent (%)	**time (ms)**	**total (B)**	**useful-heap (B)**	**extra-heap (B)**	**Function**
56, 32%	424, 631	52, 385, 992	51, 304, 4932	1, 081, 499	**read_vec3D_tables_from_file**

TABLE 2.2: Time table memory allocation details.

The storage of the time table data with the two extra-copies that were made represents more than 55% of the total memory allocated in the heap. Preventing the extra-copies decreases the time table memory footprint to almost 20%.

2.6.2.2 Memory leaks detection and cache profiling

One of the most challenging bugs to detect and that can cause serious performance issues is memory leaks. It happens when a program allocates memory and does not release it when it is no longer needed or can not access it. It has consequences of slowing down the running program by reducing the amount of available memory and eventually leading to the program crash when no more memory is accessible. A full memory leak check with `valgrind` tool confirmed as in section 2.6.2.1 that there no such leaks in XDEM program. We have regular daily and nightly memory leak tests that detect any leak as soon as they are introduced. All memory allocated in the heap, mostly with the `new operator`, are released when not needed anymore by the `delete instruction`. Using the C++ smart pointers approach is a good practice that simplifies memory management.

The **cache memory** is a small and extremely fast memory that acts as a buffer between the **RAM** and the **CPU**. It should be used to store and access frequently used data and instructions so that the **CPU** can access them immediately when required. The more the **CPU** can access data from the **cache memory**, the less time it takes for data to be accessed. A **cache performance** can be defined in order to be able to assess the performance of the cache access by characterizing two quantities:

- A **cache hit** occurred when the **CPU** reads the requested data directly from the cache.
- A **cache miss** occurred when the **CPU** does not find the requested data in the cache location and thus copies the data from the main memory to the cache before processing the data.

The **hit ratio** defined as the ratio of cache hits to the sum of the number of cache hits and cache misses is commonly used to measure the **cache performance**.

In tab 2.3, the first column lists the branch prediction and cache level instructions, while the second column is the count of missed or mispredicted instructions during the program execution. The last column is the mispredicted branch and missed caches **rate** defined as the instructions count ratio to the missed count. Two types of branches prediction and caches are shown in the table as returned by the `valgrind` tool. A **conditional** branch instruction only branches to a new address if a specific condition is true while an **indirect** branch instruction branches to a specified address. **Conditional** and **indirect** branch are not mutually exclusive, a branch can be **conditional** and **indirect**. $L1$ corresponds to the first 1^{st} cache level and LL to the last cache level, gathering the second 2^{nd} and third 1^{rd} cache levels (when $L2$ and $L3$ are available).

Summary	**Miss[-]**	**Miss rate[%]**
$L1$ Instructions fetch ($l1mr$)	$10,719,727$	0.024
$L1$ Data read ($D1mr$)	$410,790,558$	0.913
$L1$ Data write ($D1mw$)	$143,380,535$	0.318
$L1$ Miss Sum ($L1m = l1mr + D1mr + D1mw$)	$564,890,820$	1.255
LL Instructions fetch ($lLmr$)	$550,563$	0.001
LL Data read ($DLmr$)	$74,892,997$	0.166
LL Data write ($DLmw$)	$66,163,592$	0.147
LL Miss Sum ($LLm = lLmr + DLmr + DLmw$)	$141,607,152$	0.314
Conditional Branch (Bcm)	$316,484,354$	0.703
Indirect Branch (Bim)	$107,703,340$	0.240
Sum Branch ($Bm = Bcm + Bim$)	$424,187,694$	0.942

TABLE 2.3: Cache data access summary.

We can notice from tab 2.3 that there is a significant cache miss sum ($\sim 1.3\%$) that possibly represents a significant overhead. Our XDEM program is cache and memory-bounded though it is not uncommon for computer programs because the memory access is costlier than computation in modern CPUs, and designing a cache-friendly program is a very tedious job. It is also not surprising since the test case was running in sequential using only one core on one CPU, and sometimes there is nothing more we can do with serial code due to usual CPU limitations as pipelining and memory bandwidth. Parallelization is then very useful to overcome those limitations.

The mispredicted branches in tab 2.3 count every time the CPU predicted the results of conditional branches, and the predictions have proved to be incorrect. The misprediction has cycles penalty costs because it stalls the CPU (mispredicted instructions are discarded, and new correct predicted instructions are loaded in the execution pipeline). Once identified and when possible, making the branches more *"predictable"* or simply avoiding branches will increase performance.

2.7 Summary

In this chapter, we have presented the process of evaluating XDEM software performances. This step is crucial as it states the code's initial conditions and will serve as a baseline comparison for the upcoming optimizations.

We presented the profiling results in section 2.6 were the main hotspots of our XDEM program have been identified. In a pure dynamic simulation, these hotspots are mainly located in the collision detection processes. Section 2.6 also introduced a call-graph representation of XDEM which describes the caller and callee relationships among functions. We have spotted particles' collision detection as the most consuming computational time segment for the test case we defined in section 2.2.3. The process, divided into two phases, the broad and narrow-phases, represents a combined $\sim 33\%$ of the total running time. Half of the simulation time is then spent applying the bond, contact, impact, and rolling models. Those parts are therefore defined as our main priorities for optimization, redesigning, refactoring, and remodeling. The next chapters will, therefore, focus on optimizing the collision detection processes.

In the next chapter 3, we reviewed in detail the collision detection process of a DEM simulation. We presented and compared the different algorithms commonly used and propose an optimized algorithm that gives the best results for our XDEM approach. For this purpose, we have developed a benchmark framework to test and compare the collision algorithms' performance most often found in the literature.

Chapter 3

CONTACT DETECTION IN THE EXTENDED DISCRETE METHOD (XDEM)

3.1 Abstract

Extended Discrete Element Method (XDEM) is a multi-physics approach that extends the classic Discrete Element Method (DEM) by attaching chemical and thermodynamic state to the particles. One of the essential and most computation-intensive parts of XDEM is the interaction/collision detection phase during which objects in-contact or in close-range are identified. This work evaluates and compares ten different Broad-Phase Collision Detection algorithms while considering a large DEM test case. It appears that the choice of the best algorithm is a trade-off between many criteria, including the size of the search space, the number of particles, and the memory usage [a].

Highlights

- Contact detection in DEM is presented.
- Performance of Broad-phase is investigated in contact detection.
- Broad-phase algorithms comparison.
- Test on a real test case.

---

[a] A short version of this chapter was published in AIP Conference Proceedings (Rousset et al., 2018)

3.2 Introduction

The Discrete Element Method (DEM) is a *Lagrangian* approach that models granular materials by representing every single particle as an independent and distinct entity. It is used to numerically determine the displacements of a large but finite number of particles taking particle interactions into account. The eXtended Discrete Element Method (DEM) is a multi-physics framework that extends the classic DEM approach by assigning a chemical and thermodynamic state to the representation of the particle. It is an iterative computation method that simulates the motion of particles and their chemical/thermodynamic state. DEM also supports coupling with Computational Fluid Dynamics (CFD) (Mahmoudi et al., 2016c) or Finite Element Method (FEM) (Michael and Peters, 2013; Michael, Nicot, and Peters, 2013).

The entire system behavior in a DEM simulation results from the respective interactions between particles. It provides the desired results. The contact or collision detection is a memory and computation expensive process that is therefore fundamental to identity all the collisions between the particles so that the model works and requires to check the distance between all the particles in the system. For the sake of optimization and readability, the contact detection process is usually, as in our XDEM software, subdivided into two parts: the **broad** and **narrow** phases. As shown on Fig. 3.1, within each time-step, the *Interaction* step is typically divided into two phases:

- The *Broad-Phase* interaction process identifies the pairs of particles that might interact with each other within the whole system. It can lead to a hefty amount of computation. For that reason, the broad-phase algorithms work on approximate objects such as bounding boxes to quickly generate an approximate list of colliding particles. This list includes every pair of colliding objects, but it may also include pairs of objects whose bounding boxes intersect but are still not close enough to collide. To account for close-range interactions (e.g., radiation), the bounding boxes are extended by the value of the predefined interaction range.

- The *Narrow-Phase* will work on an approximate list of colliding particles (returned by the broad-phase) and perform precise calculations according to particle geometry.

During the *Integration* step, all interaction resultants (forces, torques, heat flux, species fractions, etc.) coming from the different particles are accumulated and integrated to calculate the new state the particle for the next time-step.

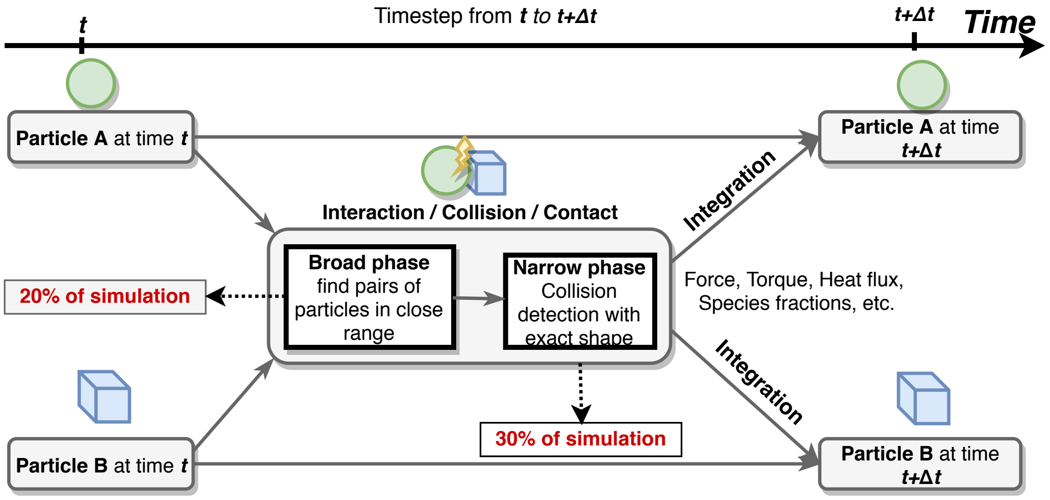


FIGURE 3.1: Workflow of an XDEM simulation: the *Broad-phase Collision Detection* finds the pairs of particles that may interact with each other by replacing the particles real shape with bounding volume and it returns a list. The latter is used in the *Narrow-phase Collision Detection* to perform a collision detection with particles real shape.

As spotted in section 2.6.1 of chapter 2, one of the essential and most computation-intensive part of XDEM is the *Broad-Phase* phase (the percentage greatly depends on the test case) during which objects in-contact or in-close-range are identified. Combined with the *Narrow-Phase*, it takes about 50% of the entire XDEM computational work. Therefore it is crucial to take a close look at the collision detection process and make a clear decision of which algorithm or approach to single out.

The current chapter is structured as follows: We first introduce the commonly used broad-phase algorithms with their specificities, limitations, and advantages in section 3.4. Secondly, the design of our benchmark framework is presented in section 3.5. Finally, the results of sequential and parallel runs are described and discussed in section 3.5.

3.3 Broad-phase collision detection

Collision detection is an ongoing research and optimization source in many fields, including video games and numerical simulations (Coumans, 2015; Jiménez, Thomas, and Torras, 2001). Collision detection aims to report geometric contact when it is about to occur or has occurred. Unfortunately, precise and exact collision detection for large amounts of objects represents an immense amount of computations, naively n^2 operations with n being the number of objects (Lin, 1997). To avoid and reduce these expensive computations, the collision detection is decomposed into two phases as shown in Fig. 3.1: the *Broad* and *Narrow* phases.

The broad-phase's primary goal is to prune away pairs of particles that are too far away from each other and thus have no chance to collide. It aims to quickly report if two particles **do not** intersect and therefore must scale very well with the number of particles in the system to make sure the time complexity is under $O(n^2)$. The broad-phase replaces the real particle shape (and usually complex shape such as polygons) to perform an upper bound for the collision to achieve this low complexity. The particle shape is encapsulated inside the bounding volume (see Fig. 3.2) such that if two bounding volumes do not cut across, then their real shapes do also not intersect. There are many popular bounding volumes used in the literature as the **sphere, bounding boxe** or **convex shape**.

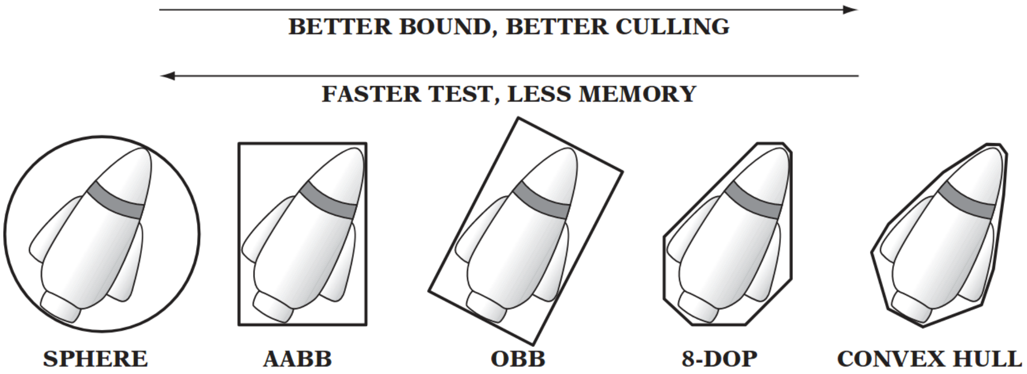


FIGURE 3.2: Type of bounding volumes used in broad-phase: sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull (Ericson, 2004).

They offer different complexities and accuracy; a **sphere** bounding volume offers a bad bound (for non-spherical shapes) as it does not match very well the particle's actual shape but provides a fast check and needs little memory to be stored (only the center coordinates and the radius). On the other hand, bounding volumes such as superquadrics shapes are more accurate and return a result similar to the real shapes but have a more elaborate check algorithm and require much more memory storage.

3.3.1 Axis-Aligned Bounding Boxes (AABBs)

The most popular bounding volume found in the literature is probably the *axis-aligned bounding volume*. It has a simple parallelepiped shape and has always it faces aligned with the axes, i.e., the edges of the box are parallel to the (Cartesian) coordinate axes (Schneider and Eberly, 2002). The assets of an AABB volume are its memory efficiency, as there is only a need to store *min and/or max points* and the *radius* to represent the volume. It is also computationally efficient as it has a quick overlap check that simply involves the direct comparison of individual coordinate values. Two AABBs intersect if and only if they overlap on the three-axis. It is, therefore, straightforward to discard pairs of particles that do not intersect. If the AABB has a *min-max points* representation, the overlap check will look like as follow:

```cpp
bool OverlapAABBs(AABB box1, AABB box2)
{
  //Check intersect on X Axis
  if( box1.max[0] < box2.min[0] || box1.min[0] > box2.max[0] )
    return false;
  //Check intersect on Y Axis
  if( box1.max[1] < box2.min[1] || box1.min[1] > box2.max[1] )
    return false;
  //Check intersect on Z Axis
  if( box1.max[2] < box2.min[2] || box1.min[0] > box2.max[2] )
    return false;
  //Overlap on the three Axis
  return true;
}
```

LISTING 3.1: C++ snipped code of intersection of two axis-aligned bounding boxes (Ericson, 2004).

3.3.2 Oriented Bounding Boxes (OBBs)

Once more, AABBs do not perfectly match a real particle shape, hence the introduction of OBBs. Unlike AABBs, OBBs edges are not aligned with the axis coordinates. It is a parallelepiped volume with an arbitrary orientation. There is much possible representation of OBBs, the most commonly used representation has a center point, a matrix orientation, and three half-edge lengths.

```
class OBB
{
  Point c;        // The OBB center
  Vector u[3];    // Local axis coordinates
  Vector e;       // Halfwidth along each axis
}
```

LISTING 3.2: C++ snipped code of oriented bounding boxes data structure (Ericson, 2004).

The intersection of OBBs is more complicated than for AABBs, and the algorithm can be found in the reference book of Christer Ericson (Ericson, 2004).

3.3.3 Bounding Spheres

The sphere is also a ubiquitous bounding volume convenient as it is memory and computationally efficient. It also has a short overlap check, few branches compared to AABBs, and only needs to compute the two-sphere centers' distance.

```
bool OverlapSpheres(Sp sphere1, Sp sphere1)
{
  //Compute distance between the two centers
  float distance = (sphere1.center − sphere2.center)*(sphere1.center − sphere2.
      center);
  float radiusSquared = (sphere1.radius + sphere2.radius)*(sphere1.radius +
      sphere2.radius);
  //Check the intersection
  return distance <= radiusSquared;
}
```

LISTING 3.3: C++ snipped code of intersection of two spheres (Ericson, 2004).

The bounding sphere can be obtained by first calculating the AABB of the object. The sphere is then deduced by choosing the midpoint of the AABB as the center of the sphere and the midpoint's distance to the farthest point as the radius of the sphere. Many more elaborated algorithms try to bound the object as much as possible in a very efficient manner (Ritter, 1990).

3.4 Broad-Phase algorithms

For this study, ten different Broad-phase collision detection algorithms have been considered covering grid, tree, and sorting approaches. Some of those algorithms were implemented from scratch, and the remaining were taking from open-source libraries. The *Bullet* (Coumans, 2015), *CGAL* (Fabri and Pion, 2009; Zomorodian and Edelsbrunner, 2000), and the flexible collision library (*FCL*) (Pan, Chitta, and Manocha, 2012) frameworks have therefore been included in our benchmark framework. When available, the studied algorithms' implementation is directly taken from their authors' book or article. Alternatively, we also proposed our implementations directly from the description or the algorithm of an enhanced version of their given and available implementations.

As referenced in tab. 3.1 Different tree based algorithms have been implemented such as *Octree*, *Loose Octree*, and *Kd-tree* (Ericson, 2004). About grid based algorithms, *Uniform grid* and *Hierarchical grid* (Pabst, Koch, and Straßer, 2010; Ericson, 2004) have been implemented and finally the *Sweep and Prune* (Ericson, 2004) algorithm for the sorting based approach. For comparison, the naive approach by *Brute Force* has also been implemented.

As stated in section 3.3, the broad-phase algorithms do not rely on the particles' real shape to perform a collision check. The main purpose is to avoid the complex shape of particles that can affect the algorithms' efficiency. Therefore it is reasonable to use a bounding volume that encapsulated the particle to primaryorm a fast overlap check. Bounding sphere and axis-aligned bounding boxes have been used in our current implementations for all the algorithms described below.

Algorithms	Implementation	Parallelization
Brute force	Scratch	Yes
Octree	Scratch	No
Loose Octree	Scratch	No
Kdtree	Scratch	No
Hierarchical grid	Scratch	No
Uniform grid	Scratch	Yes
Sweep and Prune	Scratch	Yes
Bullet	Bullet library	No
CGAL	CGAL library	No
FCL	FCL library	No

TABLE 3.1: Algorithms that have been implemented and parallelized in the C++ framework. The brute force, the grid and tree algorithms were implemented from scratch. The CGAL and Bullet libraries use spatial partitioning algorithms while FCL was used with a AABB tree based algorithm.

3.4.1 Spatial partitioning

Apart from the bounding volumes, the broad-phase algorithms rely on the domain properties like the **locality, and partitioning** and we can distinguish three types of the broad-phase algorithm that rely on a partition method: *spatial partitioning and sorting*, , *grids*, and *tree* structures.

The **spatial partitioning** techniques operate by dividing space into many regions that can be quickly tested against each object (Lubbe et al., 2020). Two objects possibly intersect if only they are contained in the same region of space; there is no need to check overlap between objects that do not overlap the same region (Glass, 2005). The number of pairwise is therefore drastically reduced, and the $O(n^2)$ operations (or complexity) is down to something more manageable ($O(\log n)$, $O(n)$, or $O(n \log n)$). Two main types of spatial partitioning will be considered: grids and trees.

The **spatial sorting** is a topological method based on the position of particles relative to the others. The algorithm consists of a sorted spatial ordering of objects. Axis-Aligned Bounding Boxes (AABBs) are projected onto x, y, and z axes and put into sorted lists. By sorting projection onto axes, two objects collide if they overlap on the three axes. This axis sorting reduces the number of pairwise tested by reducing the number of tests to perform to only pairs, which collide on at least one axis. The number of operations or complexity is estimated at $n \log n$.

The **grid-based** algorithms consist of a spatial partitioning process by dividing space into regions/cells and testing if objects overlap the same region of space. Furthermore, this reduces the number of pairwise to test.

The **tree-based** algorithms use a tree structure where each node spans a particular space area. It reduces the pairwise checking cost because only tree leaves are checked. The number of operations or complexity is estimated at $\log n$.

3.4.2 The Brute Force approach

The **brute force** algorithm is the very naive and simplest way of proceeding a collision detection. Every particle is checked for collision with every other particle in the system (see Fig.3.3). Therefore, it is not an efficient approach and should not be used for large systems with a substantial number of particles.

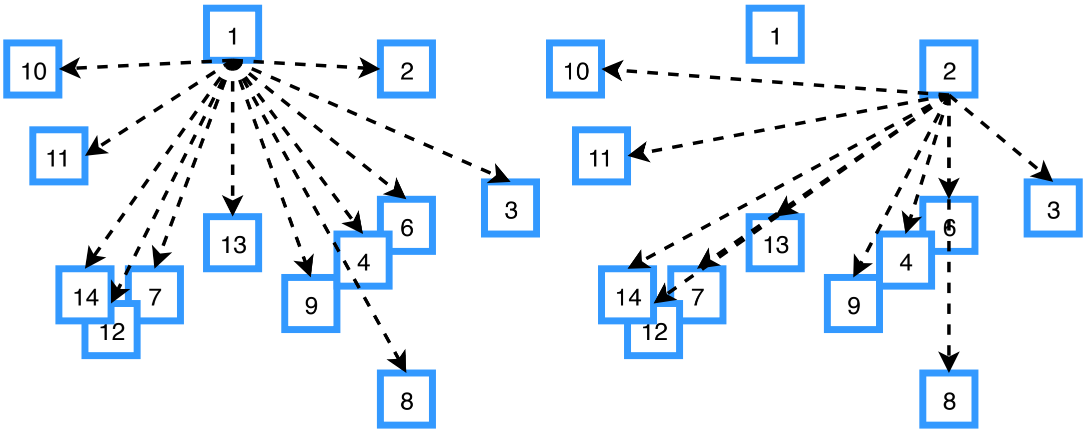


FIGURE 3.3: Check object one against all other objects, then object two against all other objects (except object one).

This algorithm has a complexity of $O(n^2)$ and scales quadratically with the number of particles as it loops twice through all the system particles.

3.4.3 The Octree and k-d tree

The *octree* is the common tree-based spatial (axis-aligned hierarchical) partitioning method in 3D, where each node has eight children. The root node is usually the bounding volume divided into eight equal-size sub-cubes, also called octants, by dividing equally in half the x, y, and z-axis. These octants form the child node of the root node. Recursively, each octant is divided in the same manner as the root node (see Fig. 3.4). It is, by definition, a divide and conquers algorithm. A criterion for stopping the recursive construction of an octree is the definition of a *maximum depth* or a *minimum size* for the octants.

The analogous structure to the octree in two dimensions is known as a quadtree.

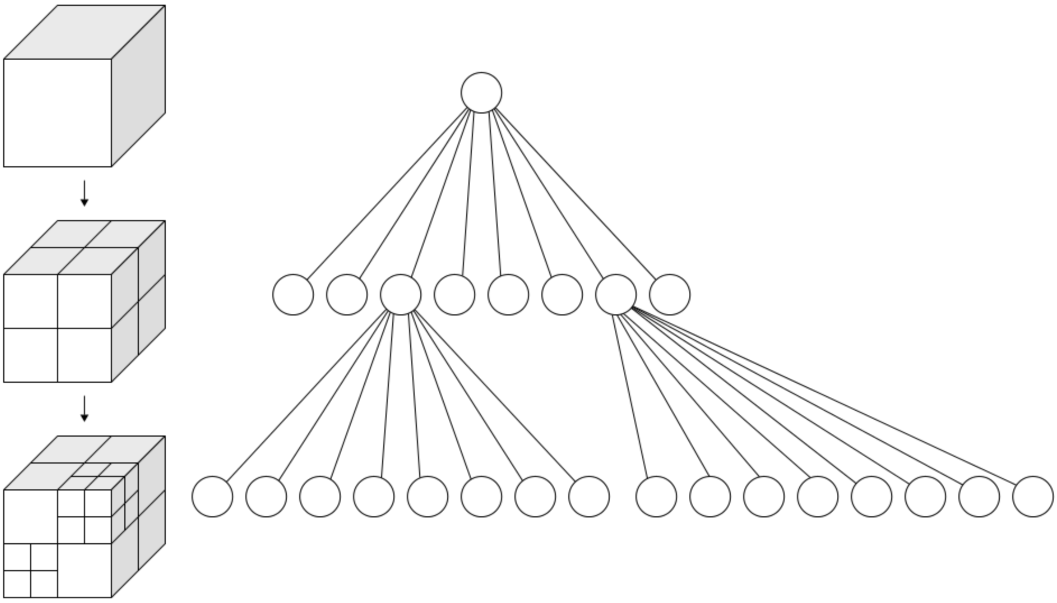


FIGURE 3.4: Left: Recursive subdivision of a cube into octants. Right: The corresponding octree (Truong, Arikatla, and Enquobahrie, 2019).

In practice, most of the octrees do not exceed five levels as a complete octree of n levels has $\frac{8^n-1}{7}$ nodes, that is, to say it can overgrow. An octree node data structure can be implemented as shown in listing 3.4 containing a center point of the node, an octree node pointers point to the eight leaves children and a list of particles in the current node.

```
class OctreeNode
{
  //Node center
  Vector3d center;
  //Node pointers to the eight children nodes
  OctreeNode *child[8];
  //Pointer to the particles contained in the current nodes
  Particle *parts;
}
```

LISTING 3.4: C++ snipped code of intersection of an Octree node data structure (Ericson, 2004).

The octree pointers representation in listing 3.4, although it uses pointers, is memory intensive as it stores eight nodes per node. *linear-octree* is an alternative representation of an octree that stores its locational code that can be used to compute the children nodes' locational code. There is, therefore, no longer a need to store explicit pointers to the children.

The *k-d tree* is a tree structure that generalizes the octree and quadtree structures in a *k-dimension* space (Zhou et al., 2008). The *k-d tree* dimension does not have to be the same as the space dimension. In a quadtree and octree, space is respectively divided into two and three and are therefore considered as *2-d tree* and *3-d tree*. For a *k-d tree* structure, on the other hand, space is usually divided in the cycle following the axis in a *k-d tree* structure. It is, therefore, possible to construct a *3-d tree* from a 2-d space by first dividing the x-axis, then the y-axis, and finally the x-axis again.

3.4.4 Loose Octree

During the construction of a dynamic octree (or quadtree), some objects become stuck due to the straddling of the partitioning planes (Ulrich, 2000) as described in Fig. 3.5.

This problem is overcome by expanding the node volumes to some extent to make them partially overlapping. The resulting relaxed octrees have been dubbed loose octrees. The loose nodes are commonly extended by half the side width in all six directions (but may be extended by any amount). It effectively makes its volume eight times larger.

It is now possible to compute particles' depth level from their size for a constant $O(1)$ insertion. The particles have a higher level than in a regular octree and therefore offer more possibility of discarding possible collisions. However, the larger the nodes, the more they overlap on a level, and this leads to more particles being checked against each other for contact detection.

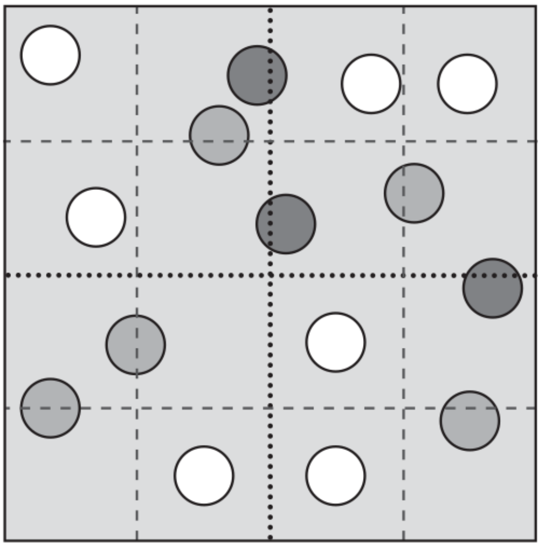


FIGURE 3.5: A quadtree node with the first level of subdivision shown in black dotted lines, and the following level of subdivision in gray dashed lines. Dark gray objects overlap the first-level dividing planes and become stuck at the current level. Medium gray objects propagate one level down before becoming stuck. Here, only the white objects descend two levels (Ericson, 2004).

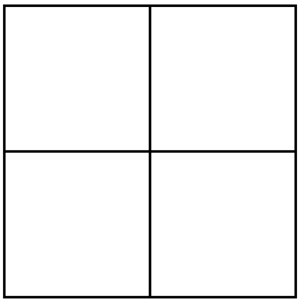


FIGURE 3.6: Regular domain decomposition as a quadtree.

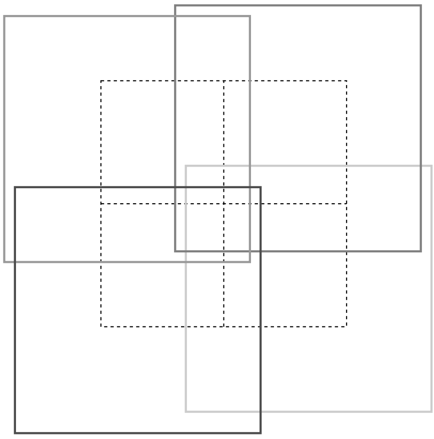


FIGURE 3.7: Loose quadtree representation. The nodes have been expanded by half the node width in the two directions.

FIGURE 3.8: Representation of a regular quadtree and loose quadtree (Ericson, 2004).

3.4.5 Grid-based spatial partitioning algorithms

The uniform grid approach's basic but efficient idea is to split the domain into subdomains or cells of equal size (Fig. 3.9). The objects are then associated with the cells; they overlap by usually using a spatial hashing. The pair of bounding volumes that do not overlap the same cells or neighbor cells are discarded for the contact list. An object is checked against objects contained in the same cell but also against all objects that lay in the direct neighbor cells, as shown in Fig. 3.10.

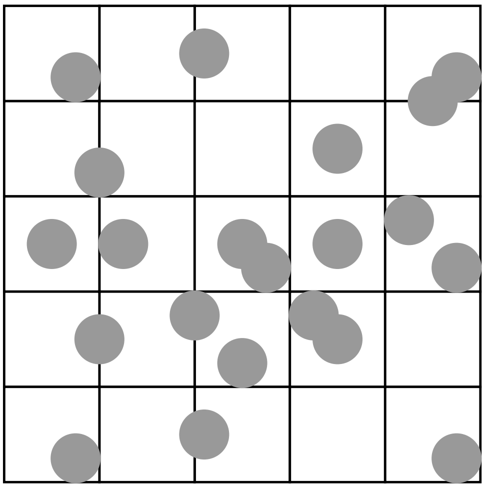


FIGURE 3.9: (A)

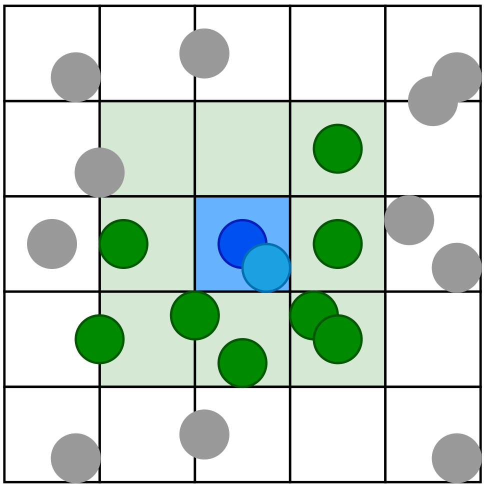


FIGURE 3.10: (B)

FIGURE 3.11: Uniform grid approach. (A) Uniform grid subdivision in equal sized cells. Each particle is assigned to a cell where its center is located. (B) The dark blue particle is checked with the light blue particle contained in the same cell. It is also checked against green particles located in the direct neighboring cells in light green.

The main concern of the uniform grid approach is the critical choice of the cell size. A too fine grid or small cell size potentially leads to excessive memory to store a large number of cells. Many cells have to be updated for large and moving objects covering a broad space, causing performance degradation. On the other hand, large cell size could have contained too many objects and lost the approach's discriminatory power and a drop in performance. Therefore, it is challenging in a world-size problem consisting of objects of different sizes to choose an optimal cell size.

One approach to deal with the object of vastly different sizes in a grid subdivision is to consider a hierarchical grid method. It consists of building multiple overlapping grids with different cell sizes. Objects are part of the grid level with the smallest cell size to fit even though objects can overlap multiple cells within multiple grids. The hierarchical grid approach is, in some ways, similar to the recursive octree method.

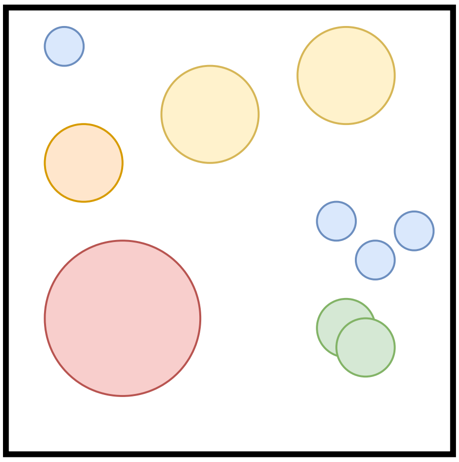


FIGURE 3.12: Hierarchical grid level 0.

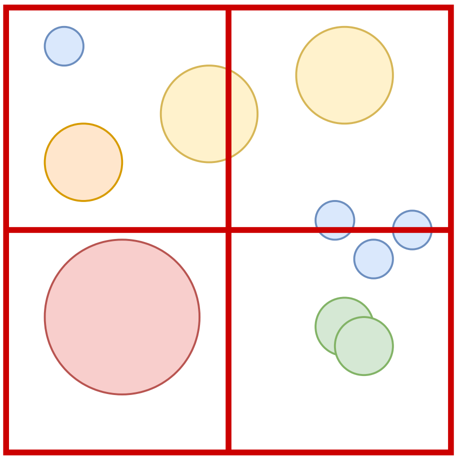


FIGURE 3.13: Hierarchical grid level 1.

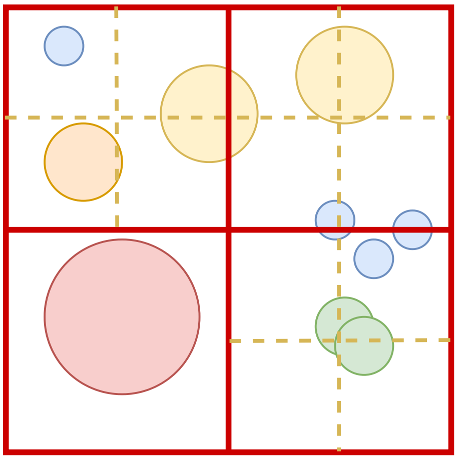


FIGURE 3.14: Hierarchical grid level 2.

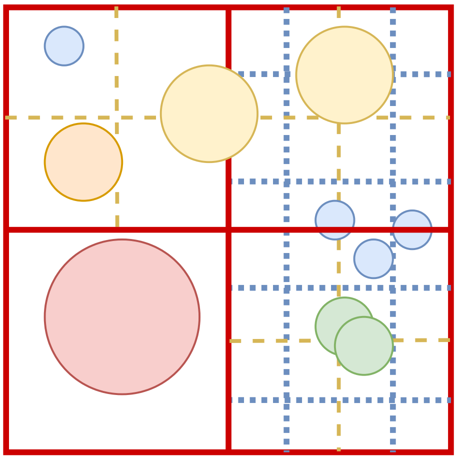


FIGURE 3.15: Hierarchical grid level 3.

FIGURE 3.16: Two dimensional hierarchical grid example. The level 0 corresponds to the domain and contains all the particles. The first level is the coarser grid (in red) and contains the biggest particle in red. The second level encapsulated the yellow particles without the biggest red particle. The third and last level contains the blue and green particles, that inserted in all the three grids.

The size of the largest object contained in the current grid is usually different from that level grid cell size at every level. For efficiency's sake, the cell size is often chosen to be a fraction of the biggest contained object. This fraction is the ratio of the cell size to the particle size and can be adjusted to control the number of objects per cell and grid and the number of grids. There exist two common approaches to check objects overlapping. The *Bottom-up* or *Top-down* sorts the objects in *ascending* or *descending* order based on objects size (Mirtich, 1996). For the *Bottom-up* approach (respectively *Top-down*), objects of a given level are only checked against an object of the same level and *higher* (respectively *lower*). The second approach would insert an object in all overlapping cells from all grid levels. For contact detection, an object is then checked with objects overlapping the same cells at all levels.

3.4.6 Sweep and prune

The *sweep and prune* method (Cohen et al., 1995) is also known as *sort and sweep* (Baraff, 1992) is a spatial sorting broad-phase collision detection method that uses the AABBs bounding volume and does not have straddling limitations as in a tree or grid approaches. In *sweep and prune* approach, the AABBs are sorted based on each coordinate's projections (start or lower bound and end or upper bound) on the axis. Two objects intersect when all X, Y, and Z coordinates axis projections overlap. The performance of the *sweep and prune* method greatly depends on the temporal coherency of the objects. When this is not the case, i.e., objects move significantly between two time-steps, sorting the complete list will greatly impact the performance. This approach does not perform well when the particles are especially clustered along any particular axis.

3.4.7 Bounding volume hierarchy (BVH)

A bounding volume hierarchy (BVH) is a tree structure of bounding volumes that form the tree's leaf nodes. Each node corresponds to a partition of the domain or set of bounding volumes, as shown in Fig. 3.17.

First, each of the bounding volumes forms the leaf nodes of the bounding volume hierarchy. After that, the nodes are arranged in multiple collections and enclosed within a larger bounding volume constituting a new set of nodes. The latter nodes are then enclosed in a larger bounding volume, and the tree is therefore constructed

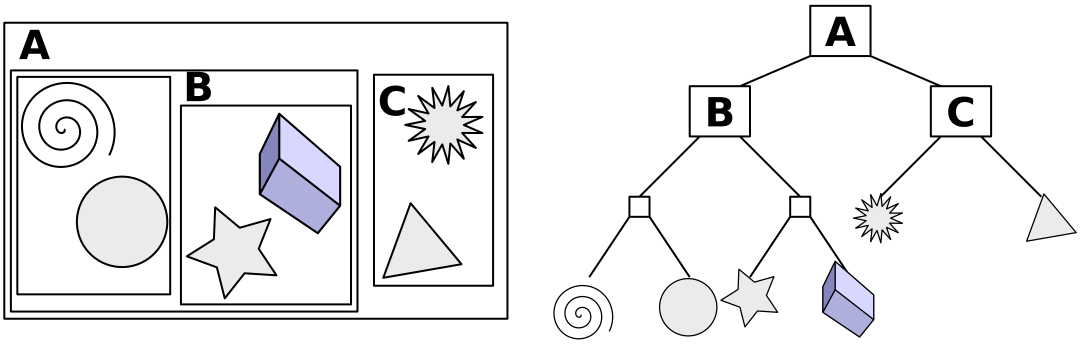


FIGURE 3.17: An example of a bounding volume hierarchy using rectangles as bounding volumes (Schreiberx, 2020).

recursively, eventually resulting in a tree structure with a single bounding volume at the top of the tree (Ericson, 2004). During a collision detection in a BVH approach, only leaf nodes (children) with the same parent are tested for intersection hence its discriminating power. The main difference between the BVH approach and the spatial partitioning approach, such as the uniform grid 3.4.5 lies in the fact that the bounding volumes in BVH can overlap the same domain regions, whereas the uniform grid partitions the domain in distinct regions.

3.4.8 Framework and C++ library for collision

We have also included three open-sourced collision detection libraries in our framework test: *Bullet, CGAL, and FCL.*

- *Bullet* is a real-time collision detection engine for virtual really(VR), games, visual effects, robotics, machine learning (Coumans, 2015). It is available on C++ version under *Zlib* license. The bullet library supports discrete and continuous collision detection for soft and rigid body dynamics. It provides a dynamic AABB tree contact detection algorithm and acceleration structures for distance and penetration points. Sphere, box, cylinder, convex, and non-convex meshes are the collision shapes supported by *bullet*.
- *CGAL* or Computational Geometry Algorithms Library (Fabri and Pion, 2009) is a software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library. It provides collision detection algorithms

such as loose and k-d trees, Convex hull algorithms, Shape analysis, fitting, and distances.

- *FCL* or Flexible Collision Library (Pan, Chitta, and Manocha, 2012) is a library for performing three types of proximity queries on a pair of geometric models composed of triangles. It can perform collision detection to find pairs of interacting shapes, distance computation for computing the closest point between two shapes, and provide continuous collision detection. Also, it is possible to access the contact information as contact points and normal. We use the recommended AABB tree algorithm 3.4.7 available in the *FCL* library as it is the fastest algorithm.

All three libraries are well-known and used in various domains such as robotics, virtual reality, computer games, and computational geometry. They all offer a different level of flexibility and implementation architecture and can easily be interfaced with other third-party libraries and software.

3.5 Benchmark

To compare the algorithms presented in sections 3.4, we have developed a `C++` framework to test their different algorithms implementations. For this purpose, a set of test cases have been defined. They consist of a packed bed of up to one million spherical particles with random radii in a 3D environment. The benchmark experiments were performed with the google benchmark library that offers easy building with fixtures, automatic iteration, argument parameters, or CVS outputs. The experiment was carried out on *Intel(R) Xeon(R) E5-2667 @ 2.90GHz* processor and the values reported are the means of at least 150 executions.

3.5.1 Sequential runs

The results presented in this section are from sequential implementations of the algorithms. We compared the simulation time over different number of particles from one thousand ($1k$) to one million ($1M$) particles. The algorithms have been separated in two groups: *slow and fast* as it appears that some algorithms are unable to perform the collision detection over 100k particles in a reasonable computational time. The *Bruteforce*, *Hierarchical Grid*, *Kd-tree*, *LooseOctree*, and *Octree* are categorized as the *Slow Algorithms* and the *FCL*, *Bullet*, *CGAL*, *Sweep&Prune*, *Uniform Grid* as *Fast Algorithms*.

The running time (and the standard error deviation *stddev*) of the *Slow Algorithms* is presented in Fig. 3.18. Surprisingly, overall, the brute force approach is the fastest algorithm with a $O(n^2)$ quadratic complexity.

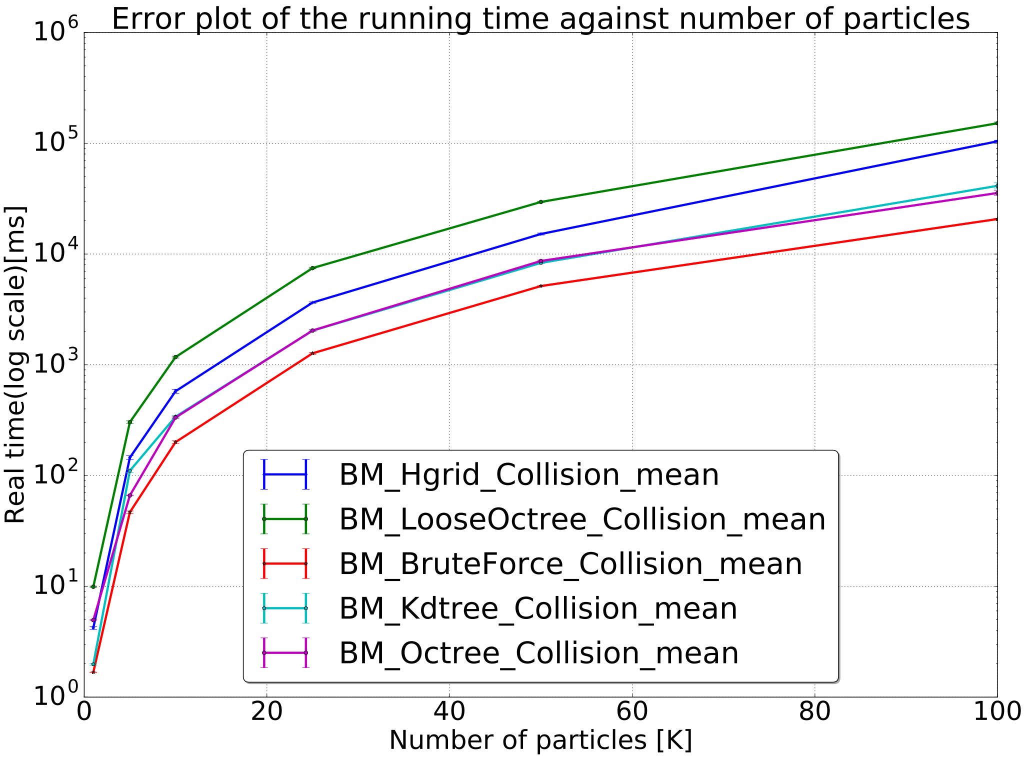


FIGURE 3.18: Comparison results for *Slow Broad-Phase Algorithms* from 1k to 100k of particles.

It can be explained by the fact that the tree algorithms(*Kd-tree*, *LooseOctree*, and *Octree*) require complex data structure that needs to be updated, which is incredibly costly in our test case where all particles move. They also require more memory for their structures. On the other hand, the brute force approach does not require any extra data-structures that involve overhead due to structure creation. The big surprise comes from the *Hierarchical Grid* approach, which was not expected to perform as one of the slowest but can be explained by the fact that the *Hierarchical Grid* approach does not suit our mono-disperse test case. Still, we were expecting it to perform better than the naive approach. The performance of the *Hierarchical Grid* depends on the size of the cell and particle size, the variance of the particle size, and, more importantly, the hash function. Therefore, it was ruled out, and we did not dig further to integrate it into our framework, but we did implement it inside our XDEM software as a non-negligible part of our test cases is poly-diverse. However, tree algorithms were not considered for implementation within the XDEM software as it implies complex data structure modifications.

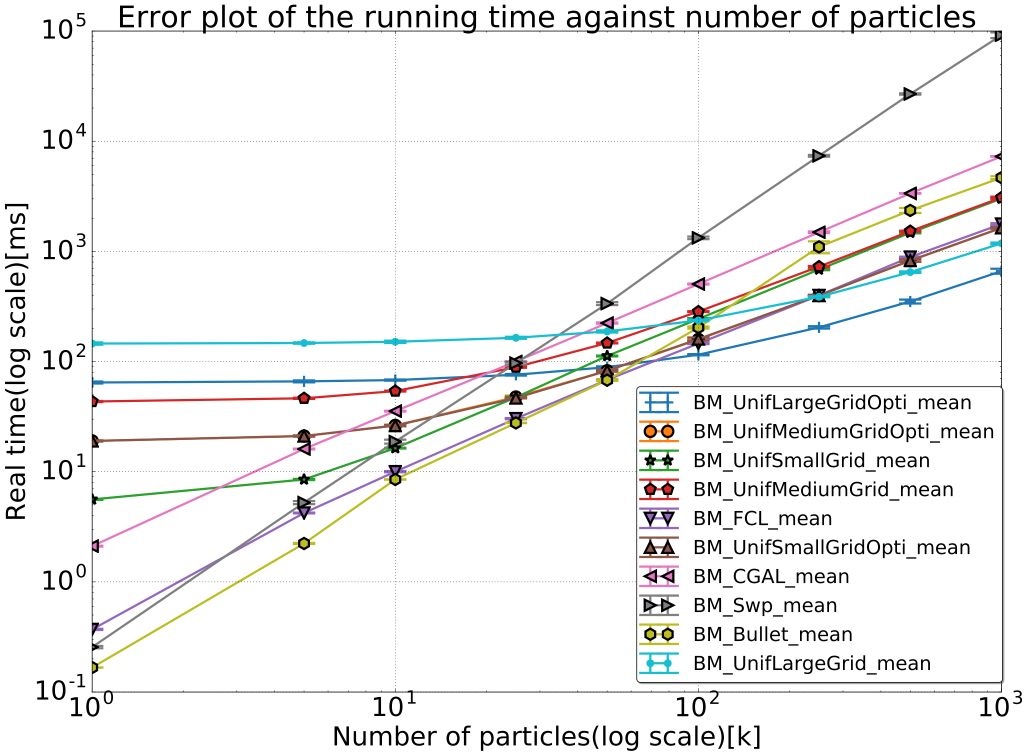


FIGURE 3.19: Comparison results for *Fast Broad-Phase Algorithms* from $1k$ to $1M$ of particles.

On Fig. 3.19, we compare the *Fast Algorithms*: *FCL*, *Bullet*, *CGAL*, *Sweep&Prune* and *Uniform Grid* algorithms. We have implemented two versions of the uniform grid: a classical approach creating and using cell structure and an optimized version where the cell structure is not constructed, but interpolation is used to compute each particle's corresponding cell. The optimized version also uses a data locality approach by using a Morton space-filling curve, which consists of accessing cell data closely stored in memory (Gaede and Günther, 1998). The *Uniform Grid* algorithm has also been evaluated in three different variations: Small (S), Medium (M), and Large (L) grid size corresponding to three different cell sizes. The purpose is to evaluate the impact of the number of cells in a grid decomposition method performance.

We can see on Fig. 3.19 that the three frameworks (*FCL*, *CGAL* and *Bullet*) and the *Sweep&Prune* algorithms are very efficient for small number of particles up to $10k$. Above $50k$ particles, the uniform grid approaches appear to fit a large number of particles best. As expected, the optimized uniform grid implementation is faster than the traditional approach, especially with many particles. The grid size impacts the uniform grid decomposition because the smaller the grid is, the better it suits a small number of particles. On the other hand, the more particles there are, the better it is to have a larger grid. Compared to other algorithms, the grid approaches under-performed for a small number of particles due to the grid creation's overhead. However, the grid approach's running time is constant for the number of particles up to $50k$ because the overhead time exceeds the actual collision detection load. Using a small number of particles and larger cell size increases the grid-browse overhead because more cells are visited than particles (contained). Using a large number of particles and a small grid partitioning increases the particles' check overhead. Indeed, this situation is similar to performing a brute force approach. From the above analysis, it can be stated that it is better to use *FCL*, *CGAL*, *Bullet*, and *Sweep&Prune* when dealing with a small number of particles. However, there is a significant drawback to using third-party libraries: they use their own data structures to perform collision detection. Converting or adjusting our own data structures in XDEM to a third-party library structure to perform the contact detection and converting back the results to our structure has a non-negligible overhead that makes it not practical to use in XDEM. For that explicit complexity, even if the third-party libraries are open-sourced, we did not consider them for further implementation inside our XDEM software.

3.5.2 Parallel runs

Some algorithms and implementations among the ones presented in this chapter have been parallelized with OpenMP. Among the *slow algorithms*, only the *brute-force* has been parallelized to serve as a baseline comparator. Among the *fast algorithms*, *sweep and prune* and six variants of the *uniform grid* algorithms (**original** and **optimized** with small, medium and large grid sizes) have adapted to an OpenMP parallelization. Fig. 3.20 presents the (strong) scalability results of the parallelized algorithms using 100*k* random particles in the test case. The number of cores/threads have been varied from 1 to 24.

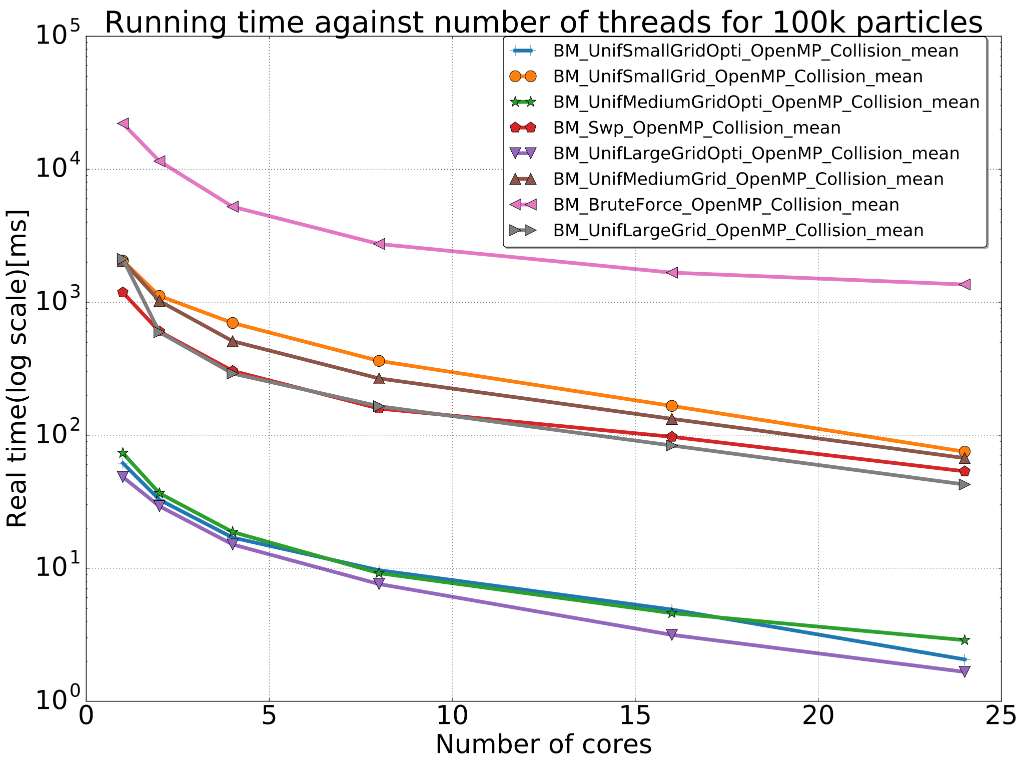


FIGURE 3.20: OpenMP scalable results for *Fast Broad-Phase Algorithms* from 1 to 24 of cores.

It appears in Fig. 3.20 that the algorithms are divided into three groups:

- The first group is composed of only the *brute-force* approach. We can observe that the *brute-force* algorithm remains the slowest approach. However, it scales very well as the load(particles) is perfectly balanced between the OpenMP threads.
- The second group is composed of the **original** *uniform grid* with the different cell sizes and the *sweep and prune* approaches. In *uniform grid*, the **large cell** method is fastest than the **medium and small cell** methods.

Medium and small methods appear to be much closer (in simulation time) with a slight advantage for the **medium** medium. The three methods' performance order agrees with the sequential results for a large number of particles, as shown in Fig. 3.19. With the scalability, the order does not change with the increase in the number of cores. The *sweep and prune* curve lies in between the *uniform grid* curves. As in Fig. 3.19 and section 3.5.1 showing the sequential performance, the *sweep and prune* performance is worst than (in sequential) the *uniform grid* approach. However, it scales very well (as there are only comparisons and data accesses) and has a similar performance than the *uniform grid* approach when the number of threads increases.

- Finally, the third and fastest group is composed of the **optimized** *uniform grid* with different cell sizes. The **large cell** method is fastest than the **medium and small cell** methods. **Medium and small** methods appear to be very close (in simulation time). The three methods' performance order agrees with the sequential results for a large number of particles, as shown in Fig. 3.19. With the scalability, the order does not change with the increase in the number of cores. As stated in section 3.5.1, using a Morton space-filling curve improves the *uniform grid* approach performance.

3.5.3 Adaptive approach

Looking at Fig. 3.5.1, we have noticed the algorithms could be classified into three groups along the x-axis:

- From 100 to 5k particles. In this interval, even the slowest algorithms as the *brute-force* are very competitive. It is faster than the *uniform grid* approach. Therefore, the *brute-force, CGAL, Bullet, FCL and sweep, and prune* can all be considered as the preferred approach for this range number of particles;
- From 5k to 50k particles. In this interval, the *Bullet and FCL* libraries are the most competitive approaches. However, this is not of great interest to us because they have not been integrated into XDEM. But we can notice that the

original *uniform grid* approach is in general faster than the **optimized** *uniform grid* approach using a Morton space-filling curve (it implies sorting overhead). As a result, the **original** *uniform grid* approach is preferred when using a case number of particles between $5k$ and $50k$;

- And from above $50k$ particles. In this interval, the **optimized** *uniform grid* method is the best approach to adopted as it gives the best performances and scalability.

From the above observation and based on the algorithms' complexities, we have implemented a *naive adaptive* method that chooses the best contact detection approach depending on the number of particles and threads (when performing parallel simulations) based on our early observations. For a simulation case with less than $5k$ particles, the *sweep and prune* approach is used for the collision detection process. The **original** *uniform grid* approach is used for simulation case with number of particles between $5k$ and $50k$. Finally, we use the **optimized** *uniform grid* for all simulation case with more than $50k$ particles.

3.6 Conclusion

This chapter investigates the collision detection process performance in the DEM method by developing a C++ benchmark for comparing multiple algorithms. An evaluation of different Broad-Phase algorithms has been investigated using the *Execution time* metric.

Section 3.3 introduces the implementation, advantages, and disadvantages of the bounding volumes commonly used in the broad-phase: Axis-Align Bounding Box (AABB), Oriented Bounding Box (OBB), and Sphere. In section 3.4, we have presented the different broad-phase algorithms implemented in the benchmark: *brute-force, sweep and prune, tree-based, grid-based, and bounding volume hierarchy*.

Performance results were proposed within a test case composed of 100 to $1M$ million of particles placed in a three-dimensional environment. It appears that some algorithms perform better with a low number of particles, whereas others are more efficient with large numbers of particles. Of course, those results are very tied to the implementation of those algorithms and the test case, which means that the best algorithms' choice depends on the application. The number of particles heavily impacts algorithms' performance. In this regard, an adaptive approach has been implemented to select the best algorithm depending on the number of particles. The *brute-force* and the algorithms and implementations considered as the fastest (*sweep and prune and uniform grid*) have been parallelized with the OpenMP approach. The trend observed in the sequential runs is confirmed in the parallel runs because the implementations scale very well.

Finally, this work is meant to provide a reference and benchmarks for future works. The next step is to feature parallel implementations of these Broad-Phase algorithms using GPU capabilities.

Chapter 4

LOCAL VERLET BUFFER APPROACH FOR BROAD-PHASE INTERACTION IN DEM

@authors: Abdoul Wahid Mainassara Checkaraou, Xavier Besseron, Alban Rousset, Fenglei Qi, and Bernhard Peters

Under review in journal of *Advances in Engineering Software*

4.1 Abstract

The Extended Discrete Element Method (XDEM) is a novel and innovative numerical simulation technique that extends the dynamics of granular materials or particles as described through the classical discrete element method (DEM) by additional properties such as the thermodynamic state, stress/strain for each particle. Such DEM simulations used by industries to set up their experimental processes are complex and heavy in computation time.

Those simulations perform at each time step a collision detection to generate a list of interacting particles that is one of the most expensive computation part of a DEM simulation. The Verlet buffer method, which was first introduced in Molecular Dynamic (MD) (and is also used in DEM), allows to keep the interaction list for many time step by extending each particle neighborhood by a certain extension range, and thus broadening the interaction list. The method relies mainly on the stability of the DEM, which ensures that no particles move erratically or unpredictably from one time step to the next: this is called temporal coherency. In the classical and current approach, all the particles have their neighborhood extended by the same value, which leads to sub-optimal performances in simulations where different flow regimes coexist. Additionally, and unlike in MD (which remains very different from DEM on several aspects), there is no comprehensive study analyzing the different parameters that affect the performance of the Verlet buffer method in DEM.

In this work, we apply a dynamic neighbor list update method that depends on the particle's individual displacement and an extension range specific to each particle and based on their local flow regime for the generation of the neighbor list. The update of the interaction list is analyzed throughout the simulation based on the particle displacement allowing a flexible update according to the flow regime conditions. We evaluate the influence of the Verlet extension range on the performance of the execution time through different test cases, and we empirically analyze and define the extension range value giving the minimum global simulation time.

4.2 Introduction

Discrete Element Method (DEM), originally proposed by Cundall (Cundall and Strack, 1979), is a popular simulation approach for studying and diagnosing bulk powder/-granular dynamic systems, which are ubiquitous in the pharmaceutical industry, food processing, chemical engineering, mining industry, and energy systems (Ketterhagen, Ende, and Hancock, 2009; Ransing et al., 2000). Considering the large scale of applied systems, one of the key efforts in the DEM development is to enhance the simulation capability of DEM software, such as by adopting advanced parallelism schemes (Maknickas et al., 2006), utilizing graphics processing units (GPU) (Gan, Zhou, and Yu, 2016) and developing coarse-grain models (Weinhart et al., 2016). However, one unavoidable functionality that DEM codes need to optimize is the collision detection, which, includes neighbor search, represents a major computational part in DEM simulations (Rousset et al., 2018; Páll and Hess, 2013).

Collision detection is often split into two phases: a broad-phase, which formulates a potential collision list for each particle (neighbor list), and a narrow-phase accounting for accurately resolving the collision instance of each pair of particles in the neighbor list. Different algorithms for constructing neighbor list in the broad-phase are available, including brute force approach (Kockara et al., 2007) of $O(n^2)$ time complexity, binning approach (Tracy, Buss, and Woods, 2009) and linked-cell method (Welling and Germano, 2011) of $O(n)$ complexity. However, the broad-phase computational efficiency is not solely determined by the time complexity of the adopted algorithm (Rousset et al., 2018), which are also affected, for instance, by the ratio of cell size to particle size in the commonly used linked-cell approach or by the frequency of updating the neighbor list in the broad-phase. The latter is usually related to the Verlet list approach that is firstly proposed in MD simulations by Loup Verlet (Verlet, 1967) for reducing the unnecessary cost of rebuilding the neighbor list at every simulation time step. The mechanism for skipping neighbor list rebuild is achieved by providing an extra margin (often called "skin") on top of the particle pairwise cut-off interaction distance. The neighbor list built is called a Verlet list. With this mechanism, the Verlet list remains unchanged until a particle displacement exceeds a certain threshold distance defined beforehand.

For MD simulations, the systems are often homogeneous, and a uniform (global) buffer is satisfactory to achieve a good speed-up. However, for the majority of powder and granular dynamic systems, the variation of particle flow properties such as particle velocity and solid fraction in the systems is significant. It becomes less efficient to adopt a uniform skin margin for particles at regions of different flow conditions. Intuitively, in such systems, providing a larger skin margin for particles moving faster leads to a more reasonable neighbor list updating frequency globally. Many parameter studies on the skin margin determination have been reported in MD simulation research (Verlet, 1967; Chialvo and Debenedetti, 1990; Chialvo and Debenedetti, 1991; Mattson and Rice, 1999; S. and S., 2006). For MD, in Lennard Jones systems, often $skin = 0.3\sigma$, where σ is the diameter of a Lennard Jones particle. For DEM development, Li et al. (Li et al., 2010) compared the performances of Verlet buffer and linked-cell approaches in gravity-driven granular collapse simulation. It is reported that appropriate determination of parameters such as search radius (skin + cut-off distance), cell size, and updating interval time step is critical for improving simulation efficiency in the Verlet buffer approach. Although the Verlet buffer mechanism has also been implemented in several DEM codes (Fang, Tang, and Luo, 2007; Munjiza, Walther, and Sbalzarini, 2009), a uniform skin margin is often adopted for all the particles. In (Angeles and Celis, 2019), the performance of neighbor search methods (Verlet table and linked cell) and associated computational costs are parametrically evaluated, and an evaluation of their suitability for carrying out the DEM/CFD numerical simulations is made. The main outcome of their research showed that the Verlet list has a strong dependency on the skin factor, and the value for this parameter equal to the particle radius does not create problems in the identification of particle pairs. Unfortunately, one noticeable problem is to set the update frequency of the Verlet list according to the uniform skin margin and globally to the maximum velocity leading to stability issues (Li et al., 2010; Fang, Tang, and Luo, 2007) considering that particles have the possibility of migrating over the skin distance within updating interval. The performance concern of Verlet buffer arises as a result of heterogeneous flow conditions commonly found in real particle systems, which makes the adoption of a uniform skin margin parameter less computationally efficient. Dynamically determining a local skin margin for each particle according to local flow conditions is suggested in lots of researches, but the optimal determination of the skin margin needs to be thoroughly studied.

In this research, we proposed a local Verlet buffer approach using a new skin margin formulation, which dynamically expresses the skin margin for each particle according to the neighborhood flow conditions and based on the particle velocity. This approach enables the heterogeneity of real particle systems to be taken into account for better computational time efficiency. In this study, the potential stability problem of particles moving over the skin distance in a period of update time is fixed by recording each particle displacement and automatically deciding when the Verlet list is to be rebuilt. We ensured and demonstrated that there are no missed interactions in our current approach, and thus our results are identical to using the naive approach. To assess the efficiency of our proposed formulation, we have implemented the local Verlet buffer approach in our in-house eXtended Discrete Element Method (XDEM) software (Peters, 2013). We, therefore, explored how the skin margin value affects the broad and narrow-phase in particular and the global simulation time in general. The main goal of this paper is to propose a broad analysis of our skin formulation implemented in DEM software like our in-house XDEM toolbox. It points out the advantages of using such formulation and its best-use case but also its drawbacks.

This paper provides a general overview of the XDEM software in the background section 4.4 and describes the collision detection method before our current research. The contribution of the article is presented in section 4.5, which describes the local Verlet buffer approach for building the list of interacting particles and proof of the method's validity. In section 4.6, The skin margin parameter is studied by employing deterministic designs to explore the effect of parametric changes within simulation models. The results and conclusion are discussed in section 4.6.4. We finally give a general conclusion of the paper in section 4.7.

4.3 Related work

The Verlet list was first introduced for molecular dynamic simulations by Loup Verlet in his article (Verlet, 1967) back to 1967. The method is now widely used in DEM simulations and considerably decreases the simulation time. The Verlet list allows to reduce the evaluation of the unnecessary interactions and to keep a neighbor list for several time-steps until a breach. Loup Verlet himself proposed to extend the particles interaction range by a certain skin margin given by:

$$R_{NL} = R_C + skin, \tag{4.1}$$

where R_{NL} is interaction range and R_C the cut-off radius. It is then possible to have the exact update interval time step for a given interaction range. Sutmann et al. (S. and S., 2006), Awile et al .(Awile et al., 2012), Mattson et al. (Mattson and Rice, 1999), and Chialvo et al. (Chialvo and Debenedetti, 1990) proposed different procedures to determine the skin value in MD depending on parameters as: density, temperature, time step, system size, and molecular geometry.

Chialvo et al. (Chialvo and Debenedetti, 1990) investigate the effects of the parameters cited above upon the optimum neighbor list radius and update frequency. The theoretical predictions (according to which the optimum neighbor list radius increases with sample size, temperature, and time step and decreases with density) validate the simulation results. The study of the paper is in some ways similar to this paper, unlike their study is based on MD simulations, while this work focuses on DEM simulations, which are very different from MDs in many aspects. In both methods, N or K, the update interval time step is not a fixed number but determined by the particle displacement. The difference comes actually from the displacement calculation method: we consider a linear displacement since the last neighbor list update and Chialvo et al. considered in their paper the displacement as the accumulated displacement suffered by the particle since the last neighbor list update.

Sutmann in (S. and S., 2006) investigates the performance of neighbor list techniques in MD simulations depending on a variety of parameters, which may be adjusted for maximum efficiency. The model presented allows choosing optimal parameters for the performance of the Verlet list and linked-cell lists. The paper targets only Lennard–Jones MD systems.

Awile (Awile et al., 2012) presents a novel adaptive-resolution cell list (AR cell list) algorithm and the associated data structures that provide efficient access to the interaction partners of a particle, independent of the (potentially continuous) spectrum of cut-off radii present in a simulation. They characterize the computational cost of the proposed algorithm for a wide range of resolution spans and particle numbers. Mattson presents a modified method here, allowing for reductions in the cell sizes and the number of atoms within the volume encompassing the neighbor cells. The algorithms determine the volume with the minimum number of neighbor cells as a function of cell size and the identities of the neighboring cells. It also evaluates the serial performance as function of cell size and particle density for comparison with the performance using the conventional cell-linked list method. The two papers by Awile and Mattson target the cell linked list method rather than the Verlet list or a combination of the two methods.

LIGGGHTS and LAMMPS (Kloss et al., 2012; Plimpton, Crozier, and Thompson, 2007) software allow setting parameters that affect the building of pairwise neighbor lists. All-atom pairs within a neighbor cut-off distance equal to their force cut-off plus the skin distance are stored in the list. The default value for skin depends on the choice of units for the simulation and the inputs.

The methods presented previously and found in the literature mainly focused on MD simulations (some on DEM) where neighbor list updates frequency is usually (not always) fixed beforehand. Our paper, on the other hand, presents an interaction range (*skin*) formulation based on the velocity of each individual particle. It also focused on an automatic update list technique based on particle displacement avoiding a divergence or crash of the system while reaching optimal performances in DEM simulations.

4.4 Background

The XDEM software is a numerical multi-physics simulation framework (Peters, 2013; Samiei and Peters, 2010) supporting parallel processing (Besseron et al., 2013; Checkaraou et al., 2018a), and based on the dynamics of granular material or particles described by the classical DEM (Cundall and Strack, 1979; Allen and Tildesley, 1990). It is extended by additional properties such as the thermodynamic state and stress/strain for each particle for more complex simulations in various domains (Peters et al., 2019; Peters and Pozzetti, 2017; Mahmoudi et al., 2016a). As in any DEM code, the particle interaction detection is a major part of XDEM, and it uses the linked-cell method to generate the interacting particles list. Firstly, an overview of the XDEM work-flow is provided with different key parts. Then the collision detection techniques and the different issues making it a major DEM component are presented. Finally, an overview of the linked-cell method and its current implementation in XDEM is given.

4.4.1 XDEM flow chart

A flow chart, as shown in Fig. 4.1, illustrates the main components of XDEM software for particle dynamics simulation. An iterative time loop is composed of five major phases:

- **Broad-phase**: uses a fast but approximate contact detection to build a list of particle pairs that can potentially interact. It should be noted that the pairs of potentially interacting particles are stored in a unique list. During this phase, the particles are represented by a bounding volume shape. It builds the list of interacting particle pairs by dividing the domain into cells with uniform size using the linked-cell method. The broad-phase could take up to 65% of the total computational time;
- **Narrow-phase** performs a rigorous contact detection of each pair of particles in the broad-phase list using the actual shape of the particle and calculates the pairwise collision parameters such as overlap, contact location, and direction. The XDEM software supports complex shapes by using the sub-shapes techniques (a shape is composed of many simple shapes as spheres) and super- quadratics. The narrow-phase represents around 15% of the total computational time;
- **Apply physical models**: based on the collision parameters and collision history information, calculate all interaction forces by applying corresponding physical

models such as normal and tangential contact models, rolling models, and cohesive models and so on;

- **Integration**: updates the particle location, velocity, rotational velocity, and orientation information by numerically integrating Newton's second law with various numerical algorithms such as leapfrog and velocity-Verlet schemes;

4.4.2 Collision detection in XDEM

The contact detection being carried out on a large number of particles, it is split into two phases in order to reduce the computational complexity: a first, fast and approximate phase called the **broad-phase** and an accurate second phase called **narrow-phase** as indicated in Fig. 4.1 and already mentioned in Section 4.4.1.

Fig. 4.2 illustrates the collision detection process for two colliding particles of any shapes in XDEM. A bounding volume enclosing any type of particle entity replacing the real particle shape is used to achieve a rapid broad-phase detection. In XDEM software, the broad-phase is carried on using bounding spheres, slightly increasing the memory usage (BS requires additional data) but greatly improves and reduces data access from the CPU. With the BSs, the distance computation becomes less computationally expensive.

Realizing that two particles far away from each other have very little chance to generate any interaction, the neighbor particle detection is often limited to a certain distance. The Algorithm 1 describes the linked-cell technique used in XDEM to spatially limit the contact detection process of a pair of particles.

The linked-cell approach is utilized to perform the neighbor list construction, which guarantees the time complexity to be linear in the number of particles in the system. As illustrated in Fig. 4.3, the pairwise interactions for a single particle are limited with all particles within the same cell (green) and in the immediate or adjacent neighboring cells (blue). The cell size is uniform and must not be smaller than the maximum bounding sphere size of all particle entities.

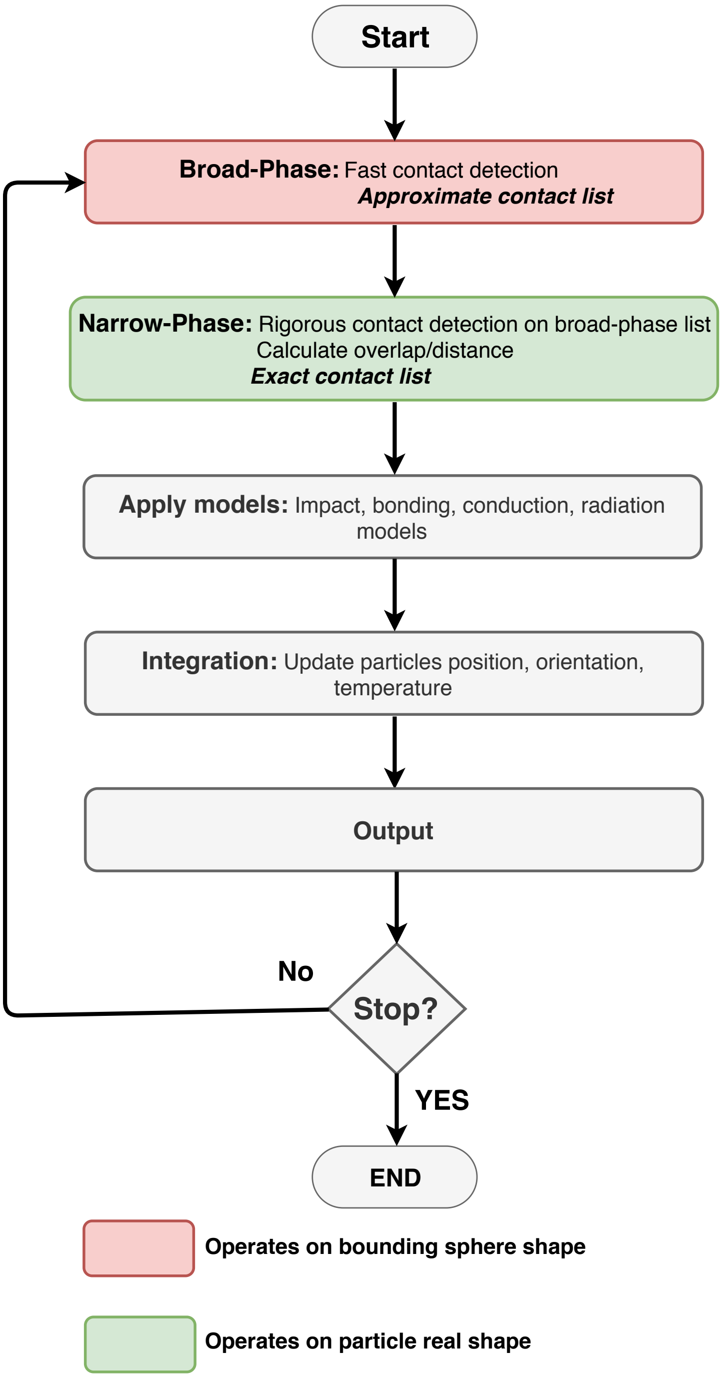

Start
Broad-Phase: Fast contact detection
Approximate contact list
Narrow-Phase: Rigorous contact detection on broad-phase list
Calculate overlap/distance
Exact contact list
Apply models: Impact, bonding, conduction, radiation models
Integration: Update particles position, orientation, temperature
Output
No
Stop?
YES
END
Operates on bounding sphere shape
Operates on particle real shape

FIGURE 4.1: Flow chart of XDEM software detailing the main different steps in an iterative simulation.

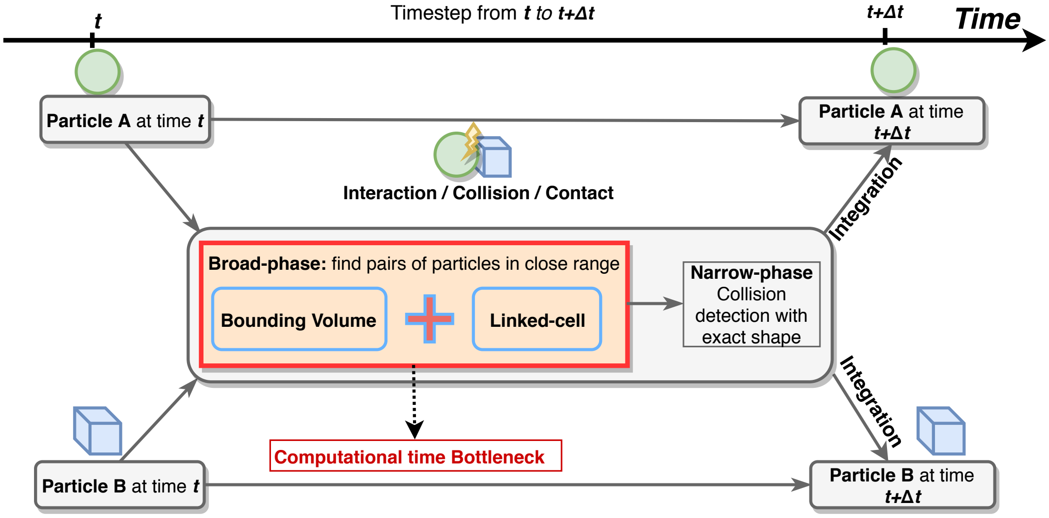


FIGURE 4.2: Collision detection (broad-phase and narrow-phase) process workload in XDEM. The broad-phase is the main computational time consumer.

Algorithm 1: Linked-cell algorithm

```
Uniform decomposition of the domain in cells;
for all $C_a$ in cell list do
    for $C_b$ in the immediate neighbour of $C_a$ do
        // Make sure to check each pair of cells only once
        if $index(C_b) < index(C_a)$ then
            for each $P_a$ among all particles in cell $C_a$ do
                for each $P_b$ amoung all particles in cell $C_b$ do
                    // Check if the bounding spheres of $P_a$ and $P_b$ intersect
                    if $||X_a - X_b|| \leq r_a + r_b$ then
                        $List \leftarrow (P_a, P_b)$;
                    end
                end
            end
        end
    end
end
```

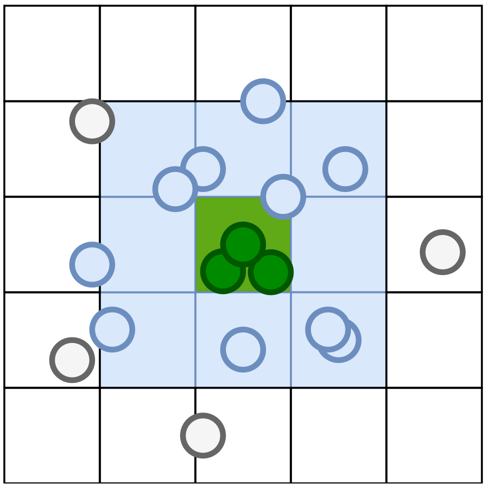


FIGURE 4.3: Illustration of the cell linked method. For the particles in green cell, a collision is checked with particles in the same cell (green particles), and also within the immediate neighbour cells (blue).

The narrow-phase, using the shapes of the real particles, is performed on the broad list of interacting pairs of particle returned by the broad-phase. The time complexity for narrow-phase collision detection is largely determined by the particle shape. For spherical particles, the narrow-phase detection is simply checked following identity:

$$\delta = r_i + r_j - ||X_i - X_j|| \tag{4.2}$$

where, r_i represents the radius of particle i, and X_i is the center coordinates of particle i. If the overlap δ is positive, the two particles collide and vice versa. For other particle shapes such as superquadratics, the collision detection becomes complex, and usually, an optimization problem needs to be solved (Williams and Pentland, 1992).

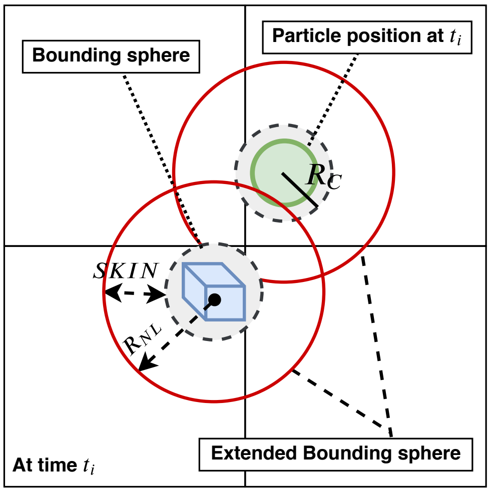
4.5 Local Verlet buffer approach

The Verlet buffer, unlike the conventional Verlet list, does not build a neighborhood list for each particle but rather a global list of interacting particle pairs, also called the Verlet list. In both methods, the particle cutoff radius (bounding sphere in DEM) is surrounded by a skin margin (Allen and Tildesley, 1990; Allen and Tildesley, 2017), to give a larger neighborhood. Another difference lies in the ability to work with any broad-phase algorithm.

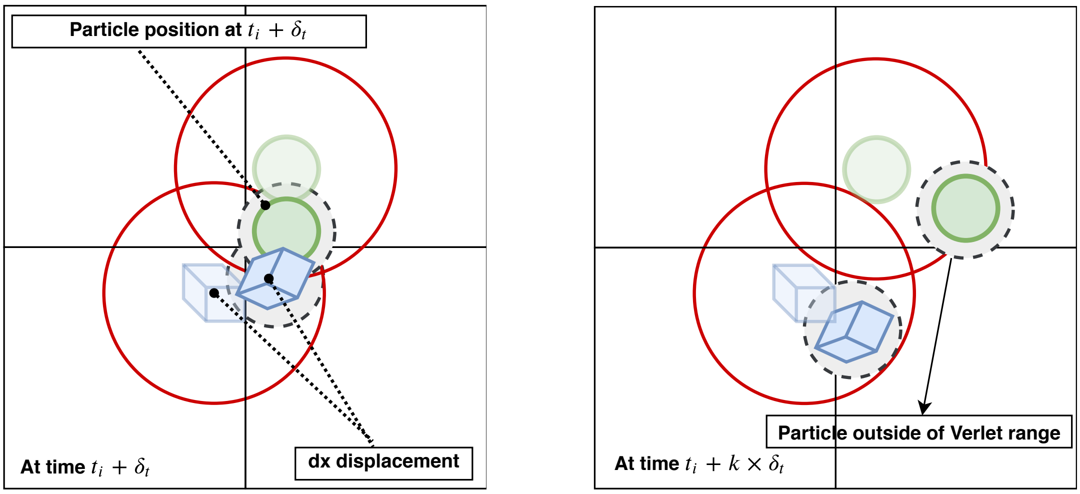
In our approach, we extend the bounding spheres used by the broad-phase to perform an approximate collision detection. The extensive range of the bounding spheres is called **skin**, and it will increase the number of potential interactions found by the broad-phase by considering pairs of particles that are located further away from each other. On one side, this will make the broad and the narrow phases costlier to evaluate, but on the other side, the broad-phase does not have to be executed at every time step anymore. By considering a larger surrounding in the broad-phase collision detection, the list of potential interactions now includes interactions that could happen in the next time steps.

In the local buffer method, the skin margin used to extend the bounding volumes is unique to every particle and is computed according to their local flow conditions. Additionally, we propose a condition that allows checking that the result of the previous broad-phase (i.e., the list of potential interactions) is still correct (Noske, 2004). When this condition is broken, we force the execution of a new broad-phase. In any case, the narrow-phase is always executed on this approximated list of interactions, and that guarantees that the results will be strictly identical with the case of having the broad-phase always executed.

The application of Verlet buffer in particle collision detection process is illustrated in Fig. 4.4. In the example of Fig. 4.4a, the two particles, with different shapes, have their bounding spheres extended by a skin margin, often called the Verlet skin. This phase extends each particle neighborhood and includes the pair of particles in the Verlet list, which without the extension, would not have been considered as potentially interacting. Fig. 4.4b shows that after a couple of steps, the Verlet list does not need to be updated. The particles, or more precisely the bounding sphere of each particle, did not leave their respective extended bounding spheres that was used to perform the broad-phase collision detection initially. That shows how an extension of a skin margin in the neighborhood of the particle includes the pair of particles in the Verlet list and catches an active collision that happens a few steps later. Finally, Fig. 4.4c illustrates the case where one of the particles moves out of its skin margin.



(A) Extension of the interaction range by surrounding the cut-off radius by a skin margin.


Particle position at $t_i + \delta_t$
At time $t_i + \delta_t$
dx displacement
Particle outside of Verlet range
At time $t_i + k \times \delta_t$

(B) Collision in extended interaction range when the two particles are still in the Verlet list. The narrow-phase is applied to check the actual collision.

(C) New broad-phase required, the green particle have moved for more than a skin distance(and could have moved in another particle's neighborhood).

FIGURE 4.4: Initial configuration and update of the Verlet list.

This means that the Verlet list is not valid anymore, and the broad-phase must be executed again to generate a Verlet list.

The overall procedure for constructing the Verlet buffer list in XDEM is shown in Fig. 4.5

During a broad-phase collision detection, for a particle i, every particle j in its neighbor cell is checked, as described in the linked-cell method, to determine whether the pair (i, j) should be included in the Verlet list or not. Particle i and j constitute a pair in the Verlet list, if they satisfy the condition of two overlapping bounding spheres. The radius of the bounding volume for particle i, called the neighborhood list radius, is calculated as:

$$R_{NL,i} = R_{C,i} + skin_i, \tag{4.3}$$

where, $R_{C,i}$ is the interaction cutoff radius determined by applied physical models, and $skin_i$ is the local Verlet skin distance, the value of which depends on the local flow properties. The construction of the Verlet list process loops over all the pairs of particles in the linked-cell neighbor list for completion. In the next following steps, the displacement of each particle starting since the last Verlet list build is examined against a local threshold. The Verlet list remains unchanged, and the broad-phase collision detection is skipped until a violation occurs. The following talks about how to determine the local skin distance and a scheme for automatically updating the Verlet list in this approach.

4.5.1 The local skin parameter

To choose the skin distance parameter, we extend the equation proposed by Loup Verlet in 1967 (Verlet, 1967) for MD and we propose to use the formula:

$$skin_p = K \times v_p.\Delta t \tag{4.4}$$

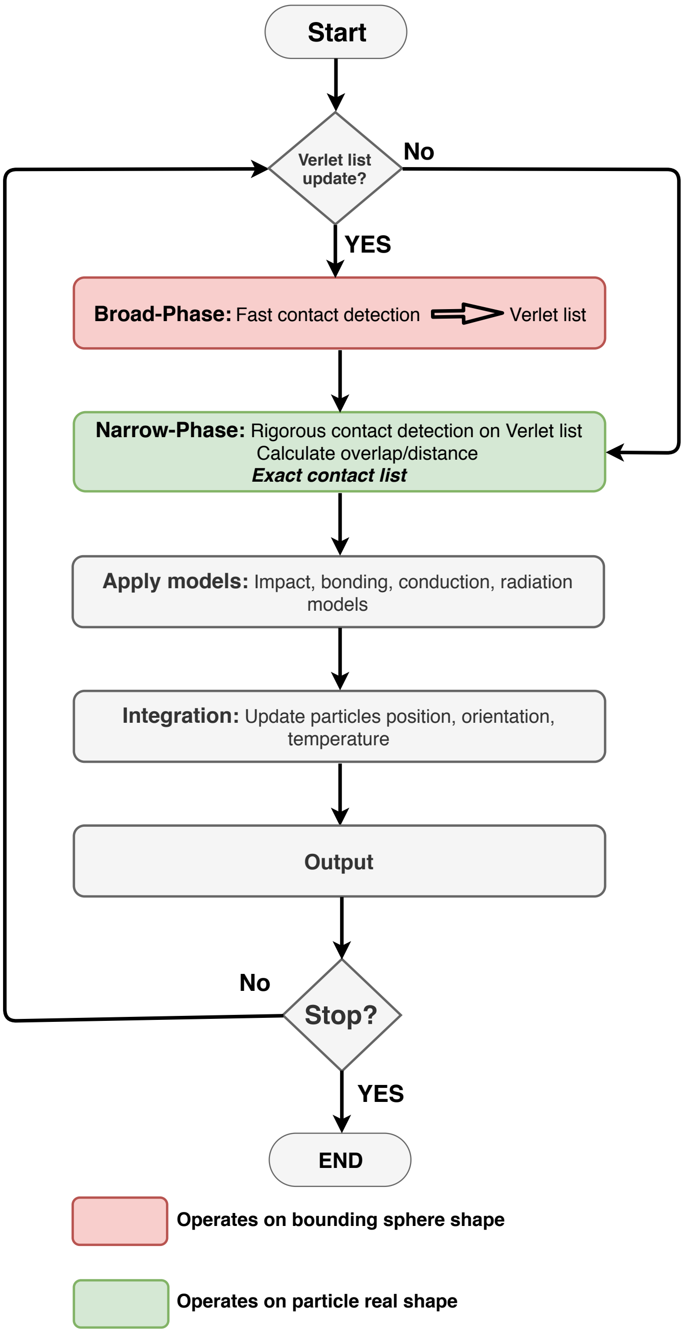

Start
Verlet list update?
No
YES
Broad-Phase: Fast contact detection
Verlet list
Narrow-Phase: Rigorous contact detection on Verlet list
Calculate overlap/distance
Exact contact list
Apply models: Impact, bonding, conduction, radiation models
Integration: Update particles position, orientation, temperature
Output
No
Stop?
YES
END
Operates on bounding sphere shape
Operates on particle real shape

FIGURE 4.5: New flow chart of XDEM software. A construction of the Verlet list is added. If the conditions are satisfied, the list is kept and the broad-phase is skipped and the simulation continues directly to the narrow-phase.

where v_p is the particle velocity in the system, Δt is the time step in the simulation, and K is the prescribed number of skipped steps in the broad-phase collision detection. This research started with Eq. 4.4 to determine the local skin parameter for each particle by replacing bulk velocity v with particle local velocity v_p.

4.5.2 Automatic update and validity of the Verlet list

In this section, we detail a condition that allows determining if the Verlet list, computed during a previous time step, is still valid (Grindon et al., 2004).

Definition 1. *The Verlet list V_L at time t is correct if*

$$\|\vec{AB}\|_t \leqslant R_A + R_B \implies \{A, B\} \in V_L$$

We use the following condition as a way to determine if the Verlet list at time t is still valid.

$$\forall \text{ particle } p,\ \Delta x_p \leqslant skin_p \tag{4.5}$$

Where Δx_p is the particle displacement of the particle since the last broad-phase. If this condition is violated, it means that a new broad-phase must be executed to update the Verlet list.

Definition 2. *We define our* ***automatic Verlet update*** *scheme by:*

- *re-using the previous Verlet list while condition 4.5 is still valid;*
- *re-executing a new broad-phase to update the Verlet list otherwise.*

We will now prove that our proposed automatic Verlet update scheme always returns the correct results.

Theorem 1. *The automatic Verlet update scheme defined at 2 ensures that all pairs of particles that can possibly collide are in the Verlet list.*

The claim in theorem 1 avoids the potential simulation accuracy and stability issues that take place in the fixed update interval scheme, as reported in (Chialvo and Debenedetti, 1990). The proof of the theorem is provided in the following step:

Proof. We assume that the initial broad-phase executed at time t, performed on the extended bounding spheres, is correct and returned the correct Verlet list, and thus by definition 1, we have:

$$\|\vec{AB}\|_t \leqslant R_{NLA} + R_{NLB} \implies \{A, B\} \in V_L, \tag{4.6}$$

Where R_{NLA} and R_{NLB} are the respective extended interaction ranges of particle A and particle B.

We want to show that, at any time $t^{'} = t + \Delta t$, the Verlet list is valid. According to the definition 1, that means that considering $\{A, B\}$ a pair of particles, we need to prove the hypothesis H0, that if A and B collide at time t', then $\{A, B\}$ is in the Verlet list V_L:

$$\|\vec{AB}\|_{t'} \leqslant R_{CA} + R_{CB} \implies \{A, B\} \in V_L, \qquad (H0) \tag{4.7}$$

Where R_{CA} and R_{CB} are the respective cutoff distance of particles A and B.

It exists two possibilities:

1. The condition 4.5 is no more valid, i.e.:

$$\exists A \mid \|\vec{AA'}\|_{t'} > skin_A, \tag{4.8}$$

   where A is a particle at a given position at time t, A' the same particle in a new position at t' time. In that case, a new broad-phase, on the extended bounding spheres, has to be executed at time t', and a new correct Verlet list is generated. So if we consider two colliding particles A and B at time t', then we have

$$\begin{aligned}
\|\vec{AB}\|_{t'} &\leqslant R_{CA} + R_{CB} && \text{because A and B are interacting} \\
&\leqslant R_{CA} + skin_A + R_{CB} + skin_B \\
&\leqslant R_{NLA} + R_{NLB}
\end{aligned} \tag{4.9}$$

and because the newly generated Verlet list is correct, then we have $\{A, B\} \in V_L$. The hypothesis H0 is verified.

2. The condition 4.5 is still valid, i.e.:

$$\forall A,\ \|\vec{AA'}\| \leqslant skin_A \tag{4.10}$$

where $\|\vec{AA'}\|_{t'}$ is the distance covered by particle A from time t to t'.

So if we consider two colliding particles A and B at time t', then we have

$$\begin{aligned}
&\|\vec{AB}\|_{t'} \leqslant R_{CA} + R_{CB} \\
&\implies \|\vec{AB}\|_{t'} + \|\vec{AA'}\| \leqslant R_{CA} + R_{CB} + skin_A && \text{by adding Eq. 4.10 for particle A} \\
&\implies \|\vec{AB}\|_{t'} + \|\vec{AA'}\| + \|\vec{BB'}\| \leqslant R_{CA} + R_{CB} + skin_A + skin_B && \text{by adding Eq. 4.10 for particle B} \\
&\implies \|\vec{AB}\|_{t'} + \|\vec{AA'}\| + \|\vec{BB'}\| \leqslant R_{NLA} + R_{NLB} \\
&\implies \|\vec{A'B'}\| + \|\vec{AA'}\| + \|\vec{BB'}\| \leqslant R_{NLA} + R_{NLB} && \text{as } \|\vec{AB}\|_{t'} = \|\vec{A'B'}\|
\end{aligned}$$

Additionally, from the triangle inequality, we have

$$\|\vec{AB}\| \leqslant \|\vec{AA'}\| + \|\vec{A'B}\| \leqslant \|\vec{AA'}\| + \|\vec{A'B'}\| + \|\vec{B'B}\|$$

and which finally gives us

$$\begin{aligned}
\|\vec{AB}\|_{t'} \leqslant R_{CA} + R_{CB} &\implies \|\vec{AB}\|_{t'} \leqslant R_{NLA} + R_{NLB} \\
&\implies \{A, B\} \in V_L && \text{because of Eq. 4.6}
\end{aligned}$$

The hypothesis H0 is verified.

□

4.6 Performance Evaluation

We carried out extensive numerical experiments to assess the performance of our proposed approach.

4.6.1 Methodology

In section 4.5, we presented our first model to establish the skin margin value of a particle:

$$skin_p = K \times v_p.\Delta t \tag{4.11}$$

where v_p is the particle velocity in the system, Δt is the time step in the simulation, and K is the prescribed number of skipped steps in the broad-phase collision detection.

Dynamic determination of the skin margin allows particles from different flow velocity to adopt distinct skin margins, but with the same K number of time steps between two consecutive updates of the Verlet list. What is the optimum K value giving the best computational efficiency? To answer this question, we performed a parameter study on the skin margin by varying the value of K. It varies from 0 to 5000 in the current study that was conducted on five different real test-cases with different purposes and flow velocity.

In the following XDEM simulations, we used the Velocity Verlet integration scheme and the Linear Spring Dashpot III contact model (only the static friction force is taken into account in the classical Linear Spring Dashpot model). All the five test-cases have been simulated for at least 5000 time steps. The Verlet buffer method is coupled with the linked-cell method with the constraint on the neighbor list range to be smaller than the cell size. This gives an upper-bound to the skin value

$R_{NL} <= \min_{x,y,z} L_C$. This means that in practice, the skin will be set for each particle independently to

$$R_{NL,p} = min(R_{C,p} + skin_p, \min_{x,y,z} L_C) \tag{4.12}$$

4.6.2 Test-cases

The following real-world example serves as concrete benchmarks for the evaluation of our implementation.

- **Hopper Discharge** The *Hopper Discharge* presented in Fig. 4.6 is a test case used with $125k$, $250k$ and $500k$ particles. The simulation works as follows: the selected number of particles (thus up to half a million) with different diameter are dropped off in a silo. Then the notch at the bottom is opened, letting all particles fall down into a chute. In this case study, the workload moves from the top portion of the domain downwards. Since the lower part of the silo is narrowed, the workload is focused on the center region of the domain.
- **Granular flows on vibrating rough inclined planes**

  The test case in Fig. 4.7 simulate a granular (spherical particles) flowing down on a roughed and incline plane. The particles are colored according to their velocity. In this test case, a silo is filled with particles of uniform size that are dropped off on an inclined plane. The latter has a roughed surface composed of many bigger particles, literally vibrating. In the free flow of particles, the ones on the top of the bed present a higher velocity because being the least in contact with the rough surface and therefore undergoing the least lateral vibrations.
- **Avalanche**

  Fig. 4.8 describes a simulation of an avalanche debut at the top of a habitable valley. Particles located upstream of the valley descend throughout the valley with an increased speed due to the inclined surface. The goal is to predict the path and the rate at which the avalanche will reach the bottom of the

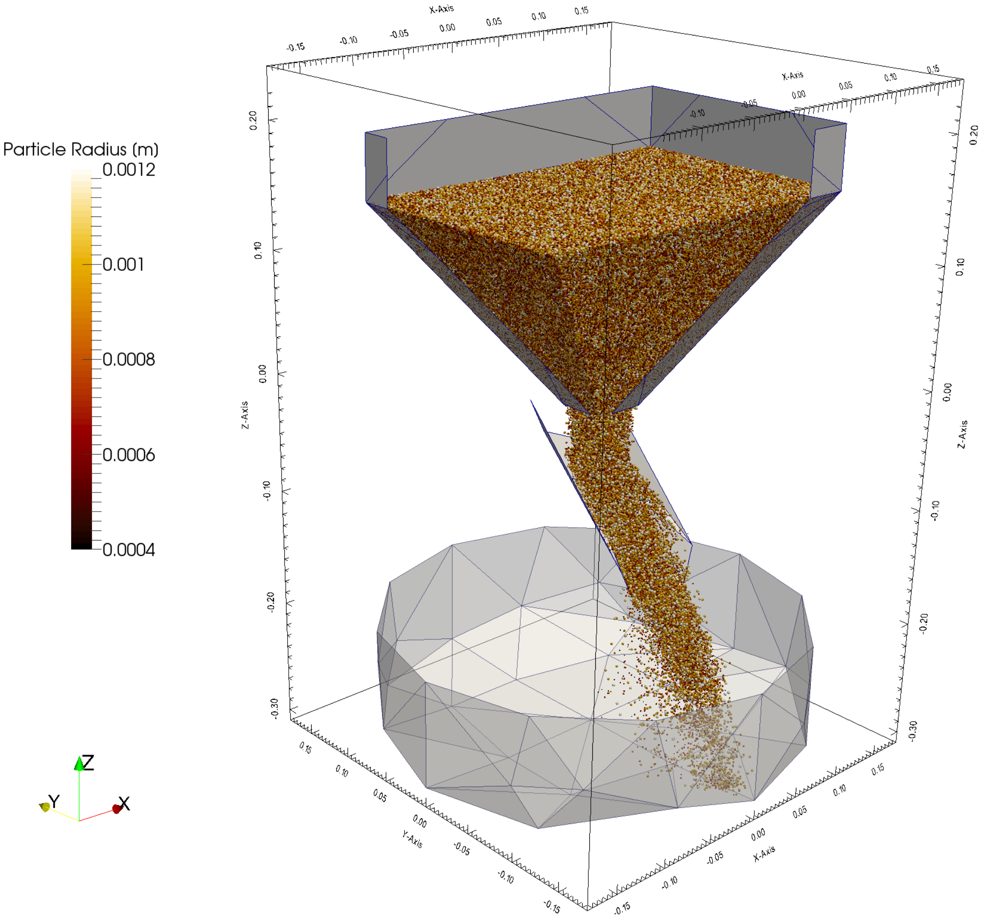


FIGURE 4.6: The test case used for the performance evaluation simulates the hopper discharge of 125k, 250k and half a million particles. It shows any overview of the set-up with the particles coloured according to their size.

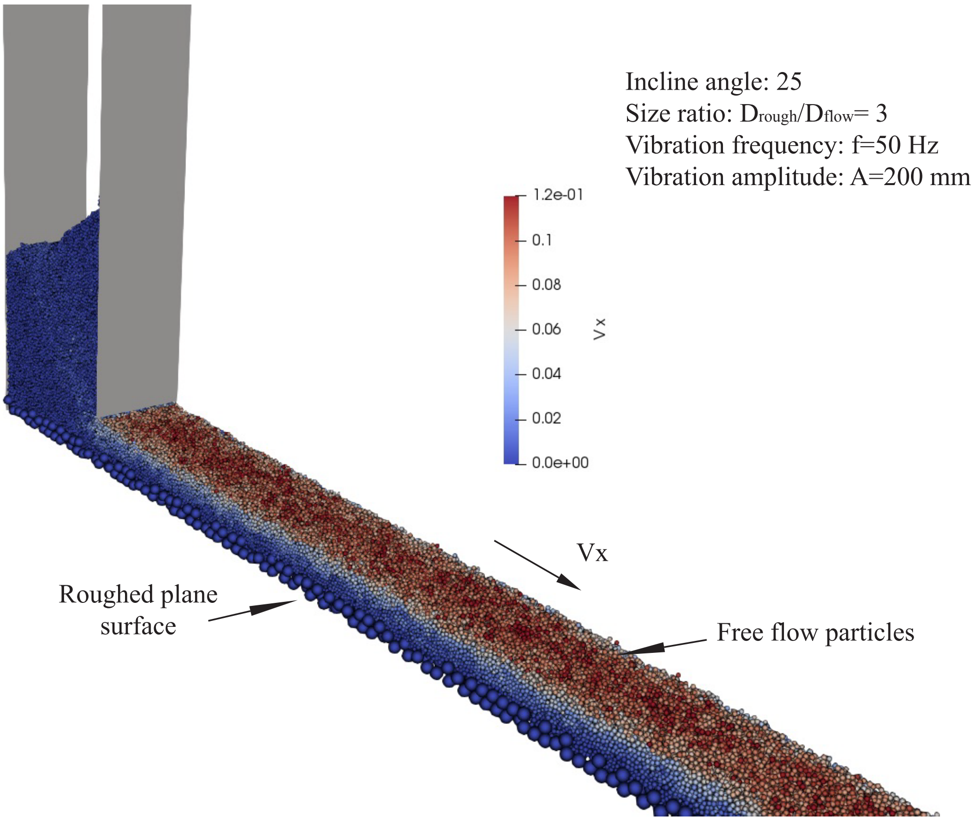


FIGURE 4.7: Granular flows on roughed inclined plane. The rough plane has particles vibrating at $50Hz$ frequency with $200mm$ amplitude. The free flow particles are coloured according to their velocity

valley at the housing level. The case, a cohesive model, uses the Elastic-plastic spring-dashpot rolling model.

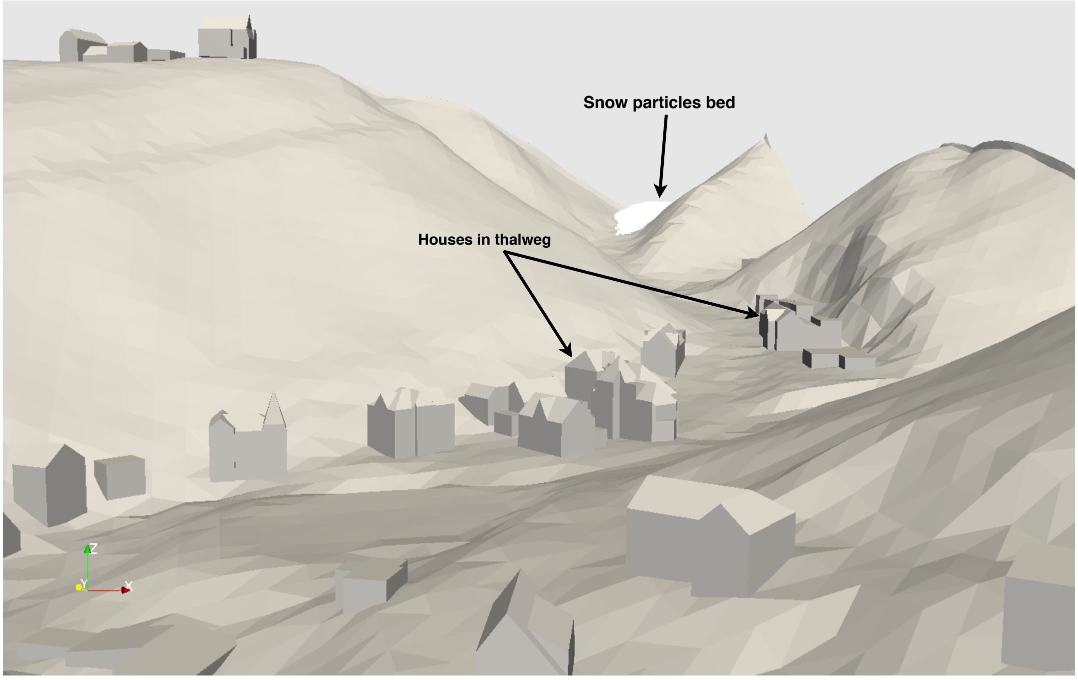


FIGURE 4.8: Simulation of an avalanche at the top of a habitable valley. The particles bed represents a cohesive snow model.

- **Biomass combustion**

  Fig. 4.9 shows a simulation of a combustion chamber of a 16 MW geothermal steam super-heater, which is part of the Enel Green Power "Cornia 2" power plant (Wikipedia contributors, 2019). The test case relies on a hybrid four-way coupling between the Discrete Element Method (DEM) and the Computational Fluid Dynamics (CFD). In this approach, particles are treated as discrete elements that are coupled by heat, mass, and momentum transfer to the surrounding gas as a continuous phase. For individual wood particles, besides the equations of motion, the differential conservation equations for mass, heat, and momentum are solved, which describe the thermodynamic state during thermal conversion.

- **Powder leveling for Selective Laser Melting in Additive Manufacturing**

  Additive manufacturing and specifically metal selective laser melting (SLM) processes are rapidly being industrialized (Donoso and Peters, 2018). The case showed in Fig. 4.10 simulates a Powder leveling for Selective Laser Melting

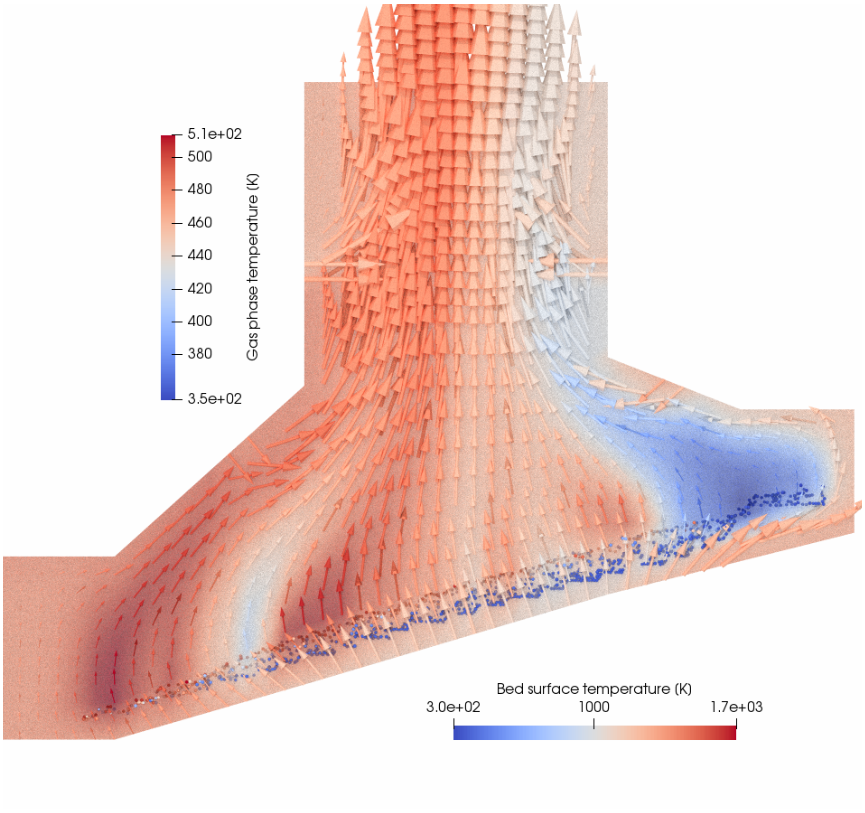


FIGURE 4.9: Furnace of the combustion of Biomass. The particle bed is arranged on four (4) moving grates. The bed is heated up in the combustion chamber by inlets located below the grates. The particles are colored according to their surface temperature.

(SLM) in Additive Manufacturing (AM) process. In this test case, an advanced discrete-continuous concept is used to address the physical phenomena involved during laser powder bed fusion. The concept treats the Powder as a set of particles by XDEM, predicting the thermodynamic state and phase change of each particle. The fluid surrounding is solved with multiphase CFD techniques to determine the momentum, heat, gas, and liquid transfer.

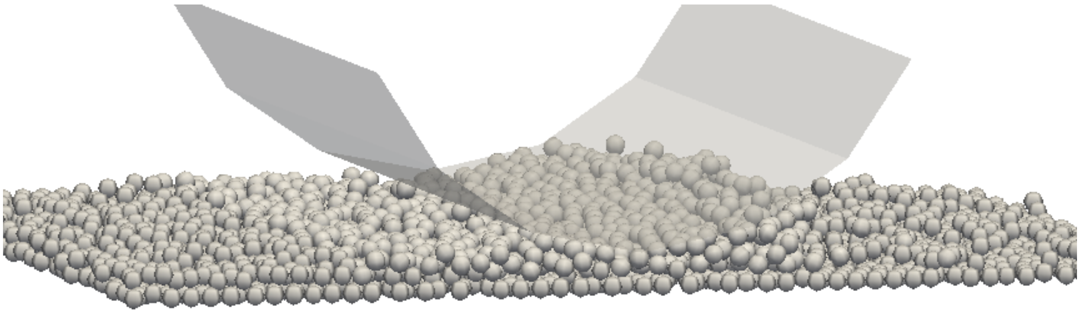


FIGURE 4.10: Powder levelling for Selective Laser Melting.

4.6.3 Experimental settings

The experiments were carried out using the `Iris` cluster of the University of Luxembourg (Varrette et al., 2014) which provides One hundred sixty-eight 168 computing nodes for a total of 4704 cores. The nodes used in this study feature a total a 128 GB of memory and have two Intel Xeon E5-2680 v4 processors running at 2.4 GHz, that is to say, a total of 28 cores per node. The nodes are connected through a fast, low-latency EDR InfiniBand (100Gb/s) network organized over a fat-tree topology.

We used version `67f029de` of XDEM software, compiled with GCC Compiler 6.4.0. The nodes were reserved for exclusive access in order to ensure the stability of the measurements. Additionally, each performance value reported in this section is the average of at least a hundred measurements. No major variance in the results was indicated by the standard deviation and is also not shown in the following graphs.

4.6.4 Results

The different simulations conducted in this study are intended to analyze and interpret the impact of the skin margin upon the performances of the automatic update algorithm presented in subsection 4.5. In this section, we take a look at how the $skin = Kv\Delta t$ model affects the test cases' computational times. The simulation time of the broad-phase, narrow-phase and apply models are illustrated in Fig. 4.11 as a function of K factor. We also compare our model of $skin = Kv\Delta t$ (skin is different for each particle) with the popular approach where the skin is uniform and equal to the particle radius (Angeles and Celis, 2019) (all cases are monodisperse and therefore the skin is identical for all the particles).

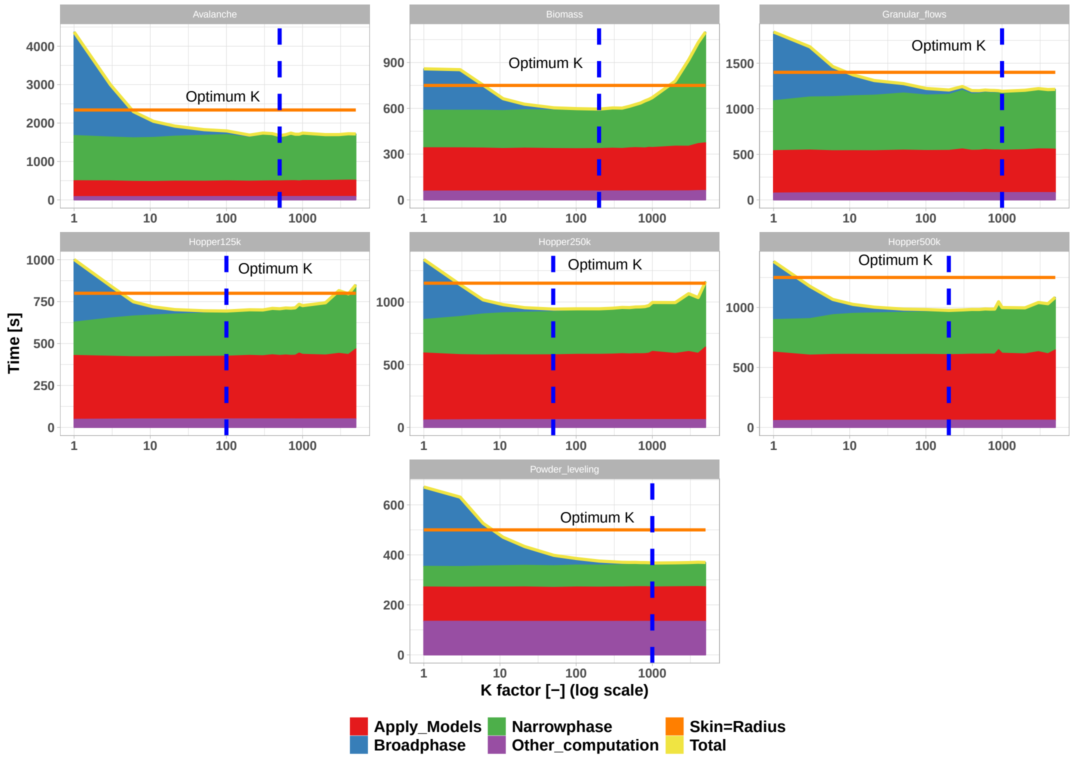


FIGURE 4.11: Dependence of broad-phase, narrow-phase and interactions models on skin K factor. The vertical blue dashed lines show the optimum K for each simulations corresponding to the lowest overall simulation time. The orange horizontal line represents the simulation time for a constant skin equal to the particles radius. The skin distance is capped by the cell size in all simulations.

It follows from Fig. 4.11 that:

- The broad-phase simulation time decreases with the K value and therefore with the *skin*. For the Avalanche case, BP[a] $= 3172.758s$ without the Verlet buffer method and $BP = 42.418s$ for $K = 400$, representing a 98.66% of time improvement.

Increasing the K factor decreases the overall broad-phase time but does increase a single broad-phase time due to the extend of the neighborhood (more pair of particles in the Verlet list). But this goes hand in hand with a decrease in the number of performed broad-phase, which decreases the overall time.

- The narrow-phase simulation time increase with the K value and thus with $skin$. For the Biomass case, NP[b] $= 288.155s$ without the Verlet buffer method and $NP = 323.626s$ for $K = 1000$. It is a 10.96% increase in the narrow-phase time.

  The rise is due to the enlargement of the Verlet list during an increase of K, on which the narrow-phase performs an exact collision detection.

- When K increases, the simulation time decreases to a low peak before starting to increase for almost all the cases, especially for the Biomass case. Without using the Verlet buffer method, simulation time equals to $962.901s$ and equals to $594.899s$ for $K = 200$ that is an increase of 61.86% of speed up overall. But at $K = 5000$, the overall simulation increases to $1100.208s$, an increase of 12.48% compare to no Verlet buffer method.

- The most improvement is achieved with the AVALANCHE test case with a 70% of simulation time improvement. The least gain is obtained with the BIOMASS test case with a 38% of time improvement. The latter can be explained by the fact that it includes a CFD coupling with OPENFOAM, adding, therefore, more computations. The percentage gain relative to broad-phase is similar to in the AVALANCHE test.

  The behavior confirms the existence of an optimum K value for the Verlet buffer method. The decrease in the simulation time observed at the beginning is a result of the two preceding bullet points. Undoubtedly, the values of K after zero offer a larger gain in broad-phase time than the increase noticed in

---

[a] Broad-phase simulation time

[b] Narrow- phase simulation time

the narrow-phase. But after a value referenced as the optimum K value, the increase in the narrow-phase time is more significant than the decrease in the broad-phase time resulting in an increase of the overall simulation time.

- At a fixed state condition, the optimum K increases with the system size due to the density and cell size change. For the hopper discharge case, $K_{optimum} = 100$ for $N = 125k$ particles and $K_{optimum} = 200$ for $N = 500k$ particles.
- The simulation time does not increase at any point but stabilizes rather at a minimum after a clear decrease as the interaction radius reaches the minimum cell size for the powder laser melting case.
- We can notice that our current approach of a dynamic skin gives a better performance when reaching the optimum skin. The difference is observable even for a case(biomass furnace) with a relatively homogeneous flow regime.
- The optimum skin distance and K value depend on the test case and, therefore, on several parameters as the solid fraction, the cell size, the ratio of particle to cell size, and the number of particles. A study on how to compute the optimum skin distance depending on the aforementioned parameters is presented in our paper (Checkaraou et al., 2018b)

In summary, the bigger K is, the wider the interaction range is, and the wider the neighborhood is, involving a reduction in broad-phase and simulation overall time, although the narrow-phase is increasing. Then, after K reaches an optimum value, the decrease time in the broad-phase does no longer compensate the increased time in the narrow-phase leading to an increase of the overall simulation. This behavior is observed when interaction never reaches the cell size. When it reaches the cell size, the simulation time remains unchanged since the skin margin is down to the value of the minimum cell size. The number of executed broad-phase as a function of K value is shown in Fig. 4.12. There is a clear decrease in the number of performed broad-phase when increasing the K (increase in the number of skipped broad-phase), but an equilibrium is reach around $K = 200$. It means that after this value, there should be no more significant gain in skipping the broad-phase. It appears from all

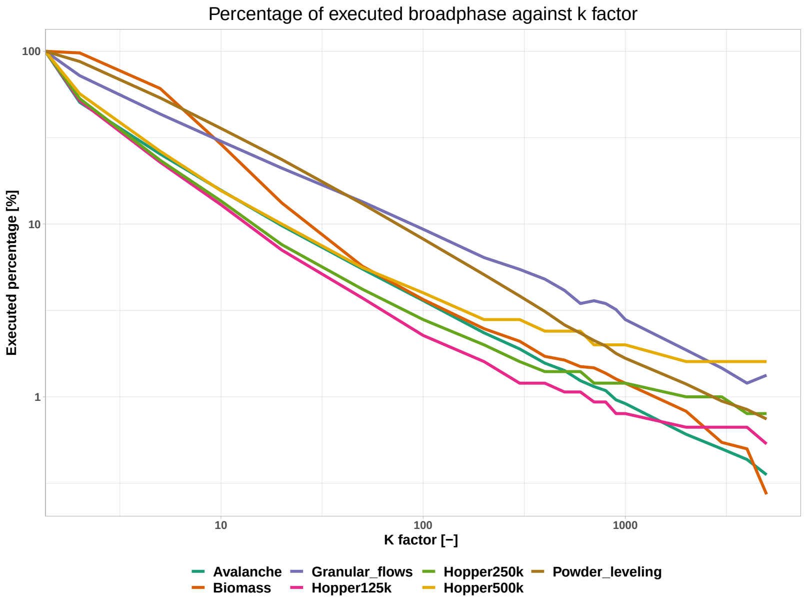


FIGURE 4.12: Dependence of executed broad-phase in percentage upon the skin K factor. The percentage correspond to the number of executed broad-phase over the total number of steps in the simulation.

our simulations that the biggest drop in the simulation time is made around $K = 10$ and $K = 50$, although there is a clear gain by increasing the skin after those values.

Fig. 4.13 enables us to put our previous observations (not significant gain after $K = 200$) into perspective. It shows the time overhead for several K values compared to the optimum time value. It confirms that the biggest time drop is made around $K = [10 - 50]$ for all the test cases. It also helps to notice that $K = 200$ is a value close to the optimum for all the test cases and can thus be chosen as a common and default value.

The table 4.1 presents a performance comparison between the optimum case, considered as the K value given the lower simulation time, which is specific to each test case and a default case. The latter is defined as the K value given an excellent performance compromise for any case. In table 4.1, the improvement in percentage is defined by the gain made compared to a simulation without using the Verlet buffer method. It is given by the following formula:

$$Improvement = \frac{Time_{w/o\ Verlet\ buffer} - Time_{case}}{Time_{w/o\ Verlet\ buffer}} \times 100 \tag{4.13}$$

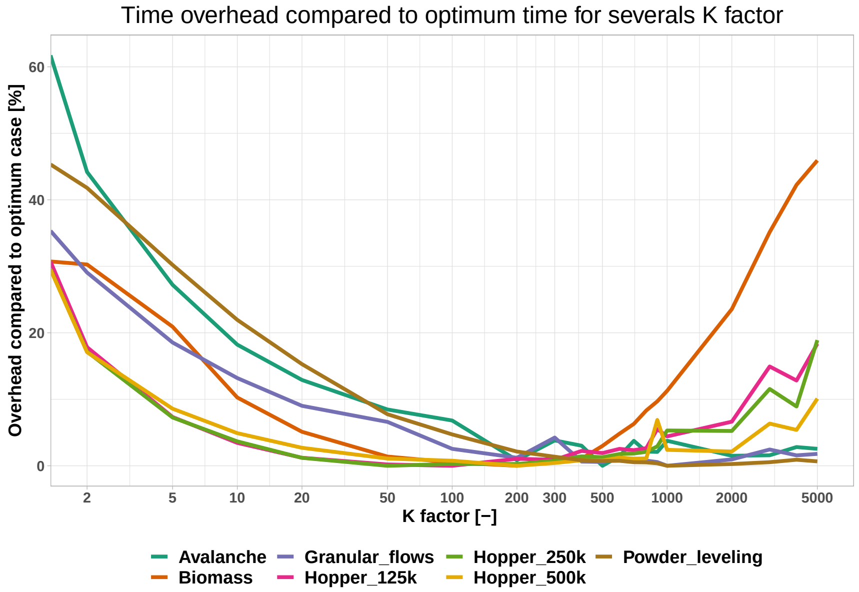


FIGURE 4.13: Simulation time overhead compared to the optimum for each K value for all test cases.

where $Time_{case}$ is the simulation time depending on which value of K is used for the Verlet buffer method.

The OVERHEAD column corresponds to the time difference (percentage) between the case where the optimum K value is used and the default case with the acceptable K value. We can notice from table 4.1 that a default $K = 200$ is an excellent compromise to the optimum K value that can be used for all the test cases. Actually, it has a maximum overhead of 2% when used for all the test cases. We then recommend choosing $K = 200$ when using the Verlet buffer method as a good arrangement to the optimum value.

4.7 Conclusion

In this article, we proposed a local Verlet buffer solution with a new skin formulation that expresses each particle's skin margin according to the neighborhood flow conditions and based on the particle velocity. The method has also been implemented in our home software with an automatic update scheme for DEM simulations of granular material. It is an improvement and a generalization of the conventional Verlet list method for DEM simulations. The method allows to keep several time

TABLE 4.1: Summary of the performance results of the Verlet buffer method over the different testcases.

Testcase	**Without Verlet buffer**	**Optimum Value for K**			**Selected Default Value $K = 200$**		
	Simulation time [s]	$K[-]$	Simulation time [s]	Improvement [%]	Simulation time [s]	Improvement [%]	Overhead to optimum [%]
Avalanche	5595.77	500	1673.18	70.12	1687.24	**69.84**	0.83
Biomass	962.90	200	594.89	38.21	594.89	**38.21**	0.00
Granular flows	2084.74	1000	1191.64	42.83	1206.19	**42.14**	1.20
Hopper 125k	1164.40	100	694.14	40.38	701.52	**39.75**	1.05
Hopper 250k	1564.46	50	943.14	39.71	945.67	**39.55**	0.26
Hopper 500k	1614.37	200	975.27	39.58	975.27	**39.58**	0.00
Powder leveling	733.98	1000	367.22	49.96	375.25	**48.87**	2.14

steps the potentially interacting pairs of particles list by surrounding the particle cut-off radius by a skin margin, in case the contact detection is divided into two steps: the broad and narrow phases. It has the advantage of giving the possibility to use any contact detection algorithm to generate the approximate interacting pairs of particles list during the broad-phase process.

We presented a new skin margin formulation based on individual-particle displacements for easy implementation and better consider the different flow velocity regimes that often coexist in granular flow DEM simulations. A parameter study on the skin margin was conducted to assess its effects on the method performances. It appears as expected, a decrease in the broad-phase overall time and an increase in the narrow-phase time while increasing the skin margin, resulting in a global decrease in simulation time. Beyond specific skin margin values, we found an opposite effect where the increase in the narrow-phase time is too high, resulting in a global simulation time increase. The study showed that most computational time gain is made around $K = 20$, and there is often after that value some gains to make, but not as significant.

Chapter 5

Predicting near-optimal skin distance in Verlet buffer for DEM

5.1 Abstract

The Verlet list method is a well-known bookkeeping technique of the interaction list used both in Molecular Dynamic (MD) and the Discrete Element Method (DEM). The Verlet buffer technique enhances the Verlet list that consists of extending each particle's interaction radius by an *extra margin* to take into account more particles in the interaction list. The *extra margin* is based on the local flow regime of each particle to account for the different flow regimes that can coexist in the domain. However, the choice of the near-optimal *extra margin* (which ensures the best performance) for each particle and the related parameters remains unexplored in DEM, unlike in MD.

In this study, we demonstrate that the near-optimal *extra margin* can fairly be characterized by four parameters that describe each particle local flow regime: the particle velocity, the ratio of the containing cell size to particle size, the containing cell solid fraction, and the total number of particles in the system. For this purpose, we model the near-optimal *extra margin* as a function of these parameters using a quadratic polynomial function. We use the DAKOTA SOFTWARE to carry out the Design and Analysis of Computer Experiments (DACE) and the sampling of the simulations' parameters. For a given instance of the set of parameters, a global optimization method is considered to find the near-optimal *extra margin*. The latter is required for the construction of the quadratic polynomial model. The numerous simulations generated by the sampling of the parameter were performed on a High- Performance Computing (HPC) environment granting parallel and concurrent executions.

This work provides a better understanding of the Verlet buffer method in DEM simulations by analyzing its performances and behavior in various configurations. The near-optimal *extra margin* can reasonably be predicted by two out of the four chosen parameters using the quadratic polynomial model. This model has been integrated into XDEM to choose the *extra margin* automatically without any input from the user. Evaluations on real industrial-level test cases show up to 26% of reduction of the execution time [a].

---

[a] This chapter was published as an article in IPDPS2020 ("Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method")

Keyword

Verlet, DEM, HPC, Optimization, Dakota

5.2 Introduction

The Extended Discrete Element Method (XDEM) is an advanced numerical process that enhances the granular material properties in classical Discrete Element Method (DEM) (Cundall and Strack, 1979) by supplementary states such as thermodynamic, stress, and strain (Peters, 2013). As part of the ongoing XDEM software development, we have implemented an optimized variant of the Verlet list (Verlet, 1967) using the local flow regime of each particle ("Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method", *submitted to Advances in Engineering Software*). This method, called *Verlet buffer*, is applied to the broad-phase and takes advantage of the temporal coherency of a DEM simulation.

Multiple works in the literature have studied the behavior of the Verlet-list for Molecular Dynamics (MD) (Verlet, 1967; Chialvo and Debenedetti, 1990; Chialvo and Debenedetti, 1991; Mattson and Rice, 1999; S. and S., 2006), in particular how far the list can be expanded and what are the primary influence parameters. However, no such complete study exists for DEM simulations. (Li et al., 2010) compared in DEM the performances of the Verlet list and linked-cell methods in a gravity-driven granular collapse simulation. It showed that special care must be carried when choosing the interaction range (skin + cut-off distance), cell size, and updating interval time as they are crucial to obtain the best performance out of the Verlet list method. The latter interaction range is controlled and adjusted by the user through the skin distance.

In XDEM, this skin distance parameter does not change the result of the simulation, but it can significantly influence the performance of the execution ("Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method"). The best value for the skin distance factor typically depends on each input case and the local conditions on each part of the case. As described in the upper

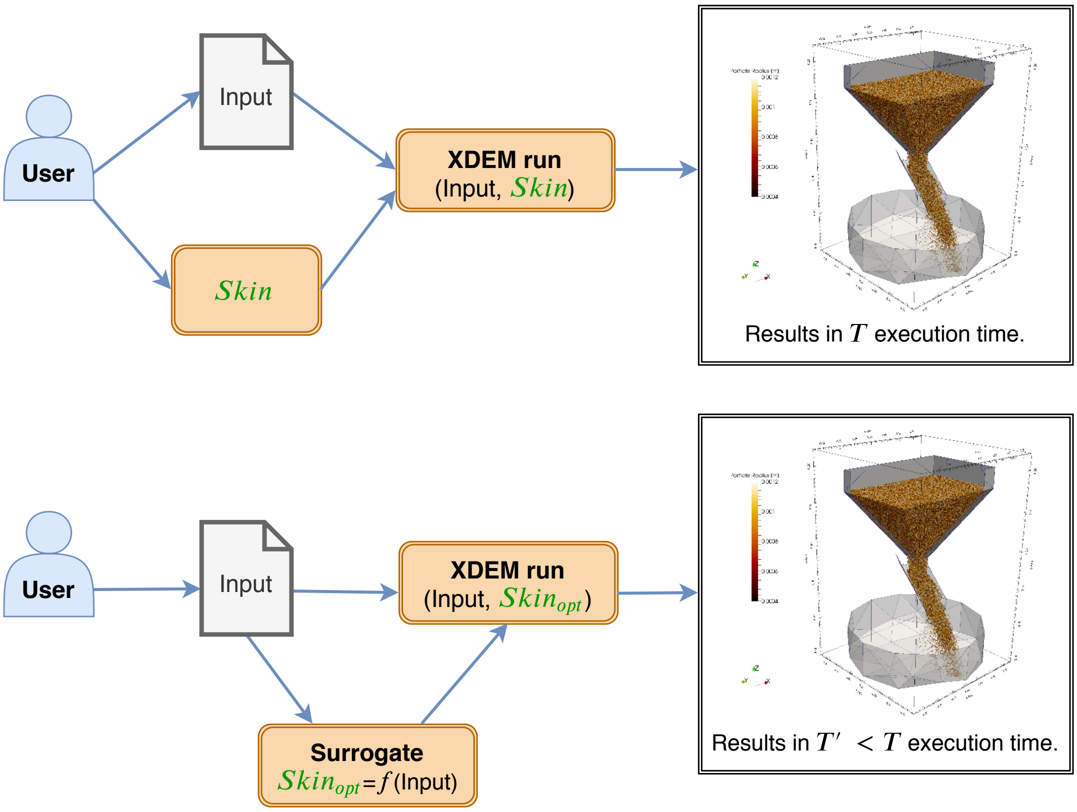


FIGURE 5.1: XDEM user specifications.

diagram of Fig. 5.1, the user can specify the skin distance along with the input case when running its XDEM simulation. However, finding the best value for the skin distance factor can be difficult and time-consuming. Thus, the work presented in this article focuses on finding an intuitive approach to optimize this hyper-parameter. As shown in the bottom part of Fig. 5.1, our goal is to construct a surrogate function that determines, at low-cost, a near-optimal value for the skin distance to speed-up the XDEM simulation.

The literature is replete with numerous hyper-parameter optimization algorithms (e.g., Grid Search Bergstra and Bengio, 2012, Bayesian Optimisation Snoek, Larochelle, and Adams, 2012, etc ...). Nevertheless, this meta-optimization approach should be performed for different simulation contexts. Therefore, the near-optimal skin distance value is a function of the simulation inputs and can be approximated using a surrogate/predictor function. Our contribution relies on this surrogate function's definition using a training data set obtained by optimizing the skin distance on a pre-established number of simulation cases.

The remainder of this article is organized as follows. Section 5.3 introduces the XDEM simulation toolbox and the problem around the Verlet buffer method. Section 5.4 formalizes the skin distance optimization problem and then explicit the methodology to tackle it. Section 5.5 describes the design and the implementation of a surrogate function to characterize the near-optimal skin distance for different inputs. Experiments using the DAKOTA SOFTWARE are highlighted in section 5.6. Finally, the last section 5.7 concludes this work and proposes future investigations.

5.3 Background

5.3.1 The Extended Discrete Element Method

The XDEM software is a multiphysics toolbox (Samiei and Peters, 2010) based on the dynamics of granular material described by the classical DEM (Cundall and Strack, 1979; Allen and Tildesley, 1990). It extends the usual particle properties with thermodynamic state, stress/strain or electromagnetic field (Peters, 2013; Peters and Pozzetti, 2017; Mahmoudi et al., 2016a). The software is written in C++ and is composed of a set of modules: *Dynamics* governing the motion of the particles, *Conversion* for the chemical conversion and thermodynamics, *CFD Coupling* for the coupling through an external CFD library such as OpenFOAM (Jasak, Jemcov, and Tukovic, 2007). Each of the simulation modules can be enabled separately and have specific time settings. The XDEM simulation driver is responsible for executing at each iteration the necessary phases for the activated modules.

An XDEM simulation is an *iterative* time loop which contains the following main phases:

- The **Broad-Phase** uses a fast but approximate collision detection to build a list of particle pairs that can potentially interact. During this phase, the particles are represented by bounding spheres with an appropriate radius (express the interaction range).

- The **Narrow-Phase** processes the list of potentially interacting particle pairs and performs a precise contact detection using the actual shape of a particle (*e.g.* sphere, cube, disk, cylinder, triangle-based shape). It calculates the overlap/distance, the contact point, and the direction between the two particles.
- **Apply Physics Model**: this phase selects the physics models defined in the particle properties (*e.g.* for impact, bonding, rolling, conduction, radiation, chemical reaction) and calculates the contribution to each particle involved (in term of force, torque, heat flux, chemical species mass fraction).
- The **Integration** updates the state of the particles after accumulating the contributions of all the interactions. Different integration schemes are available for the different components of the state of the particles (*e.g.* position/orientation, temperature, chemical composition).

To leverage large-scale HPC platforms, XDEM supports parallel distributed and shared-memory executions based on MPI and OpenMP (Checkaraou et al., 2018a; Besseron et al., 2013).

5.3.2 Verlet buffer method for XDEM

The *Verlet buffer* method is an enhancement of the broad-phase, which is detailed in ("Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method"). Instead of calculating the list of potential interactions for the direct neighboring particles, it considers particles that are further away from each other by increasing their bounding spheres. As a result, the interaction list also contains potential future interactions, and then the broad-phase does not need to be re-executed at every iteration. As shown on Fig. 5.2, the interaction range of each particle, or cut-off radius R_C, is extended by a skin distance, $skin$, that offers a broader neighbourhood, or neighbour list radius R_{NL}:

$$R_{NL} = R_C + skin \tag{5.1}$$

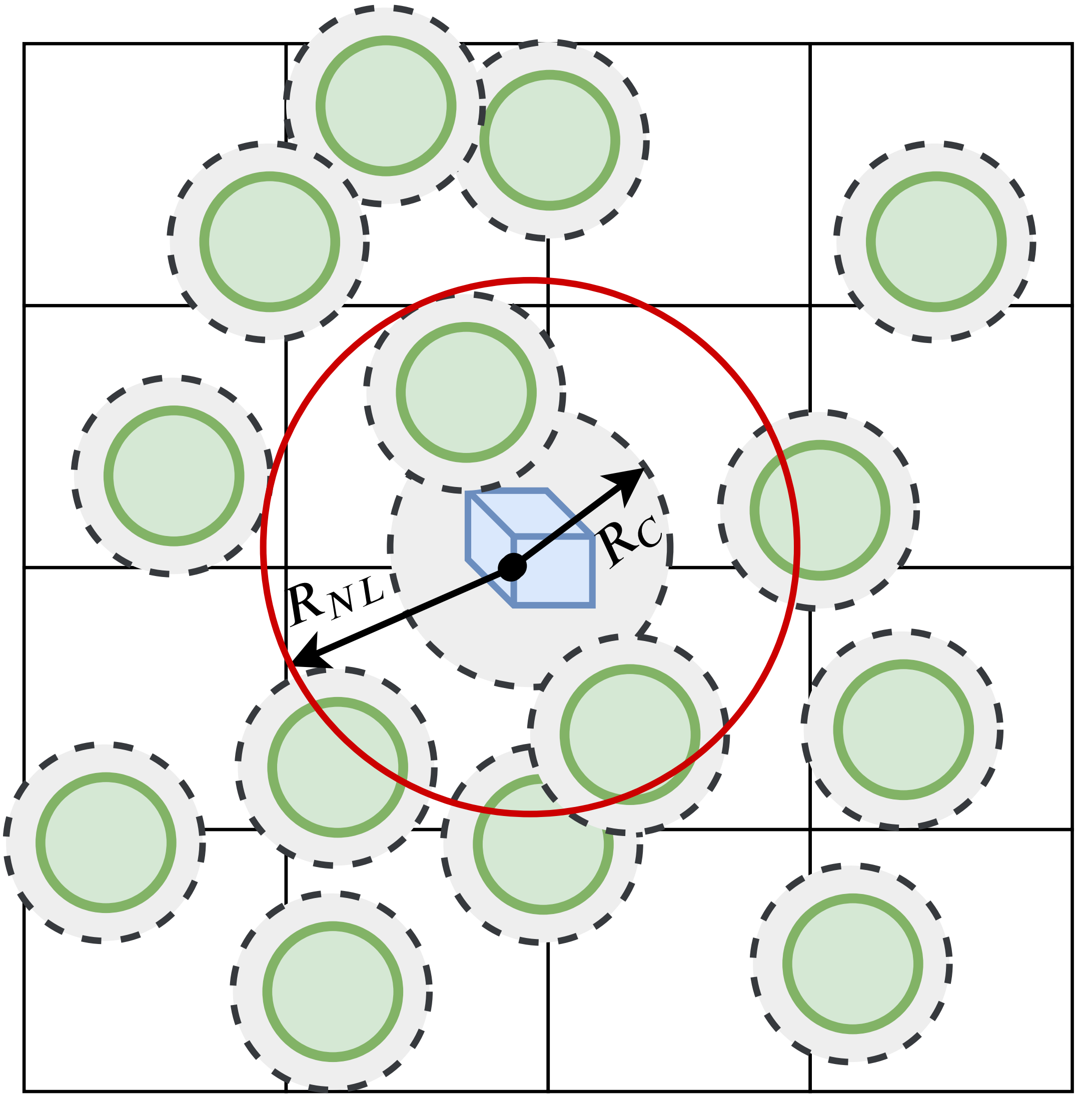


FIGURE 5.2: In the Verlet buffer approach, the bounding sphere of the particles is extended by the skin distance such as $R_{NL} = R_C + skin$.

In contrast to the conventional Verlet list (Allen and Tildesley, 1990; Allen and Tildesley, 2017), the Verlet buffer does not create a neighborhood list for every particle, but rather a global list of interacting particle pairs referred to as the Verlet list. Additionally, because it merely applies to each particle's bounding sphere, our approach applies to any broad-phase algorithm ("Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method"; Rousset et al., 2018). We also propose a condition that enables the previous broad-phase results (i.e., the list of potential interactions) to be ascertained. It will compel the execution of a new broad-phase if this condition is broken. In any case, the narrow-phase is always performed on this approximate interaction list and ensures that the results are the same as our approach.

In practice, the skin distance of every single particle is unique in the local buffer procedure and is calculated according to local flow conditions, *e.g.* the particle velocity. The skin distance is initially determined according to a borrowed algorithm to Molecular Dynamics (Chialvo and Debenedetti, 1990):

$$skin = K \times V_p \times \Delta t, \tag{5.2}$$

where K is called the skin distance factor, V_p is the particle velocity in the system, and Δt the DEM time step interval.

5.3.3 Dakota Software Package

The DAKOTA (Design Analysis Kit for Optimisation and Terascale Applications) toolkit provides a flexible and extensible interface between simulation codes and iterative analysis methods. DAKOTA SOFTWARE contains algorithms for optimization with gradient and non-gradient-based methods; uncertainty quantification with sampling, reliability, and stochastic expansion methods; parameter estimation with non-linear least squares methods; and sensitivity/variance analysis with design of experiments and parameter study methods. By employing object-oriented design to implement abstractions of the critical components required for iterative systems analyses, the DAKOTA SOFTWARE toolkit provides a flexible and extensible problem-solving environment for design and performance analysis of computational models on high-performance computers.

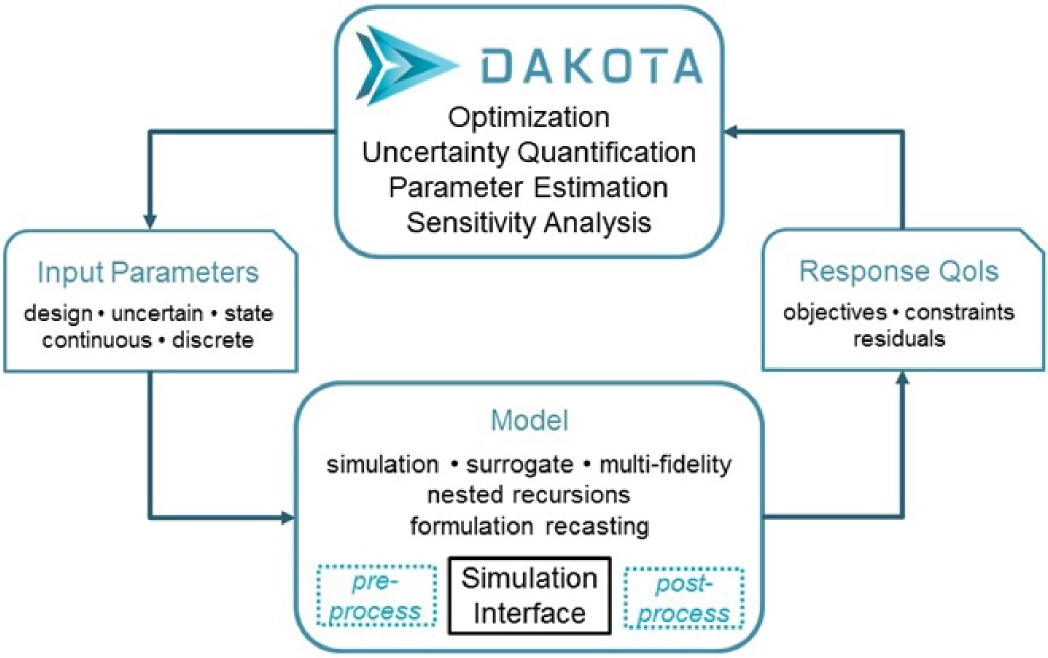


FIGURE 5.3: The loosely-coupled or "black-box" interface between DAKOTA SOFTWARE and a user-supplied simulation code (Adams et al., 2019).

The DAKOTA SOFTWARE supports multiple optimization capabilities, including gradient-based, derivative-free methods local, and global methods (as Genetic Algorithm used in this study) for use in science and engineering design applications (Adams et al., 2019). The toolkit also supports multi-objective and surrogate-based optimization. The methods and algorithms in DAKOTA SOFTWARE are designed to exploit parallel computing resources such as those found in a desktop multiprocessor workstation, a network of workstations, or a massively parallel distributed computing platform. For more details, one should refer to DAKOTA SOFTWARE Users Manual (Adams et al., 2019).

In this research, the optimization and surrogate capabilities have been combined in a nested model. High-performance capabilities have also been used through coarse and fine-grained parallelism on a cluster node.

5.4 Skin distance optimisation problem

This section discusses the optimization process and methods used to find the near-optimal skin distance for a given input case.

5.4.1 Definition

In the Verlet buffer approach, the skin distance parameter can be tuned to provide better performances. Each particle's neighborhood depends on the skin distance, which once enlarges, increases the size of the interaction list in the broad-phase. A broad list means that the broad-phase execution can be skipped more often, and the list could be kept for longer (time-steps). That is where the gain in time is achieved. Nevertheless, the narrow-phase executed at each time-step has now to be performed on a more extensive list, which has a negative impact on the performance, i.e., an increase in computing time. The computing time gained in the broad-phase may not compensate for the time lost in the narrow-phase leading to an increase in overall computational time. The simulation time is, therefore, a non-linear function of the skin distance parameter. To find the best trade-off between the two phases, one needs to find the near-optimal skin distance according to the overall computing time. As an example to motivate the search of this optimum, Fig. 5.4 depicts a real-case related to biomass simulation. This example illustrates the existence of such near-optimal skin distance and shows the simulation of a combustion chamber of a 16 MW geothermal steam super-heater, which is part of the Enel Green Power" Cornia 2" power plant (Wikipedia contributors, 2019). It relies on a hybrid 4-way coupling method between the XDEM and the OpenFOAM toolboxes (XDEM+OpenFOAM).

Fig. 5.4 shows the simulation time of the broad-phase, narrow-phase, and application models as a function of the skin distance factor (K). The skin distance factor K varies from 0 to 2000 in current experiments. The vertical blue dashed line shows the best skin distance factor K, corresponding to the lowest overall simulation time. It confirms the existence of a near-optimal skin distance factor K and, therefore, of a near-optimal skin distance.

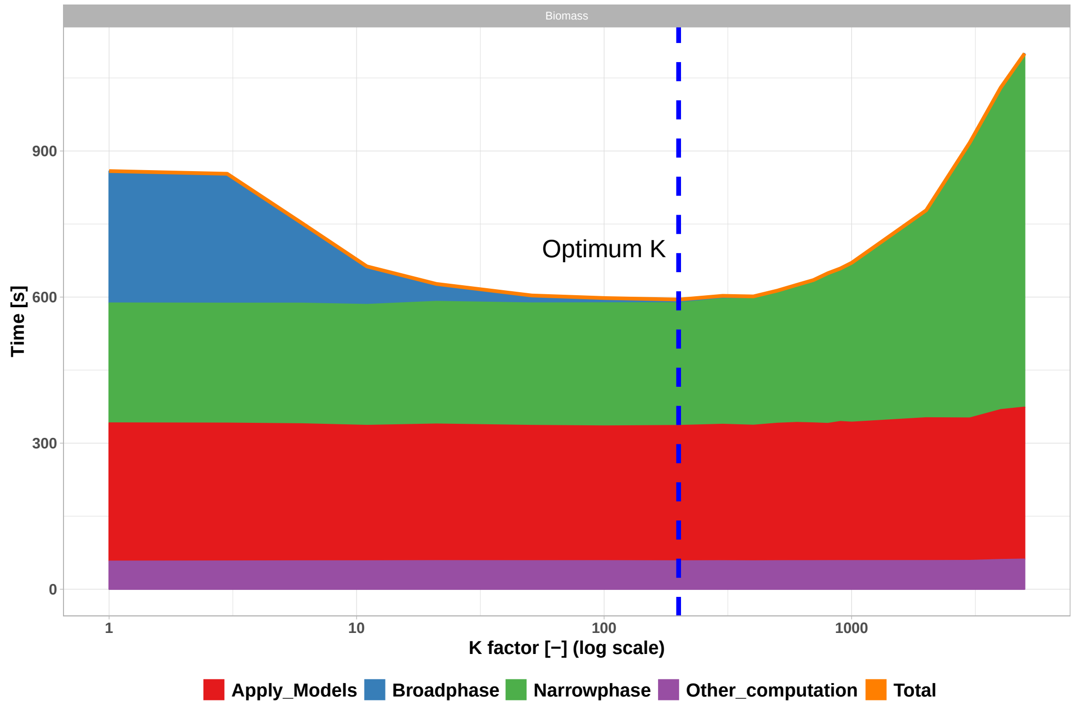


FIGURE 5.4: Dependence of broad-phase, narrow-phase and interactions models on skin distance factor K.

5.4.2 Evolution optimisation of the skin distance parameter

Due to the non-linear property of the skin distance optimization problem, Evolutionary Optimisation (EO) has been considered. EOs are bio-inspired optimization algorithms that belong to the class of meta-heuristics. They have been widely used in single-level optimization cases to tackle NP-hard problems. Among the EOs algorithms, Genetic Algorithms (GAs) have been retained to determine the near-optimal skin distances. Since the execution time of the simulations is subject to noise, gradient-based approaches are prohibited. XDEM simulations have been carried out on exclusive nodes with pinned threads to reduce those fluctuations as much as possible. Thus, a global and derivative-free approach such as GA seems to be more robust, reliable, and inherently parallel for solving the skin distance optimization parameter.

The choice has been made to use the SOGA solver, a *Single Optimisation Genetic Algorithm* to solve the skin distance optimization problem since it supports parallel and concurrent execution on HPC systems. SOGA is a generational and population-based algorithm relying on Darwin's theory of evolution. In a nutshell, an initial population of solutions (i.e., skin distance) is randomly generated. This population is then evolved using natural selection principles. From generation to generation, promising genetic material is transmitted to offspring solutions and tends to convergence towards an optima which are not necessarily realized. To escape local optimal, punctual mutation of the solutions can modify solutions sensibly to keep enough diversification among the population (Haftka and Gürdal, 2012; Zames et al., 1981).

Table. 5.1 summaries the SOGA parameters considered in this work for solving the optimisation problem with the biomass case. DAKOTA SOFTWARE starts multiple evaluations on the *Iris* HPC cluster (Varrette et al., 2014). During initial population generation, *crossover* and *mutation*, the XDEM evaluations are made concurrently to speed up considerably the workflow of SOGA.

TABLE 5.1: SOGA parameters

Parameters	**SOGA**
Iterations	500
population size	10/20/50
Selection	Roulette Wheel
Crossover operator	SBX
Crossover probability	0.8
Mutation operator	Polynomial / Uniform
Mutation probability	0.2

A study has been conducted to observe the convergence rate for 10,20 and 50 individuals to determine the best population size. Fig. 5.5 illustrates the SOGA solver convergence rate for these different population sizes.

One can notice that the population size selection has a non-negligible effect on the number of simulations to perform. The benefit of using a population of 50 solutions is, in this case, minimal compared to the overhead cost. On the contrary, a population of 10 solutions converges too rapidly due to a lack of diversity. Finally, this approach uses a population of 20 solutions that show the best trade-off between the solution quality and the number of simulations executed. Last but not least, the population size also depends on computing power. In some cases, the simulation cost becomes too high and requires a small population for some practical results.

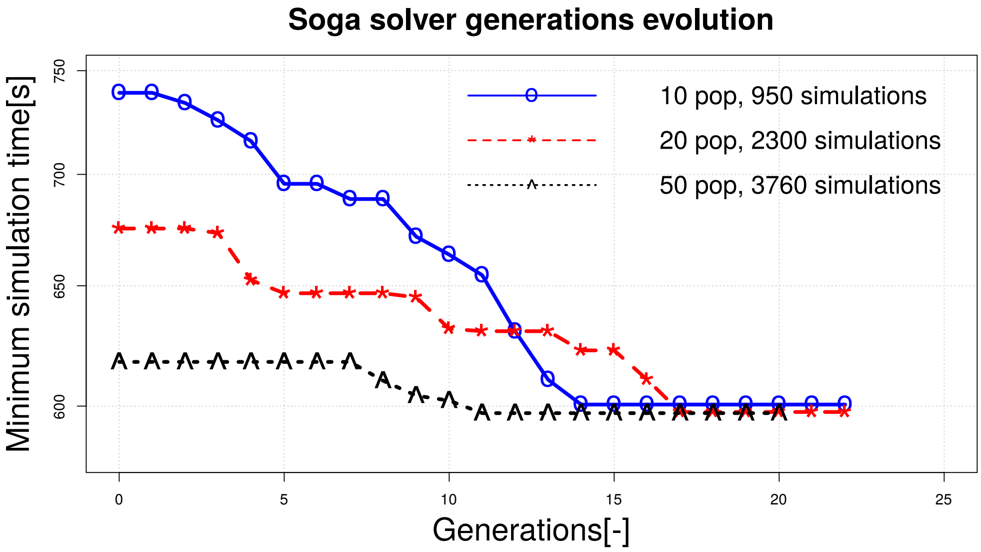


FIGURE 5.5: The SOGA solver convergence with the biomass case. Three populations with different initial sizes have been considered: 10, 20, 50.

5.5 Near-optimal skin distance characterisation

In the Verlet buffer approach, the near-optimal skin distance is case-dependent and can be characterized by a few parameters that define the particles' local flow regime. Indeed, it is not uncommon in particulate flow systems that both slow and rapid motion coexist. Dynamic determination of the skin distance grants particles from different flow regimes to adopt distinct skin distances. Local flow conditions decide the optimum. Therefore, when adopting a skin distance value for a DEM case, some physics parameters should be taken into account.

5.5.1 Design of experiments

In this study, four parameters describing particle local flow have been chosen to characterize the near-optimal skin distance:

- Solid or void fraction (S_f) of the cells, which defines the quantity particles or vacuum present in the cells;
- Ratio of (containing) cell size to particle size (C_s);
- Particle velocity (V_{el});

- Number of particles in the system (n).

The characterization can be summaries in five steps, as described in Fig. 5.6.

1. First, a sample covering the four parameters space is generated. It is used to construct XDEM box cases for the next step.
2. For each box, the optimisation problem describes in subsection 5.4.2 is solved and the skin distance factor K, K_{opt}, is returned.
3. Those K_{opt} are used as training data for the construction of a **predictor** model.

The boxes' training case is a homogeneous particle system, as shown in Fig. 5.7, in which parameters such as particle velocity, solid fraction, the ratio of cell size to the particle size and system size, can be easily varied. The mean particle size is $5mm$ with 10% variation of the size to prevent crystallization. The solid fraction varies from 0.1 to 0.6, and velocity magnitude is in the range of [0.1,20] with random direction at the beginning, and the ratio of cell size to the particle size is varied from 1.11 to 3.0 in the design of the experiment.

In the DEM simulation, the Hertz-Mindlin contact model is adopted with a reduced Young's module $Y = 5 \times 10^6$ to account for the stiffness of particles and cube walls. To arrive at a steady-state, energy dissipation due to friction and damping is ignored, which is achieved by setting friction coefficient $\mu = 0$ and coefficient of restitution $e = 1$. Initially, the particle is organized in a lattice structure and allowed to evolve until a steady state is reached, which usually requires a time duration of $10(\pi/6\phi)^{1/3}d_p/V_p$. Thereafter, the computational performance of each numerical experiment is evaluated for a time duration of $2(\pi/6\phi)^{1/3}d_p/V_p$. In the simulation, the DEM time step interval remains a constant value of 1×10^{-6} s. The gravitational acceleration is set to 0 in all simulations.

5.5.2 Methodology

To evaluate the relationship between the near-optimal skin distance (corresponding to the best computational efficiency) and the flow conditions, the particle system presented in the design of experiment section 5.5.1 is used to generate multiple boxes

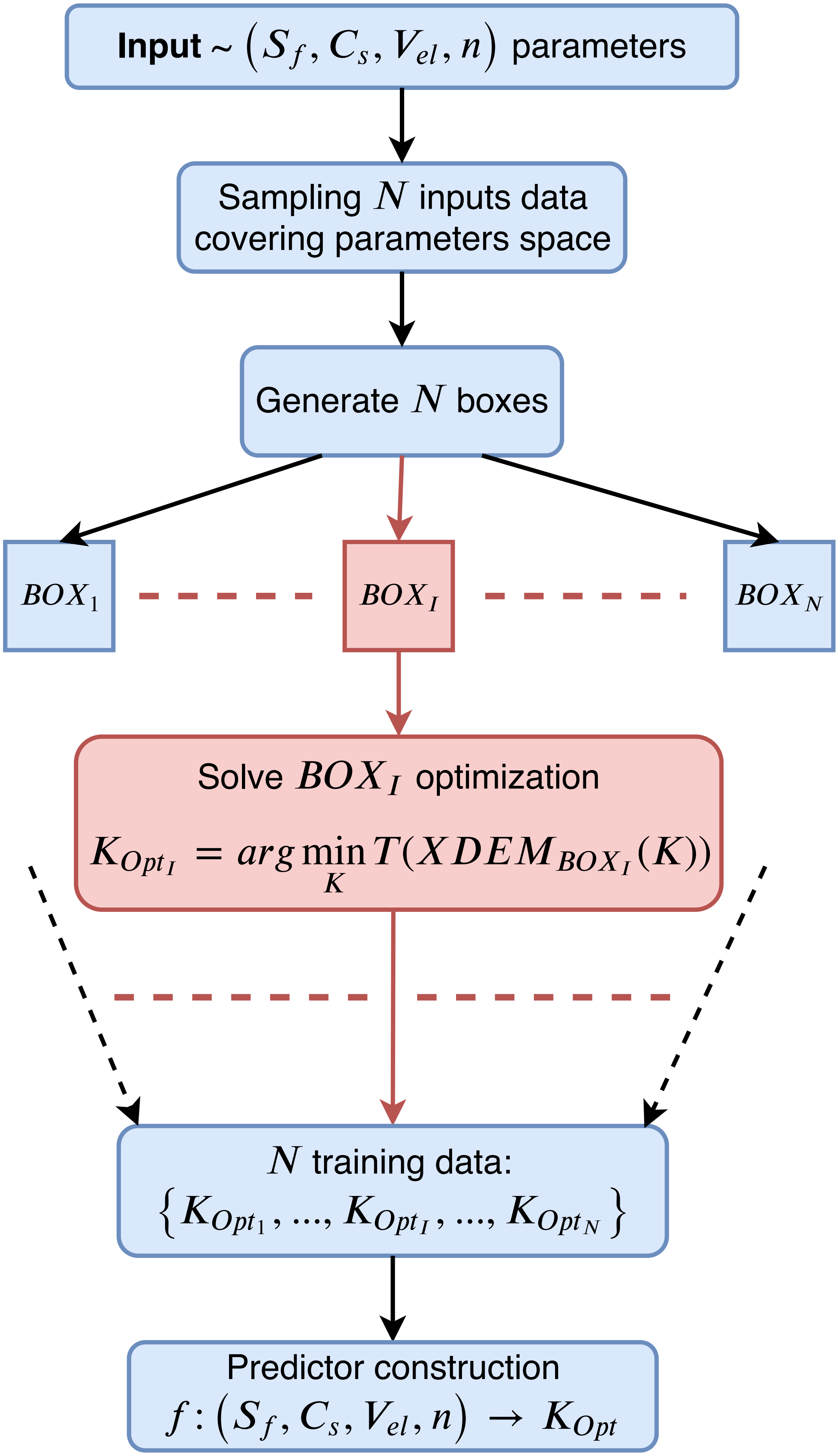

Input ~ (S_f, C_s, V_{el}, n) parameters
Sampling N inputs data covering parameters space
Generate N boxes
BOX_1
BOX_I
BOX_N
Solve BOX_I optimization
$K_{Opt_I} = arg\min_K T(XDEM_{BOX_I}(K))$
N training data:
$\{K_{Opt_1}, ..., K_{Opt_I}, ..., K_{Opt_N}\}$
Predictor construction
$f: (S_f, C_s, V_{el}, n) \rightarrow K_{Opt}$

FIGURE 5.6: Design of experiment steps. S_f: Solid fraction, C_s: Ratio cell to particle size, Vel: Velocity, n: Number of particles, T: XDEM computational time.

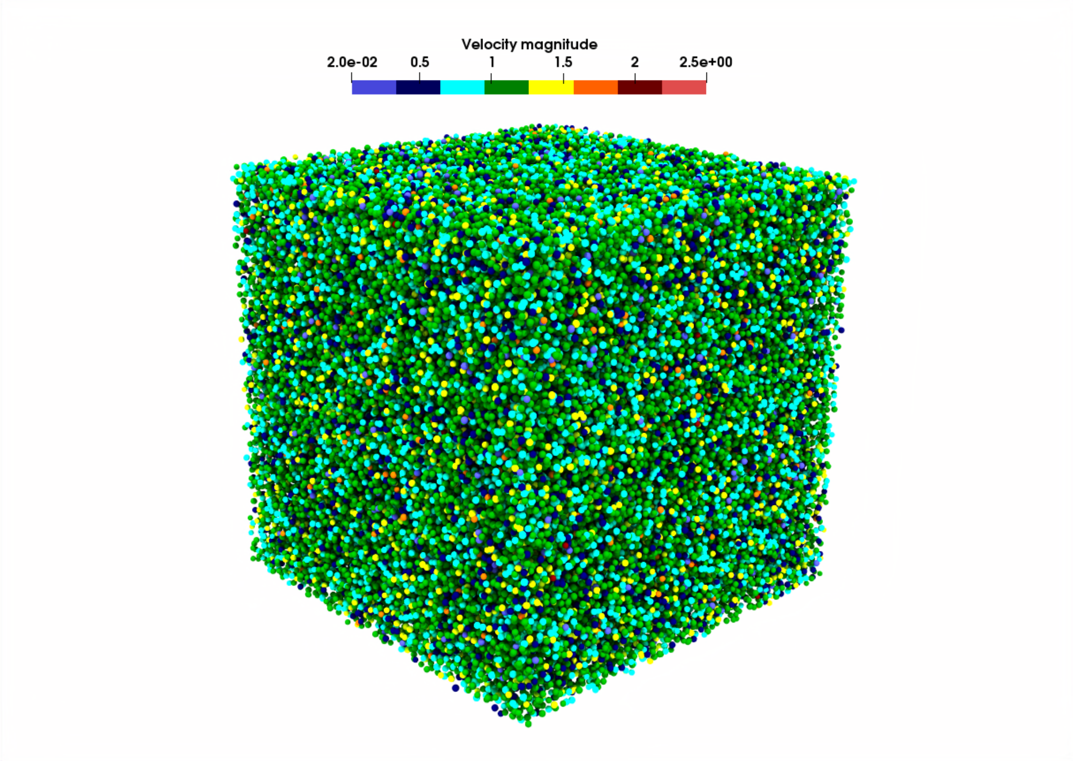


FIGURE 5.7: Figure of a homogeneous particle system for near-optimal skin distance determination.

of different solid fraction, cell size, particles average velocity and number of particles. For each of these boxes, the optimization problem in Eq. 5.3 is solved with the SOGA solver as described in Section 5.4.2.

$$\underset{skin}{\text{minimize}} \quad f(skin) \tag{5.3}$$

With the different near-optimal skin distances obtained from the boxes, a quadratic polynomial model is constructed to express the near-optimal skin distance factor K as a function of the flow conditions. The latter are presented in Tab. 5.2.

All the methodology steps are done in one process with the DAKOTA SOFTWARE as detailed below:

1. First, a Latin Hypercube Sampling (LHS) is used to generate the different points of the flow regime conditions;
2. For each point, the related optimisation problem is solved by:
    - generating the corresponding box;

TABLE 5.2: Simulation designs for near-optimal skin distance.

System particles L/d_p	Solid fraction ϕ	Velocity V_p [m/s]	Ratio cell to particle size $\Delta L/d_p$	skin distance factor K
$5-80$	$[0.1, 0.6]$	$[0.1, 20.0]$	$[1.11, 3.0]$	0-2000

Δt is the DEM time step interval.
L is the box length, d_p, and V_p are the particles' mean size and velocity.
ΔL is the cell size and K the skin distance factor from Eq. 5.2.

  - perform all XDEM evaluations of the box queried by the SOGA solver;
  - returned the near-optimal skin distance factor K;

3. Construct the quadratic polynomial (with zero-order additive correction) model by using the surrogate model capabilities of DAKOTA SOFTWARE;
   (a) the first construction of the models;
   (b) use **root_mean_squared** cross-validation (CV) to refine the models;
   (c) generate new points using LHS again;
   (d) refine the models by using the previous and new points;
   (e) back to 3b until CV convergence;
4. Export the models.

5.5.3 Results

In this section, we present the **surrogate** model expressing the near-optimal skin distance factor K as a function of the flow conditions (Tab. 5.2). The polynomial function is given by the following equation:

$$K_{Opt} = \sum_{k=1}^{n} (c_k \times \prod_{i=1}^{j} (x_i^{p(k,i)})), \tag{5.4}$$

where $n = 15$ and $j = 4$. x_i and c_k are respectively the polynomial variables and coefficients, and $p(k,i)$ the variables degrees.

The variables degrees and coefficients of the polynomial function are presented below:

$$\begin{aligned}
K_{Opt}(S_f, C_s, V_{el}, n) = &\ 4257 \\
&- 5189 \times S_f \\
&+ 3190 \times S_f^2 \\
&+ 1031 \times S_f \times C_s \\
&+ 12.34 \times S_f \times V_{el} \\
&- 2309 \times C_s \\
&+ 377.3 \times C_s^2 \\
&- 14.00 \times C_s \times V_{el} \\
&- 21.27 \times V_{el} \\
&+ 1.580 \times V_{el}^2
\end{aligned} \tag{5.5}$$

The polynomial model has been used to generate response surfaces of the flow conditions. The results are shown in six figures, each presenting two (of the four) parameters at the average value of the other two parameters.

The near-optimal skin distance factor K shows in Fig. 5.8 both solid fraction and ratio cell to particle size as a large effect in the response surface. Lower values of the solid fraction and ratio cell to particle size drive up to higher values of the near-optimal skin distance factor K. One can notice in Fig. 5.10, 5.12, 5.13 that the number of particles has little impact on the near-optimal skin distance factor K. On the other hand, the solid fraction and the ratio cell to particle size are confirmed to have the highest impact.

Tab 5.3 presents a performance comparison between the near-optimal case, considered as the K value returned by the polynomial model, which is specific to each test case, and a default case. The latter is defined as the K value given an excellent performance compromise for any case. The default K value has been established in previous work ("Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method"). In Tab 5.3, the improvement in percentage is defined

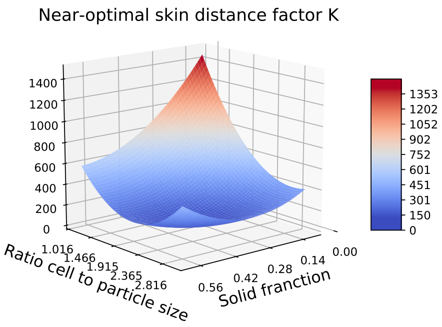


FIGURE 5.8: Response surfaces for the near-optimal skin distance factor K for solid fraction and ratio cell to particle size.

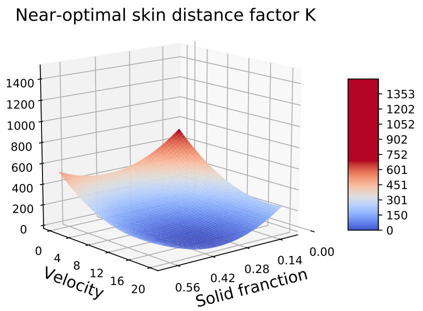


FIGURE 5.9: Response surfaces for the near-optimal skin distance factor K for solid fraction and velocity.

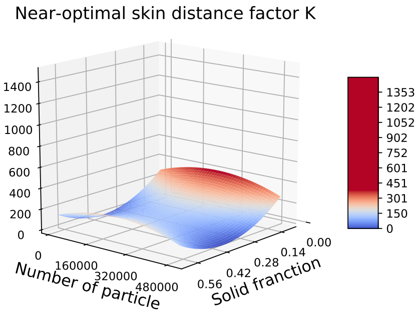


FIGURE 5.10: Response surfaces for the near-optimal skin distance factor K for solid fraction and number of particles.

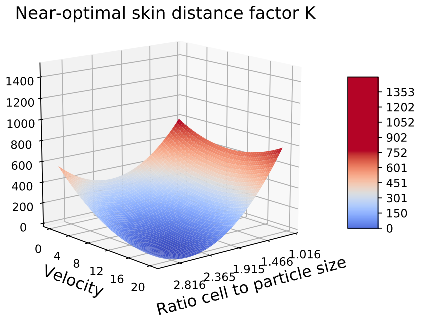


FIGURE 5.11: Response surfaces for the near-optimal skin distance factor K for ratio cell to particle size and velocity.

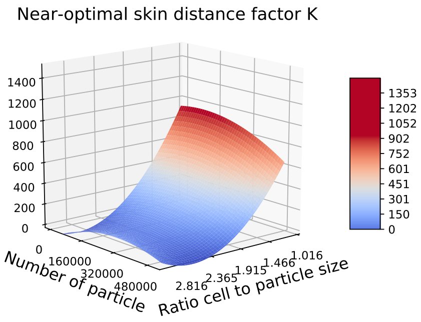


FIGURE 5.12: Response surfaces for the near-optimal skin distance factor K for ratio cell to particle size and number of particles.

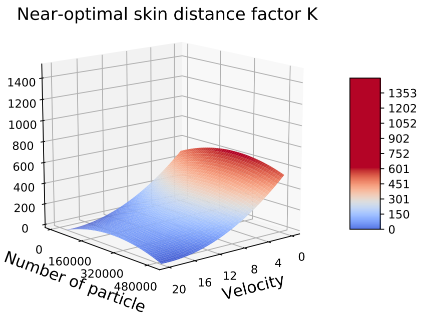


FIGURE 5.13: Response surfaces for the near-optimal skin distance factor K for velocity and number of particles.

TABLE 5.3: Overview of the Verlet buffer method results with the surrogate model over the different test-cases.

Testcase	**Without Verlet**	**Selected Default Value** $K = 200$		**Optimum Value for** $K_{Opt} = \sum_{k=1}^{n}(c_k \times \prod_{i=1}^{j}(x_i^{p(k,i)}))$		
	Simulation time $[s]$	Simulation time $[s]$	Improvement $[\%]$	$K_{Opt}[-]$	Simulation time $[s]$	Improvement compared to default case $[\%]$
Avalanche	3201	581	81.8	392	459	**3.2**
Biomass	492	267	45	481	245	**5.00**
Granular flows	1167	648	44.5	444	342	**26.2**
Hopper 125k	415	295	29	170	284	**2.57**
Hopper 250k	580	432	25.51	210	427	**0.8**
Hopper 500k	954	774	18.87	602	750	**2.51**
Powder leveling	374	187	50	286	164	**6.15**

by the gain made compared to a simulation without using the Verlet buffer method. It is given by the following formula:

$$Improvement = \frac{Time_{w/o} - Time_{case}}{Time_{w/o}} \times 100 \tag{5.6}$$

where $Time_{case}$ is the simulation time depending on which value of K is used for the Verlet buffer method. $Time_{w/o}$ is the simulation time without using the Verlet buffer technique.

The OVERHEAD column corresponds to the time difference (percentage) between the case where the near-optimal $K_{Opt} = \sum_{k=1}^{n}(c_k \times \prod_{i=1}^{j}(x_i^{p(k,i)}))$ value is used and the default case with the acceptable K value. We can notice from table 5.3 that a default $K = 200$, although being an excellent compromise to the near-optimal K_{Opt} value, presents some important overhead in some cases. It has a minimum overhead of 2.51%, which may appear limited but can be very substantial when running simulations for days. The near-optimal K_{Opt} given by the polynomial model provides more than 25% of performance gain in the granular flows. Given the above results, the polynomial model can be used to figure out the near-optimal K value in the Verlet buffer technique.

5.6 Parallel Execution of DAKOTA SOFTWARE on HPC cluster

The University of Luxembourg has a High-Performance Computing infrastructure for research and development. It is used by all the University of Luxembourg researcher community and some partners. The DAKOTA SOFTWARE toolbox has a different level of parallelism in order to take advantage of workstations or HPC resources and are categorized four levels (Adams et al., 2019).

In this study, although DAKOTA SOFTWARE supports *MPI* parallelization, it has been run in sequential with a master process. The latter managed the coarse-grained parallelism (Adams et al., 2019) by starting multiple jobs concurrently using asynchronous job launching techniques. Each of every job launched by DAKOTA SOFTWARE's master runs in parallel, taking advantage of the fine-grained parallelism (Adams et al., 2019) of the function evaluation. The data are then collected in a blocking synchronization manner. All jobs in the queue are completed before exiting the scheduler and returning the results to the algorithm. The job queue fills and then empties, which provides a synchronization point for the algorithm (GAs do not support asynchronous). Fig. 5.14 syntheses and explains the process.

The *Iris* cluster uses the *SLURM* scheduler for managing the workload of the job (Yoo, Jette, and Grondona, 2003). It is configured with a set of **partitions** and **QOS** that enable advanced workflows and accounting. The quality of service (QOS) is related to the partitions as follows: **qos-batch, bigmem, GPU, interactive, long** with one additional QOS called **qos-besteffort**. It is preemptible by all other QoS but has the advantage of not having the limitations imposed on the other QOS, such as the maximum number of nodes, wall-time. **Best-effort** jobs can be set to be automatically re-queued if pre-empted by regular jobs.

To disturb users as little as possible, we configured DAKOTA SOFTWARE iterators to start concurrent jobs using the best-effort QoS. In this way, we only use available resources not used by the other users. When a DAKOTA SOFTWARE evaluation job is running on a resource requested afterward by another user, the concerned resource is freed for the user, and the evaluation is re-queued. For our study that required almost 15000 evaluations, the latter situation happened only 18 times (see Tab 5.4). It means that the other users were hardly affected by our study.

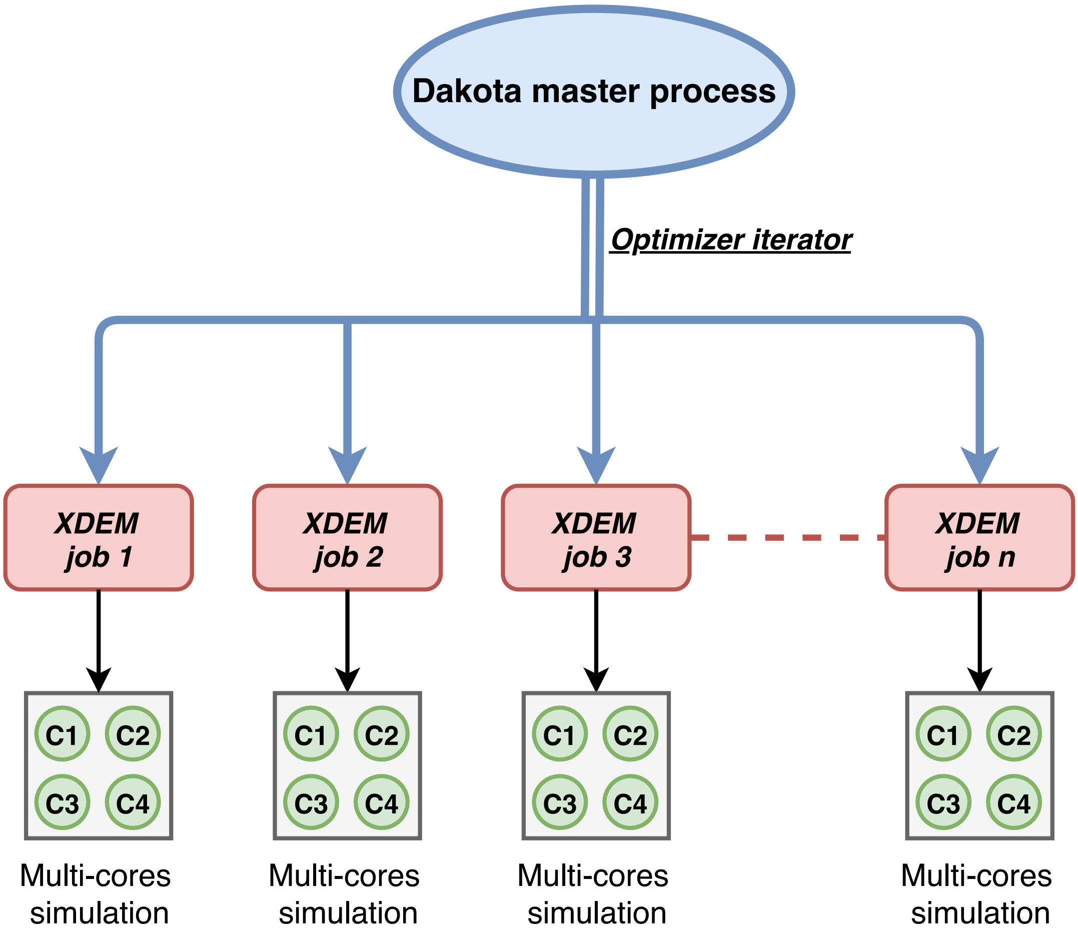


FIGURE 5.14: The DAKOTA SOFTWARE parallelism and scheduling scheme.

Jobs	**Total**	**Percentage**
Completed	146387	99.9877%
Canceled	18	0.0122%

TABLE 5.4: Scheduled job statistics.

5.7 Conclusion

The Verlet buffer technique relies on particles' local flow regime and can be tuned to achieve better efficiency. The latter can be reached by predicting the near-optimal skin distance. In this article, we revealed the existence of such near-optimal skin distance on a life-size case of a biomass furnace. The paper portrays the highlighting of the near-optimal using the Genetic Evolutionary algorithm. The near-optimal skin distance has also been characterized by the solid fraction, the ratio cell to particle size, the velocity, and the number of particles. Those parameters have been used to construct a quadratic polynomial model in order to predict the near-optimal skin distance for any DEM case.

From the polynomial model, we have been able to characterize the near-optimal skin distance. Indeed, the latter is mostly biased by the solid fraction and the cell's ratio to particle size. Lower solid fraction and small cell size lead to a high near-optimal skin distance. Conversely, velocity, especially the number of particles, has a minor impact on the near-optimal skin distance. Density in cells is all(most) what matters.

This model has been implemented in XDEM in order to predict a suitable skin distance for any given input simulation case. The performance evaluation on various real industrial-level test cases shows a reduction of the execution time up to 26%.

Chapter 6

Hybrid MPI+OpenMP Implementation of XDEM

6.1 Abstract

The Extended Discrete Element Method (XDEM) is a novel and innovative numerical simulation technique that extends classical Discrete Element Method (DEM) (which simulates the motion of granular material), by additional properties such as the chemical composition, thermodynamic state, stress/strain for each particle. It has successfully been applied to numerous industries for the processing of granular materials such as sand, rock, wood, or coke (Peters and Pozzetti, 2017; Mahmoudi et al., 2016a).

In this context, computational simulation with (X)DEM has become a more and more essential tool for researchers and scientists to set up and explore their experimental processes. However, increasing the size or the accuracy of a model requires the use of High-Performance Computing (HPC) platforms over a parallelized implementation to accommodate the growing needs in terms of memory and computation time. In practice, such a parallelization is traditionally obtained using either MPI (distributed memory computing), OpenMP (shared memory computing), or hybrid approaches combining both of them.

In this paper, we present the results of our effort to implement an OpenMP version of XDEM allowing hybrid MPI+OpenMP simulations (XDEM being already parallelized with MPI). Far from the basic OpenMP paradigm and recommendations (which consists of decorating the main computation loops with a set of OpenMP pragma), the OpenMP parallelization of XDEM required a fundamental code refactoring and careful tuning to attain acceptable performance. There are two main reasons for those difficulties. Firstly, XDEM is a legacy code developed for more than ten years, initially focused on accuracy rather than performance. Secondly, the particles in a DEM simulation are highly dynamic: they can be added, deleted, and interaction relations can change at any time-step of the simulation. Thus this article details the multiple layers of optimization applied, such as a deep data structure profiling and reorganization, the usage of fast multi-threaded memory allocators and of advanced process/thread-to-core pinning techniques. Experimental results evaluate each optimization's benefit individually and validate the implementation using a real-world application executed on the HPC platform of the University of Luxembourg. Finally, we present our Hybrid MPI+OpenMP results with a 15%-20% performance gain and how it overcomes scalability limits (by increasing the number of computing cores without dropping of performances) of XDEM-based pure MPI simulations [a].

Keyword

DEM, OpenMP, MPI, High Performance Computing(HPC)

6.2 Introduction

Granular materials are widely used in industry and are an active field of research (Duran, 2012). The eXtended DEM is a novel approach that proposes to extend the classical DEM technique by simulating, besides the motion of granular particles, additional properties like thermodynamics state, chemical conversion, magnetic fields or stress/strain (Peters, 2013). A computational simulation like eXtended DEM is becoming increasingly important in numerous research fields.

Recently, thanks to the availability of large scale High-Performance Computing (HPC) platforms, the interest for parallel DEM simulations has grown as it allows us to obtain more accurate and meaningful results previously intractable. The detailed physics models, together with many particles required for realistic DEM simulations, increase the amount of computation and memory consumption automatically. However, nowadays, throughput towards memory is not increasing as quickly as processor computing power, which increases the gap between the memory speed and the cores' theoretical performance, hence the time lost by the processors waiting for the memory (Wulf and McKee, 1995). It brings the question to mind: how do we take advantage of modern and massively parallel machines' increasing power?

---

[a] This chapter is an article published in the IEEE 30nd International Symposium on Computer Architecture and High-Performance Computing

Two major programming models allow to exploit efficiently large scale HPC platforms: Firstly, the distributed memory approach, e.g., Message Passing Interface (MPI) (Clarke, Glendinning, and Hempel, 1994), exploits distributed nodes connected via a high-performance network but requires extra communication even within the same computing node. Secondly, the shared memory approach using OpenMP (Chandra et al., 2001) takes advantage of multi-core nodes and avoid costly communication using a multi-threaded process but is limited to a single node.

The combination of both, specifically hybrid MPI+OpenMP, allows us to overcome these bounds. The hybrid MPI+OpenMP approach brings the following advantages compared to a pure MPI implementation by reducing the number of processes in favor of many threads per process: **Memory savings** for implementations having many replicated data or data structures depending on the number of MPI processes. Furthermore, the number of ghost layers between distributed processes can be reduced. **Better load balance** because the number of partitions to be generated (one per process) is reduced. **Improved scalability** by reducing the number of messages exchanged between processes. **Fit more modern NUMA architectures** that are also hybrid with a distributed memory across nodes and shared within a node (but unique addressing for all NUMA nodes).

In this paper, we present the results of our effort to implement an OpenMP version of XDEM allowing hybrid MPI+OpenMP simulations (XDEM being already parallelized with MPI (Besseron et al., 2013)). Our work goes beyond the basic OpenMP paradigm and recommendations (which summarizes by decorating the main computation loops with a set of OpenMP pragma). Our OpenMP parallelization of XDEM required a fundamental code refactoring and careful tuning to reach acceptable performance. There are two main reasons for those difficulties. Firstly, XDEM is a framework code developed through many years by abounding developers, preferring accuracy over performances. Secondly, the particles in a DEM simulation are highly dynamic: they can be added, deleted, and interaction relations can change at any time-step of the simulation. As a consequence, the contributions of this work are three-fold:

1. the main optimizations applied in order to parallelize our XDEM code efficiently with OpenMP are detailed. Beyond the classic DEM, our approach could be applied to other highly dynamic code;
2. the impact on the performance of each of the proposed optimizations are evaluated independently;
3. finally, the scalability of the proposed new hybrid MPI+OpenMP implementation of XDEM application is assessed.

The remainder of this paper is organized as follows. In the next section, we introduce general background notions on XDEM software and its MPI parallelization. Section 6.3 presents the challenges solved for the OpenMP parallelization of XDEM. Experimental results and performance evaluation are detailed in Section 6.4. Finally, we draw our conclusions in Section 6.5.

6.3 Challenges and implementation for the OpenMP parallelization of XDEM

The OpenMP parallelization layer over an MPI parallelization of XDEM proved to be challenging and required various optimizations and customizations to be efficient. Indeed, the basic OpenMP paradigm which consists of decorating the main computation loops with dedicated *pragmas*, was not applicable within the XDEM software. There are two main reasons for those difficulties. Firstly, XDEM is a code developed within the dynamic LUXDEM group for years, used to make very accurate simulations at the expense of performance. Secondly, the particles in a DEM simulation are highly dynamic: they can move from one process to another, which means that they are deleted on one process and created on the other. Also, interaction relations can change at any timestep of the simulation. It follows that to achieve the significant performance improvements reported in this article, a fundamental code re-factoring has been necessary. The steps taken to reach an effective OpenMP parallelization are now detailed.

6.3.1 Data structures and concurrent accesses

One of the biggest challenges that affect the performance of a program is the choice of data structures. For a C++ application like XDEM, the Standard Template Library (STL) (Stepanov and Lee, 1995) offers a wide range of efficient data structures. For an OpenMP program, the best performance is traditionally reached using containers offering random access and contiguous memory, like C arrays or C++ `std::vector` or `std::array`. However, because XDEM code is dynamic (the number of particles can change at every timestep, the number of collisions are hard to predict) fixed-size structure are not well-suited[b]. A careful analysis of the used structures within XDEM is, therefore, necessary to select the most appropriate approach, depending on the potential for parallelization. For this reason, Fig. 6.1 lists the main phases that constitute the XDEM time loop, where the colors indicate on which element a given phase operates.

Prediction and **Integration** phases loop through the particles. In the sequential implementation, the particles were initially stored in a `std::map` using the particle *ID* as the key. It allowed operations like fast search (necessary when receiving particle state update from remote processes), deletion (when particles leave the domain), and insertion (when new particles are created or move from another process). However, the biggest drawback of the `map` is that it does not provide any random access iterators, which prevents any efficient OpenMP parallelization. Additionally, the `map` elements are not stored contiguously in memory (but in a tree instead), which significantly slow down the memory accesses.

As a replacement, we decided to use the `flat_map` data structure provided by the Boost library (Schäling, 2011). Indeed, `boost::flat_map` presents the same functionality as the STL `map` but relies on a vector to store its elements and sorts them according to the key. A direct benefit of such an approach is the availability of a random access iterator as well as a contiguous memory storage. However the insertion of new elements can be costly as it may require a re-allocation of the array and a copy of its content. This drawback can be circumvented by reserving

---

[b] For example, adding a new element in a vector requires to re-allocate memory and copy the whole content of the vector to the newly allocated space.

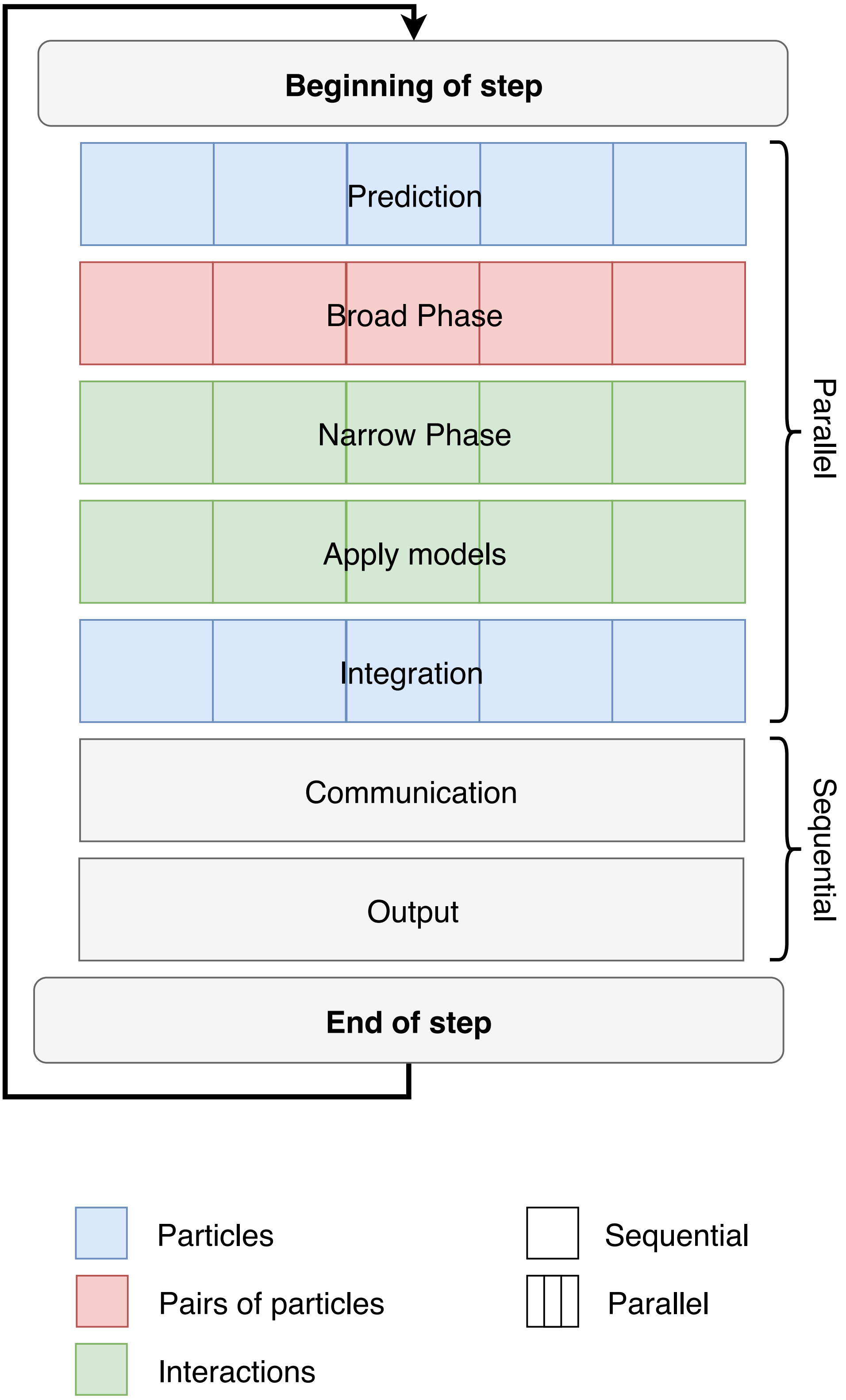


FIGURE 6.1: The different phases of an XDEM iteration loop.

additional space to amortize the overhead induced by insertion operations upon their occurrence.

As regards to the **Broad-Phase**, this stage operates on pairs of particles[c] and generates a list of particle pairs that can *potentially* interact. While the Broad-Phase would also benefits from `flat_map` when iterating on the particles, such a data structure would not be adapted. Indeed, the result of the Broad-Phase is a list of particle pairs whose size is not known in advance – the number of contacts can vary significantly from zero to many times the number of particles depending on the packing level of the particles. Additionally, in an OpenMP parallelization, the elements added to this list will be generated from multiple threads. The sequential implementation of XDEM was naturally using the `std::list` data structure in which new element can be added in constant time. And for the OpenMP implementation of XDEM, we decided to use an STL `deque` structure to accumulate the list of particle pairs. A `deque` can be considered as a list of vectors of constant size. Insertion at the end is done in constant time (even when a new block is allocated because there is no need to copy data from the previous blocks).

There is still the problem of concurrent accesses from the different threads to add new pairs to this list. OpenMP proposes different mechanisms, like *atomic* or *critical* regions and *Reduction* clause (since OpenMP 3.0) to handle safely this type of concurrent operations. However, they represent a major performance bottleneck. As detailed in the Listing 6.1, our solution to this problem is to accumulate the results of each OpenMP thread in their private `deque`. In a second step, each thread's results are copied into a single result `std::vector`. The access range of the result `vector` is calculated for each thread using the simple prefix calculation on the `deque` size of each thread. In practice, this approach appeared to be much more efficient than a *critical* region (in particular with large number of threads) or a *Reduction* clause that does a merge operation for each element of the loop rather than for each thread.

The **Narrow-Phase** and **Apply Physics Model** phase operate on the list of interactions (or pair of particles) generated during the Broad-Phase. Thanks to the

---

[c] Of course, efficient broad-phase algorithms do not consider *all* pairs (Rousset et al., 2017).

```
std::vector<Particle_Pair> interactions;
std::vector<int> sizes;

#pragma omp parallel
{
 std::deque<Particle_Pair> private_deque;

 int i_thread   = omp_get_thread_num();
 int nb_threads = omp_get_max_threads();

 #pragma omp single
 {
  sizes.resize( nb_threads + 1 );
  sizes[0] = 0;
 }

 #pragma omp for
 // Parallel broad-phase algorithm:
 // - Check particle pairs for interaction
 // - If interacting, add in the private deque
 for(...)
 {
  if(interacting(p1,p2))
  {
   private_deque.push_back(Particle_Pair(p1,p2));
  }
 }

 sizes[i_thread+1] = private_deque.size();

 #pragma omp barrier
 #pragma omp single
 {
  std::partial_sum(sizes.begin(), sizes.end());
  interactions.resize(sizes[nb_threads]);
 }

 std::copy(private_deque.begin(), private_deque.end(), interactions.begin() +
     sizes[i_thread]);
}
```

LISTING 6.1: Algorithm to accumulate the list of interactions from different OpenMP threads

optimizations applied to the Broad-Phase, the list of interactions is now stored in an STL `vector`, which naturally provides random access to its elements.

As a summary, the main optimizations of XDEM data structures are: (1) the STL `map` of particles have been changed to a Boost `flat_map`; (2) the STL `list` of the interactions generated during the Broad-Phase is now performed using a private STL `deque` for each thread, a prefix calculation on the sizes and then a copy in an STL `vector`.

Those changes allow to benefit from random data access for the OpenMP parallelization and to avoid concurrent accesses to shared containers. Finally, Tab. 6.1 summarizes the characteristics of the different data structures used in XDEM before

---

[c] Insertion and removal in a `flat_map` have a logarithmic search time, plus linear with the number of elements bigger than key.

and after our optimizations (applied for the OpenMP parallelization). We only consider the operations that are used in the XDEM. The complexity values are taken directly from the official STL and Boost documentation.

6.3.2 Memory allocation

As detailed before, XDEM is a highly dynamic C++ code. It implies the creation and deletion of objects all along of a simulation. This results in an intensive call of the memory allocator (*e.g.* `malloc()` and `free()`), many of them happening within the OpenMP parallel regions. Standard system memory allocator like GNU C library, or **glibc**, use mutexes to prevent concurrent access to allocator structures and preserve their consistency. In multi-threaded applications such as XDEM, different threads concurrently invoke memory allocator, and as a result, we have a large number of lock conflicts. So most of the time is spent in locking/unlocking mutexes even if threads are working autonomously (thread is accessing objects created only by itself). This result in some critical contentions which limited the scalability of our implementation. The obvious solution is to avoid the creation of objects in dynamic memory and allocate objects on the stack instead, but it is not always possible or convenient. To workaround this issue, we have used alternative memory allocators: **jemalloc** (Evans, 2006) and **TCMalloc** (Ghemawat and Menage, 2009) which are designed to support highly multi-threaded workflow. Those optimized memory allocators use multiple independent *arenas* (for jemalloc) or thread caches (for TCMalloc) to reduce contention in a multithread application. They can be used by explicitly linking the executable to the memory library or by merely setting the `LD_PRELOAD` environment variable.

TABLE 6.1: Characteristics of the containers used in XDEM before and after our optimizations. Only the operations used in XDEM are considered.

Container	**Insert**	**Erase**	**Find**	**Random Access**	**Memory Contiguity**
`map`	$O(\log n)$	$O(\log n)$	$O(\log n)$	No	No
`flat_map`	$O(n)^{c}$	$O(n)^{c}$	$O(\log n)$	Yes	Yes

Container	**Push Back**	**Random Access**	**Memory Contiguity**
`list`	$O(1)$	No	No
`deque`	$O(1)$	Yes	Partially
`vector`	Not used	Yes	Yes

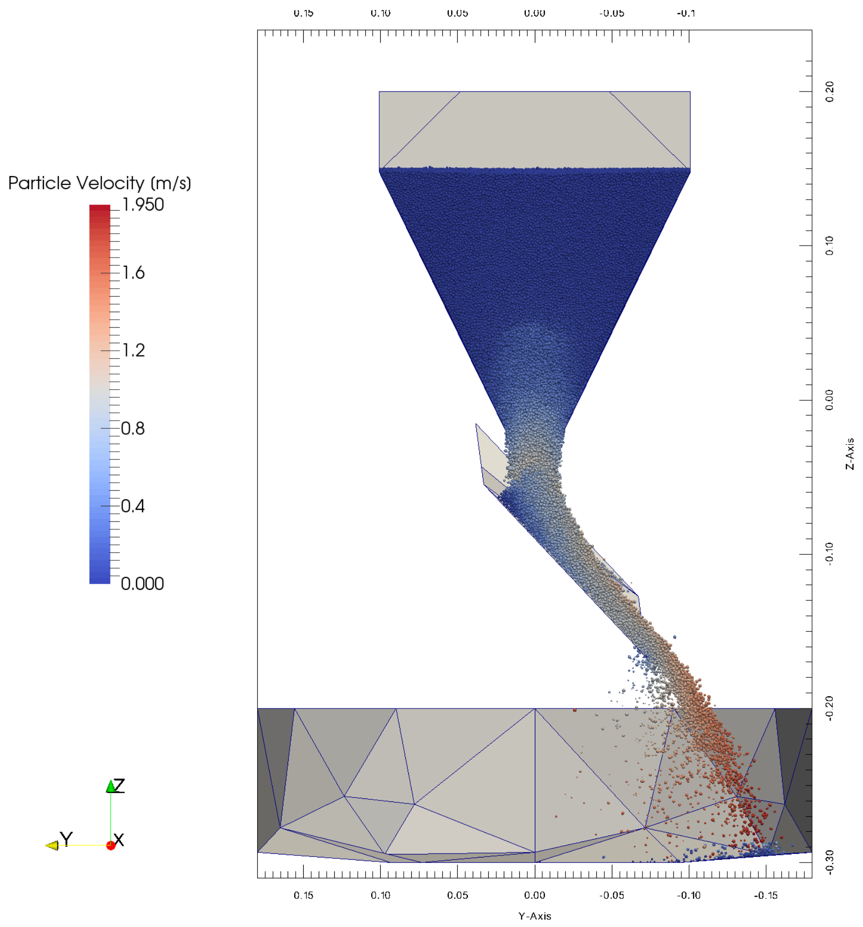


FIGURE 6.2: The test case used for the performance evaluation simulates the hopper discharge of 1 million particles. The left part shows an overview of the setup with the particles colored according to their size. The right side displays the middle slice allowing us to see the particle velocity distribution during the discharge process.

6.4 Experimental Results

In order to assess the validity of our approach and evaluate the scalability of the proposed strategies, we have set up and executed the **Hopper Discharge** test case described in the section 6.4.1 below and illustrated in Fig. 6.2.

6.4.1 Case for experimental evaluation

To investigate the performance and the behavior of the OpenMP and Hybrid implementation, the **Hopper Discharge** test case is used with 250K and 1M particles. The test case has been introduced in section 2.2.3.

6.4.2 Experimental settings

The experiments were carried out using the `Iris` cluster of the University of Luxembourg (Varrette et al., 2014) which provides 168 computing nodes for a total of 4704 cores. The nodes used in this study feature a total a 128 GB of memory and have two Intel Xeon E5-2680 v4 processors running at 2.4 GHz, that is to say, a total of 28 cores per node. The nodes are connected through a fast, low-latency EDR InfiniBand (100Gb/s) network organized over a fat-tree topology. We used XDEM version `b6e12a86`, compiled with GCC Compiler. Parallel executions were performed using OpenMPI over the InfiniBand network. The nodes were reserved for exclusive access to ensure the stability of the measurement. Additionally, each performance value reported in this section is the average of at least a hundred measurements. The standard deviation showed no significant variation in the results.

6.4.3 Impact of data structures

In this section, we highlight the impact of the data structure optimizations applied to the XDEM code. We made a comparison between our original code and the new OpenMP version on sequential execution.

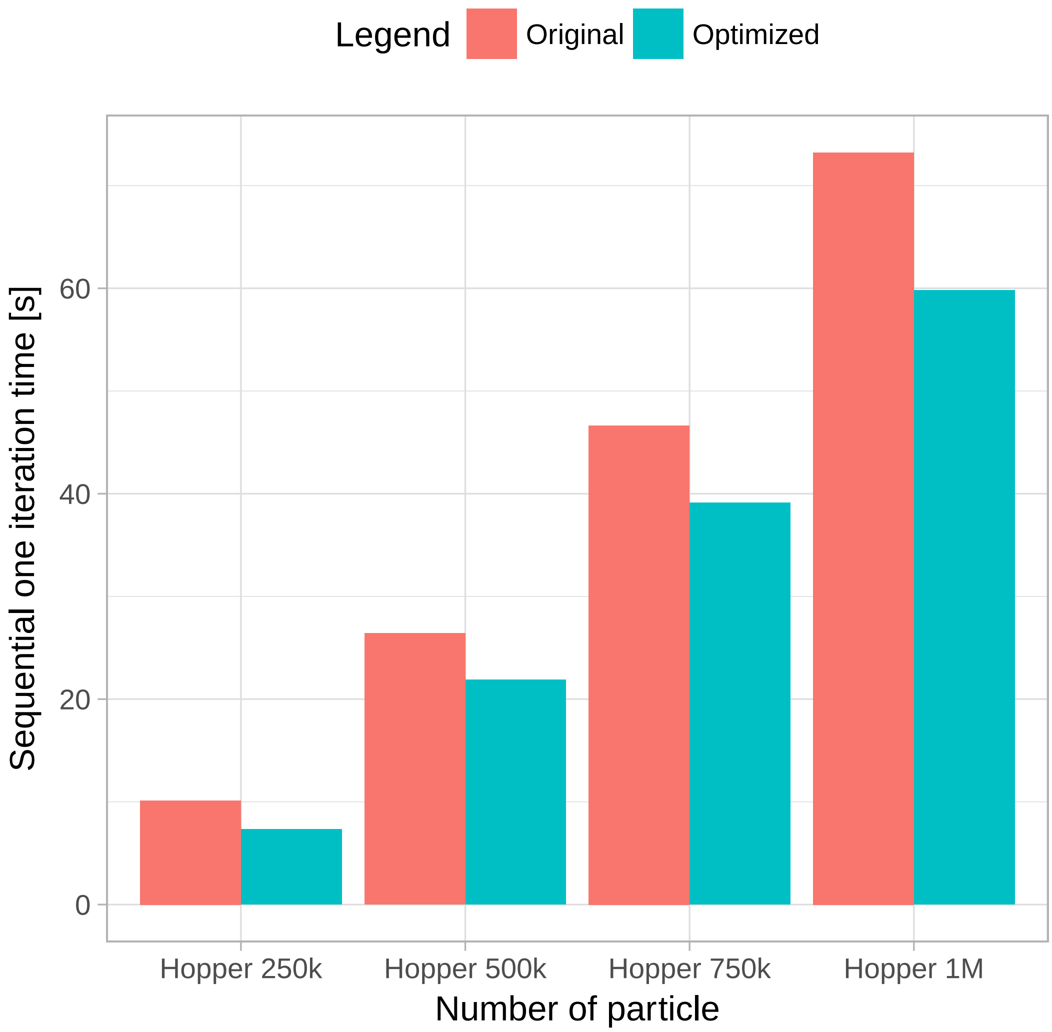


FIGURE 6.3: Impact of data structure in sequential for Hopper test (from 250k left to 1M right).

Within this context, Fig. 6.3 compares the iteration time with the two versions of XDEM for the hopper discharge test case with a different number of particles. We can see the clear benefit of those optimizations, even in sequential execution, reducing the execution time by 16% to 26% depending on the size of the case.

6.4.4 Impact of memory allocator

In this section, we quickly investigate the impact of the memory allocator on the code's performances. We compare the default memory allocator **glibc** to **TCMalloc** and **jemalloc**.

Fig. 6.4 is a plot comparing the main loop time with **glibc**, **TCMalloc** and **jemalloc** memory allocators previously introduce in Section 6.4.1 for the hopper discharge test case with 250k particles. As expected, jemalloc and TCMalloc offer better performances in multi-threaded programs, and the **glibc** performances drop as the number of thread increases. Nevertheless, we see that **jemalloc** is slightly more efficient than **TCMalloc**. In multi-threaded programs, the heap is a bottleneck that makes the program not scalable. When multiple threads simultaneously allocate or deallocate memory from the allocator, the operation is "serialized" by the **glibc** allocator. *jemalloc* and *TCMalloc* allow to reduce the contention for memory operations by using independent *arenas* (Evans, 2006) or *thread caches* (Ghemawat and Menage, 2009). XDEM code makes intensive use of the allocator (manual allocations and calls to the C++ STL) that slow down the program as the number of threads increases. jemalloc and TCMalloc eliminate this bottleneck by emphasizing fragmentation avoidance and scalable concurrency.

6.4.5 OpenMP and MPI scalability

We have measured the execution time of the main loop of XDEM for the Hopper Discharge case with 250K particles for our pure MPI and pure OpenMP versions on one node, varying the number of cores from 1 to 28. Fig. 6.5 shows the speedup comparison of the two approaches with the number of cores on the x-axis and the speedup on the y-axis.

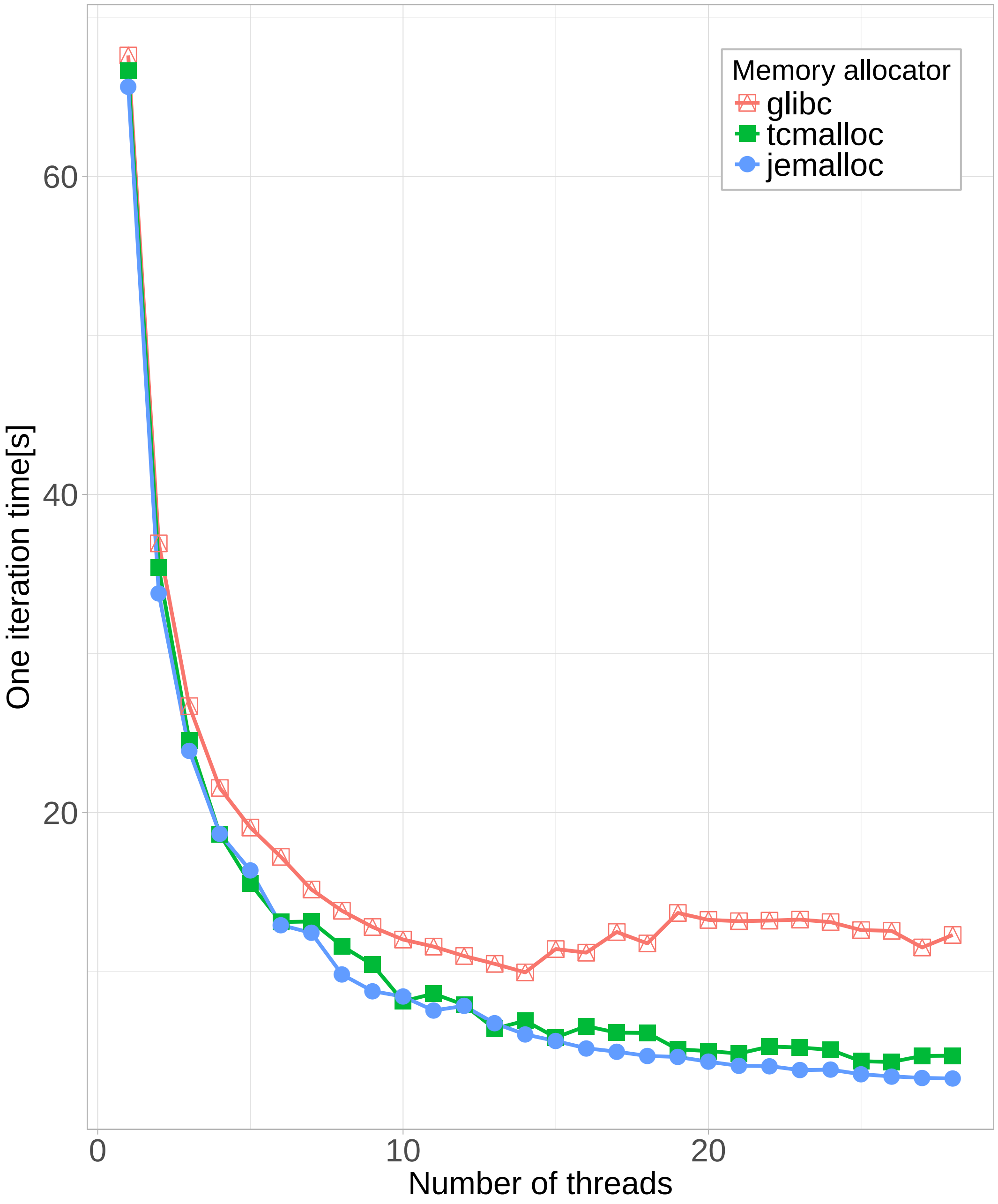


FIGURE 6.4: Impact of different memory allocators on one iteration time with OpenMP version on one node (from 1 to 28 cores) for Hopper test case (1M particles).

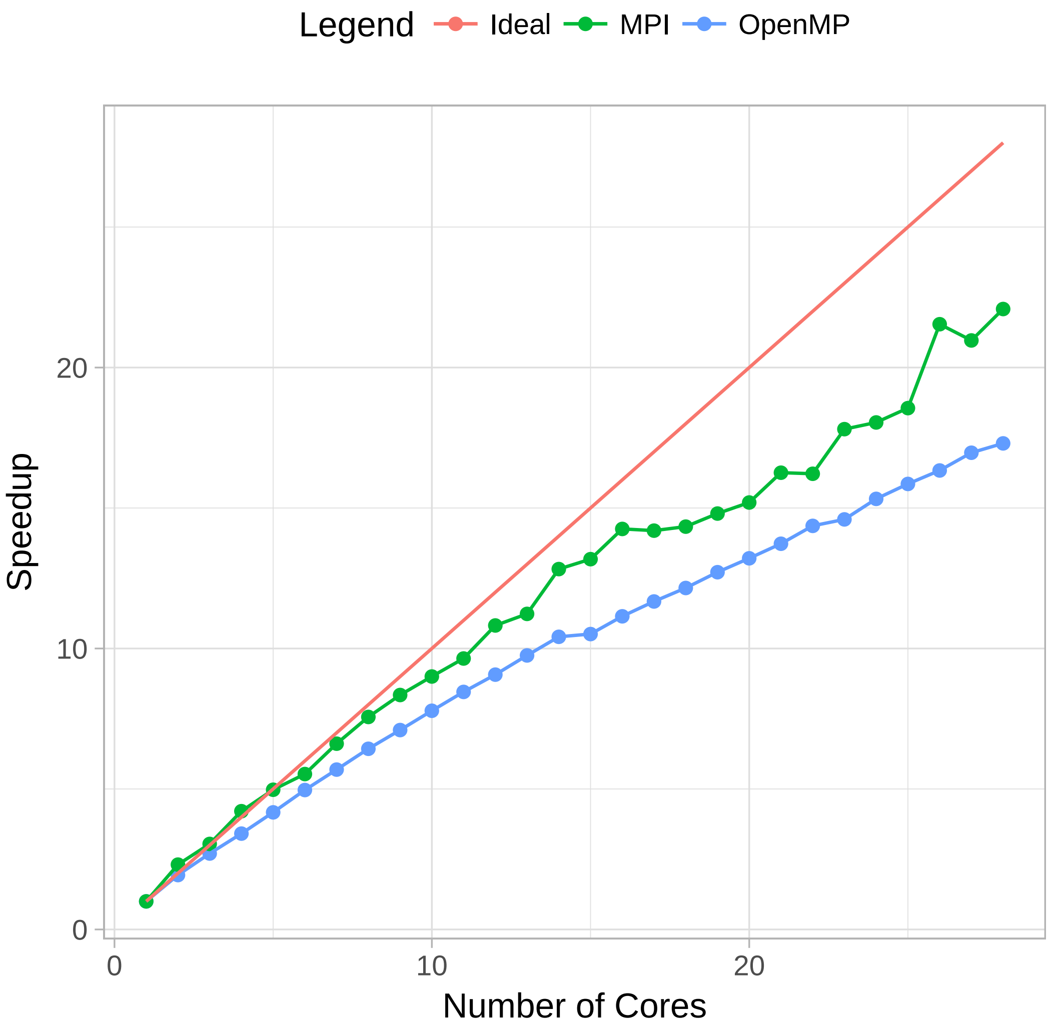


FIGURE 6.5: SpeedUp of MPI and OpenMP versions on one node *i.e.* up to 28 cores.

The pure MPI version with a speedup of 23 (82% efficiency on 28 cores) scales better than the OpenMP version with a speedup of 18 (64% efficiency on 28 cores). We would expect the pure OpenMP version to perform better than the pure MPI due to the overhead of the MPI communication but it is not the case. One possible explanation is that, as shown on Fig. 6.1, the XDEM workflow is composed of different phases which corresponds to different parallel OpenMP sections and therefore represents implicit barriers. On the other end, the MPI version only uses barriers during the communication phase.

6.4.6 Hybrid execution

To analyze the hybrid performance, we measured the main loop's average time per process with different schemes within our SMP nodes. The goal is to compare different hybrid configurations with the same number of cores. Our cluster nodes are composed of two processors, one per socket, and each socket has fourteen cores. Taking into account this configuration, we have tested various hybrid MPI+OpenMP strategies per node:

- 28 OpenMP threads per node, full OpenMP threads;
- 1 MPI processes per socket and 14 threads per MPI;
- 2 MPI processes per socket and 7 OpenMP per MPI;
- 7 MPI processes per socket and 2 OpenMP per MPI;
- 14 MPI processes per socket, full MPI processes;

The speedup tests were performed on 18 nodes *i.e.* 504 cores with the Hopper Discharge one million case. The first remark is the OpenMP version under-performing compared to the hybrid and full MPI version. The main reason comes from insufficient workload as the number of cores increases. From 1 to 12 nodes, the full MPI speedup better than the different hybrid strategies but from 12 (336 cores) to 18 nodes (504 cores) the hybrid "7 MPI processes per socket and 2 OpenMP per MPI" speedup better than the full MPI. It is mainly due to the MPI communications overhead when the

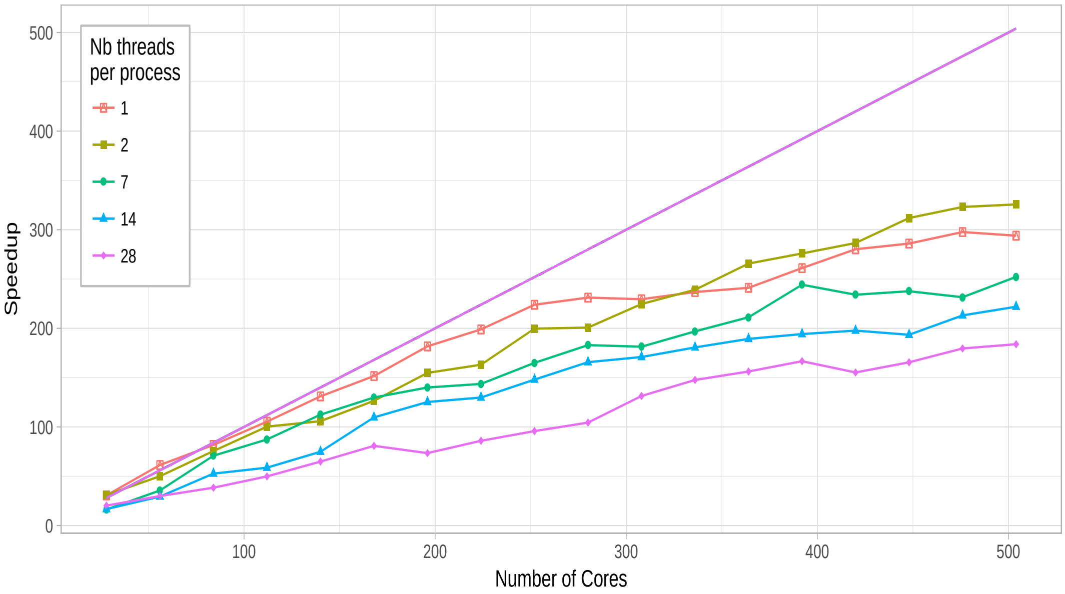


FIGURE 6.6: Speedup of hybrid MPI+OpenMP executions for different number of threads per process on the Hopper Discharge case with 1 million particles.

number of processes is increasing. The primordial advantage over a hybrid code is to overcome full MPI bottleneck at an equal number of computing cores.

6.5 Conlusion

The Extended DEM is a C++ legacy code parallelized with MPI and developed for more than ten years by many researchers contributing to several distinct feature of the code. In this paper, we present the results of our effort to implement a complementary OpenMP version of XDEM allowing for hybrid MPI+OpenMP simulations. In particular, from the deep data structure profiling, a non-trivial code reorganization has been performed, which includes several drastic changes in the used data structures, as well as the selection of optimized fast multi-threaded memory allocators. Our design choices are performance-oriented, and the experimental results obtained on a real-world application validate the implementation changes and permitted to comfort the proposed approach. More precisely, when comparing the performances of our full MPI, full OpenMP, and different hybrid strategies in a HPC context, *i.e.* up to 504 computing cores (18 nodes) of the HPC facility of the University of Luxembourg, we demonstrate the relevance of the hybrid version when increasing the computing cores with a 15% $-$ 20% performance gain. These open novel perspectives for the efficient parallelization of the XDEM software. In particular, the future work induced by this study includes the addition of a GPU layer within XDEM, and a detailed cache optimization analysis to mainly improve sequential runs for bulky cases.

Part II

Performance analysis and Application

Chapter 7

PERFORMANCE ANALYSIS

7.1 Introduction

In chapter 2, we introduced some standard methods to evaluate a computer program performance. We have detected several hot-spots that have been addressed and other aspects as the memory footprint that we ensured stay reasonable during our developments. From chapters 3 to 6, we presented the major optimizations introduced in our XDEM software to improve it. This chapter will therefore analyze the global impacts of all those optimizations by setting-up a macro benchmarking (using the *Dam break* test case) approach and comparing the latest version (10/2020, git hash: *11ee77d0093b8409f1c6c8ec7dec749775c0da34*) to the old version (03/2017, git hash: *18a22cbfdadf7fe2afd8bbc9ba02744d75b775d0*) of XDEM using the *Biomass* baseline test case. The goal of the current chapter is therefore to firstly compare the performance of the old and latest versions of XDEM by highlighting where the gains have been made in sequential and parallel runs. The behavior of our XDEM software in a coupled simulation with OPENFOAM is also studied at a large and parallel scale environment.

Strong scaling and speed up have been performed using the execution time as a performance metric.

7.2 New vs old XDEM versions performance

In this section, we compare the performance of the 03/2017 and 10/2020 XDEM versions using the baseline biomass test case defined in section 7.2.1. The main purpose is to assess and highlight the performances brought by all the efforts made during this doctoral research. As for our previous performance assessments, the simulation time has been chosen as a performance metric as it is the main performance criteria for XDEM users. Only the simulation time of the *dynamic* and *conversion* parts of XDEM are presented and studied in this comparison.

7.2.1 Biomass furnace combustion

The DEM test case presented in the section is an application case used throughout the research to assess our optimization developments' performance. The test case simulates a combustion chamber's behavior of a 16 MW geothermal steam superheater, which is part of the Enel Green Power "Cornia 2" power plant and includes both the moving wooden bed and the combustion chamber above it. In this test case, the XDEM simulation platform is based on a hybrid four-way coupling between the Discrete Element Method (DEM) and Computational Fluid Dynamics (CFD). In this approach, particles are treated as discrete elements coupled by heat, mass, and momentum transfer to the surrounding gas as a continuous phase. Besides the equations of motion for individual wood particles, the differential conservation equations for mass, heat, and momentum, which describe the thermodynamic state during thermal conversion, are solved. This test case aims to propose a numerical approach that can combine computationally low-cost simulations and practical use of the design for industrial applications with sufficient accuracy of the results.

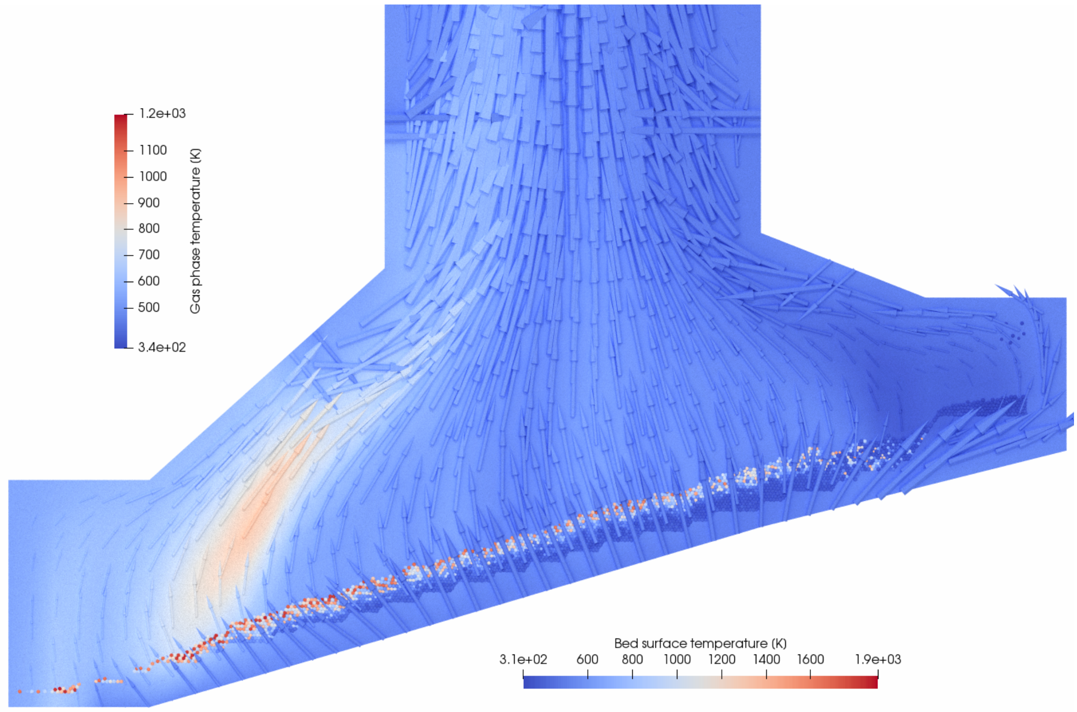


FIGURE 7.1: The Enel Green Power "Cornia 2" biomass combustion power plant.

The case has 2224 particles arranged on moving and fixed grates. The grate has three different moving sections to ensure adequate mixing of the biomass parts and an appropriate residence time. The primary air (PA) enters from below the grate in the combustion chamber. Those grates are split into four different zones (sections). Furthermore, a secondary air (SA) is injected at high velocity straight over the fuel bed through two circular nozzles. A Flue Gas Re-circulation (FGR) is partly injected through two jets along the vertical channel and partly from below the grate (see Fig. 7.1). The biomass furnace's geometric data and operating conditions can be found in the Master thesis (LUPI, 2017a) and chapter 8.

7.2.2 Sequential performance

Fig. 7.2 compares the sequential simulation time of the 03/2017 and 10/2020 versions of XDEM using the baseline biomass test case with *Dynamic* and *Conversion* modules. The performance evaluation does not include OPENFOAM as it was disabled. The plot is a stacked barplot of the different XDEM main parts for the old and new versions. We can notice a clear gain in the performance as the actual 10/2020 version is 13× times faster than the 03/2017 version, and they respectively have 1567s and 120.53s simulation times.

The speed up from the old to new version of the code is perceptible in all the main parts of the simulation, as shown in Tab. 7.1.

XDEM Parts	03/2017 [s]	10/2020 [s]	Speed up (between the two versions)
Broad phase	235	6	40
Narrow phase	438	30	14.60
Dynamic models	470	36	13
Integration	235	18	13.05
Conversion	16	1.20	13.33
Others	31.34	18	1.72

TABLE 7.1: Table to compare XDEM versions simulation times in sequential.

The gains shown in Fig. 7.2 and Tab. 7.1 demonstrated that the multiple optimizations and improvements also benefit the sequential runs. The significant changes introduced in data structures, algorithms and many small improvements have proved their benefits in the sequential runs. It is an essential performance gain for XDEM

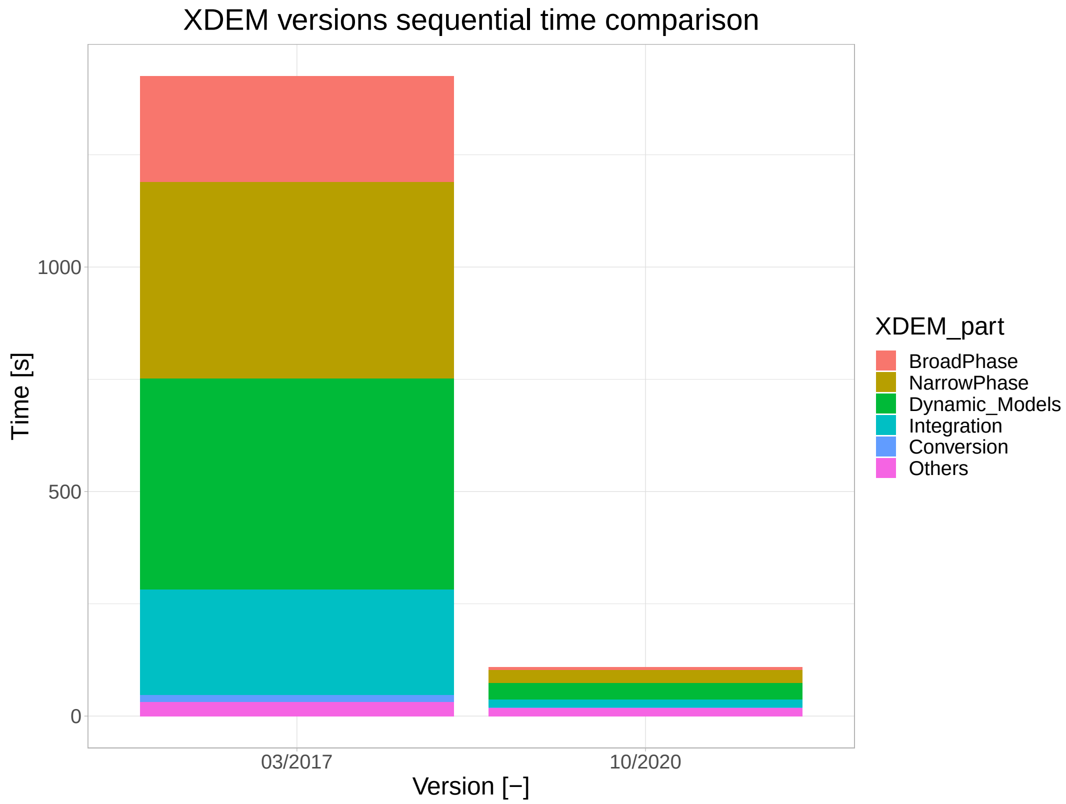


FIGURE 7.2: Sequential simulation time comparison between 03/2017 and 10/2020 XDEM versions.

users as the sequential runs are often used to set-up a case before production runs, which are usually performed in parallel.

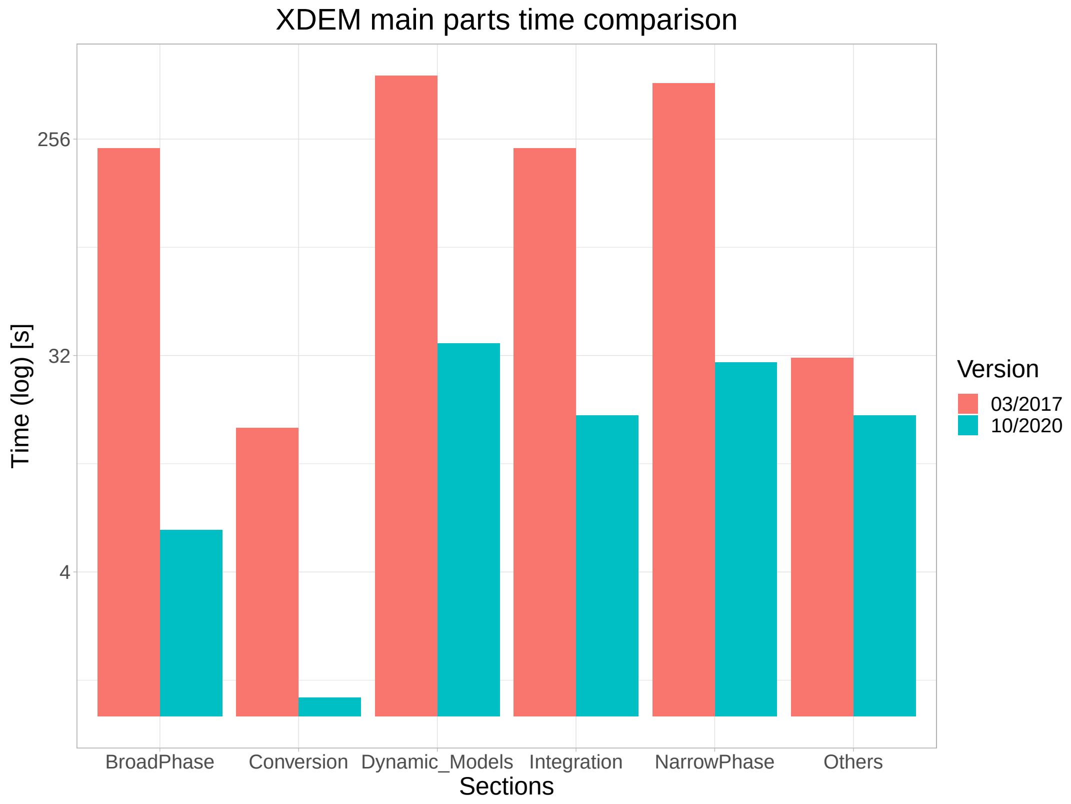


FIGURE 7.3: XDEM main parts simulation time comparison between 03/2017 and 10/2020 XDEM versions. The y-axis is represented in log time.

In Fig. 7.3, we present the simulation times of the main parts of the XDEM software: broad and narrow-phases, dynamic integration and contact models, and conversion (chemical reactions). It compares the 03/2017 and 10/2020 versions. It comes out that we have made a big step forward in the collision detection processes as the most gains were made in the broad/narrow phases and dynamic models (see also Tab. 7.1). The broad-phase simulation time is reduced from 235 to 6 seconds, illustrating a huge speed up of 40 between the old and new versions. On the other hand, the narrow-phase and dynamic models' simulation times are down from 438 to 30 seconds and from 235 to 18, respectively, showing speed ups of 15 and 13. This comes as a non surprise as the chapters 3, 4, and 5 focused on the collision detection processes optimization. Indeed, the sequential runs' significant gains come from optimizing the algorithms, implementations, and data structures. Important performance gains were also made in the *Integration* and *Conversion* parts (speed up of 13 for both).

7.2.3 Parallel performance

This section compares the parallel runs simulation time of the 03/2017 and 10/2020 XDEM versions with *Dynamic* and *Conversion* modules. The performance evaluation does not include OPENFOAM as it was disabled. Fig. 7.4 shows the primary loop simulation time over the number of cores of the old and new XDEM versions. We can observe that the 10/2020 version is much faster (11 times on 28 cores) than the 03/2017 version in parallel as it was in sequential runs. There is a speed up of 13 in sequential and 11 in parallel using 28 cores between the two versions.

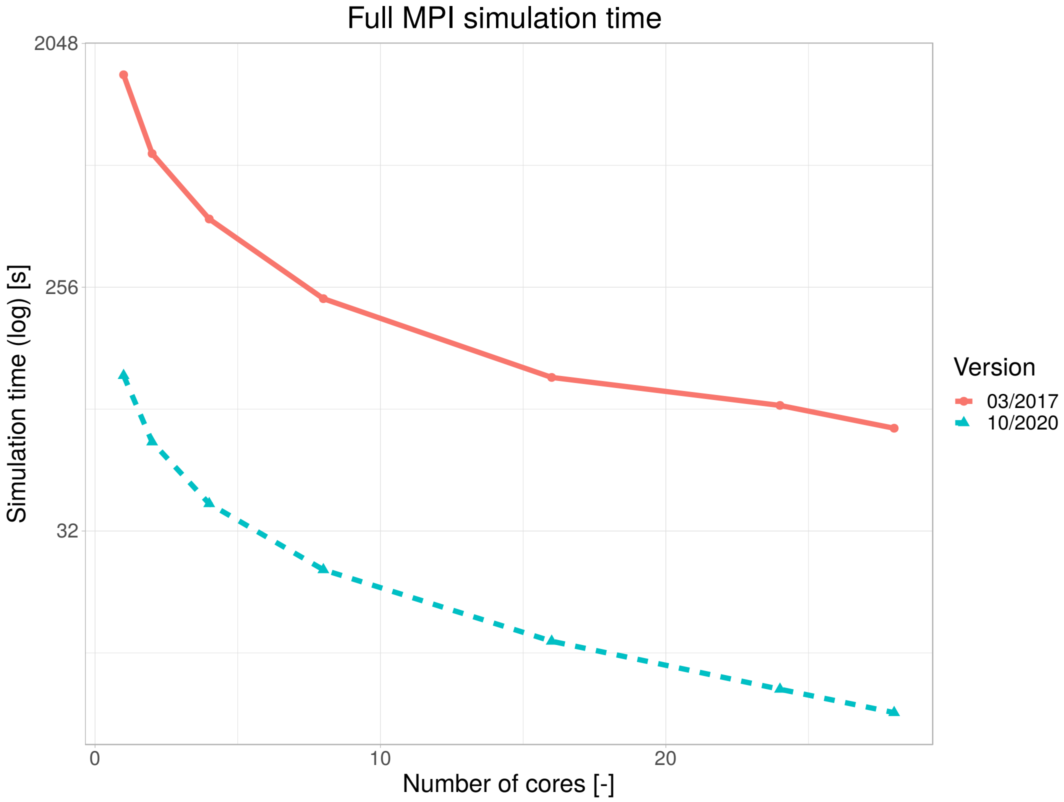


FIGURE 7.4: Strong scaling of the main loop's simulation time. The number of MPI processes is on the x-axis, and the log simulation time on the y-axis. The execution was made on one node with 28 cores.

Tab. 7.2 compares the main loop simulation times of the two XDEM versions in Full parallel MPI. The number of cores is also the number of MPI processes. On average, in parallel, the 10/2020 version is 11 times faster than the 03/2017 version (13 times in sequential).

Number of cores	03/2017 [s]	10/2020 [s]	Speed up [-] (between the two versions)
1	1567	120.53	13
2	800.10	68.40	11.70
4	458.10	40.40	11.34
8	232.20	23	10.10
16	118.60	12.50	9.50
24	93.40	8.30	11.25
28	76.90	6.80	11.30

TABLE 7.2: Table to compare XDEM versions simulation times in parallel, full MPI configuration.

In full MPI configuration, the 10/2020 version's strong scalability is better than the 10/2020's version highlighting the performance gains that have also been made through MPI optimizations. However, the 10/2020 XDEM version has OpenMP parallel capabilities and offers different hybrid MPI/OpenMP configurations. In Fig. 7.5, we compare the strong scalability of three different hybrids MPI/OpenMP parallel configurations: *full OpenMP, 2 MPI processes per node and 14 OpenMP threads per MPI process, and 14 MPI processes per node and 2 OpenMP threads per MPI process.*

We can notice that the three strategies are very close in simulation time, but the full OpenMP emerges as the fastest strategy. All hybrid configurations are better than the full MPI configuration. Hybrid configurations have less MPI process, thus less communication overhead, and therefore less simulation time and better speed up.

Fig. 7.6 shows the speed up comparison between the 03/2017 and 10/2020 XDEM versions using the baseline test case. The 03/2017 version was performed using a full MPI configuration, as it was the only parallel configuration available. The latter version used the fastest parallel configuration, in full OpenMP to take advantage of the OpenMP implementation introduced during this doctoral research (chapter 6). The simulations were performed on one node (28 cores).

We observe in Fig. 7.6 that XDEM most recent version 10/2020 outperforms the 03/2017 version as it has a better speed up. Indeed, the recent version, with all the latest optimizations, has a speed up of 24 over 28 cores ($\sim$ 86% of parallel efficiency compared to sequential run) while the 03/2017 version has got only 12 speed up

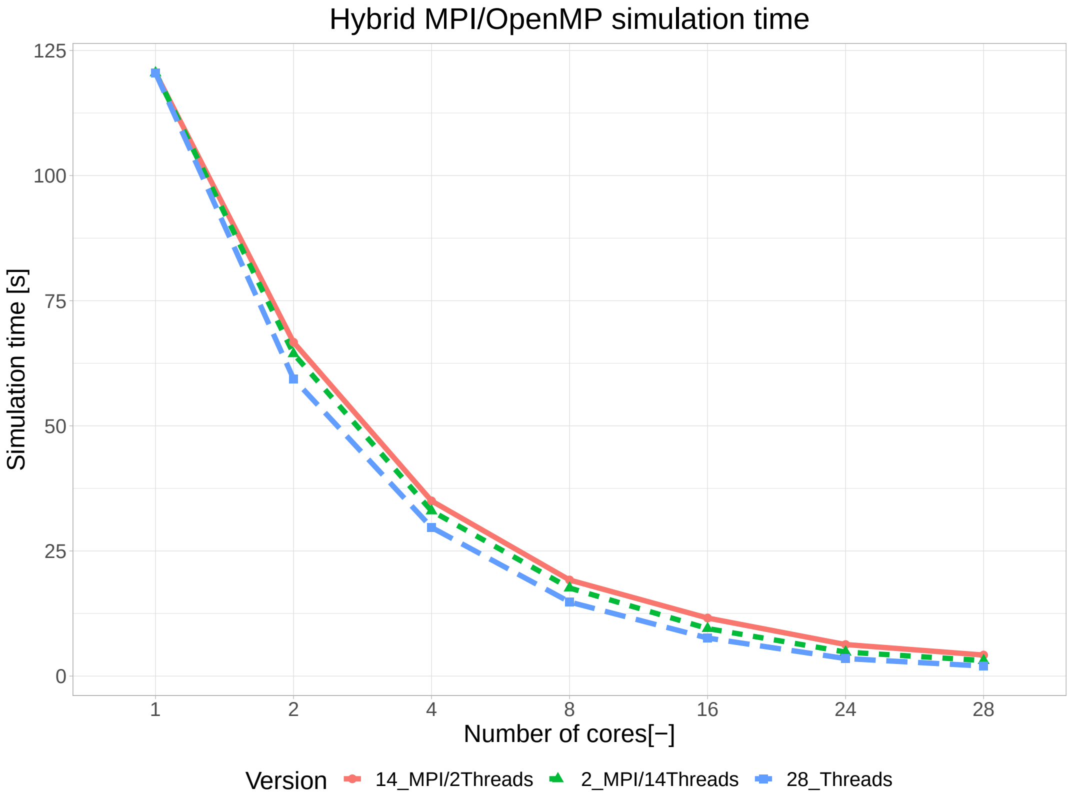


FIGURE 7.5: Strong scaling of the main loop's simulation time of three hybrid MPI/OpenMP parallel configurations. The number of cores is on the x-axis, and the simulation time on the y-axis. The executions was made on one node with 28 cores (exclusive reservation).

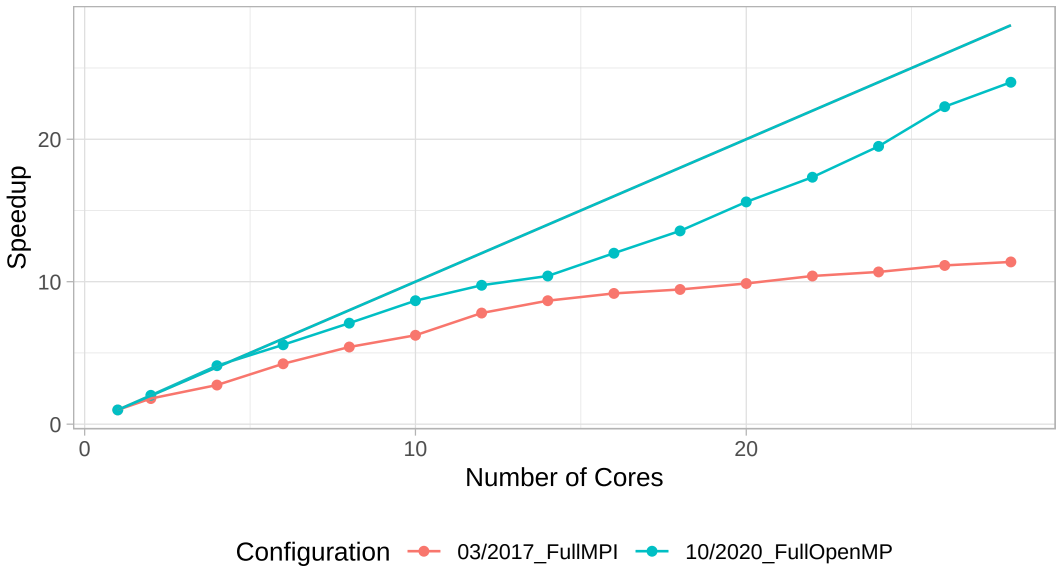


FIGURE 7.6: Speed up comparison between 03/2017 XDEM version in Full MPI configuration and 10/2020 XDEM version in Full OpenMP configuration. The x-axis represents the number of processes and the speed up on the y-axis. The executed was made on one node with 28 cores.

over 28 cores ($\sim$ 43% of parallel efficiency). Therefore, it is evident that a shared memory approach over one node is the best strategy to follow. Thus, it justifies the OpenMP implementation for simulations on workstations, personal computers, and large supercomputers (where the hybrid configuration is the best strategy).

In Fig. 7.7, we compare the improvement percentage (between sequential and parallel (on 28 cores) simulation times) of some essential parts of the XDEM code: the broad-phase, the narrow-phase, the integration (dynamic and conversion), and the dynamic models. The improvement is defined as follow:

$$Improvement = \frac{T_{seq} - T_{para}}{T_{seq}} \tag{7.1}$$

where T_{seq} and T_{para} are respectively the sequential and parallel (28 cores) simulation times.

The main gain was made in the broad and narrow-phases whose improvements went from 55% to 91% and 47% to 78% respectively. That is a significant gain as it plays a massive part in the overall gains. A critical gain was also made in the dynamic

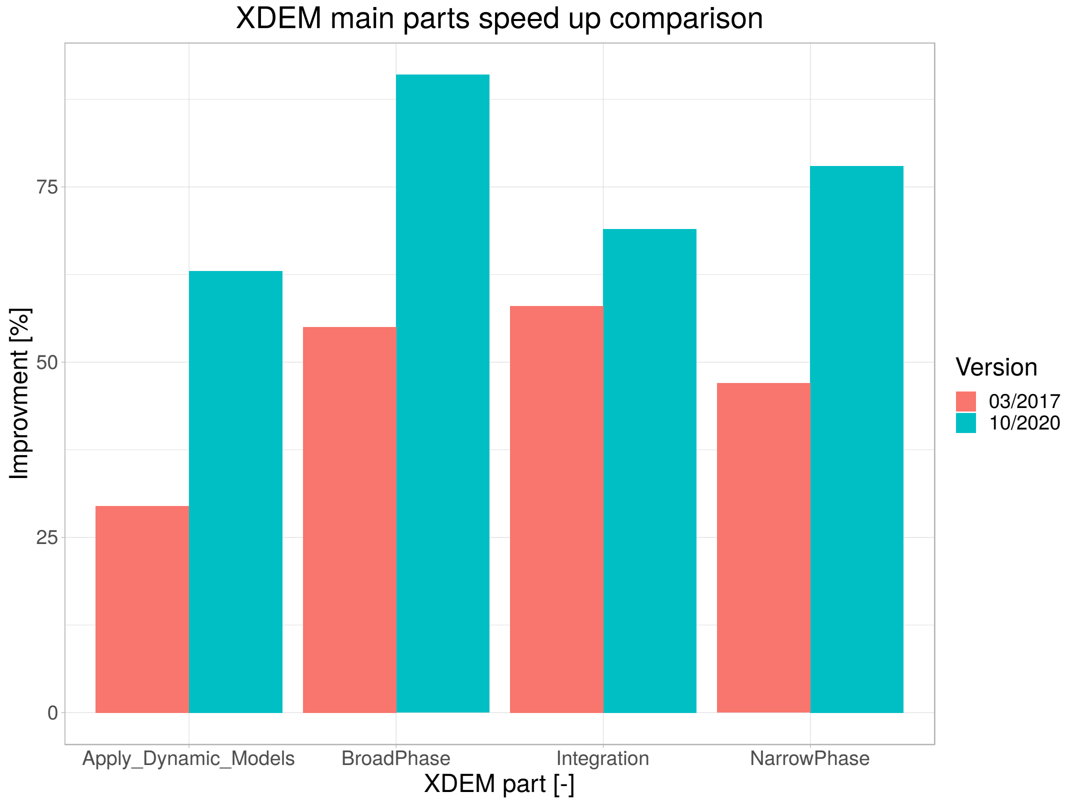


FIGURE 7.7: Improvements comparison between 03/2017 XDEM version in Full MPI configuration and 10/2020 XDEM version in Full OpenMP configuration. The comparison is made between sequential and parallel simulations using 28 cores. We compare the improvements of some critical parts of the XDEM code: the broad-phase, the narrow-phase, the integration (dynamic and conversion), and the dynamic models. The x-axis represents the critical parts of the XDEM code and the improvement in percentage (compared to sequential) on the y-axis.

models' module as the speed up went from 29% to 63%. The lesser gain that was accomplished is the Integration module with speed up going from 58% to 69%.

7.3 Large scale XDEM-OPENFOAM coupling performance

In this section, we want to study the behavior of a coupled XDEM+OPENFOAM simulation at a large scale. The goal is to evaluate the performance of XDEM in this particular coupling use case. For this purpose, we selected a test case which is big enough (in domain size and number of particles) to be executed with thousand of cores. Therefore, we measured the main loop's simulation time using different parallelization schemes within our SMP nodes to analyze the performance of the dam-break (see the case in section 7.3.1 below) XDEM-OPENFOAM test case.

7.3.1 Dam break test case

The Dam break is a very common benchmark for two-phase flow simulations. The case domain is a box of dimensions. $0.2m \times 0.1m \times 0.3m$ in which a column of water of extension $0.05m \times 0.1m \times 0.1m$ is located in the left corner and where two layers of spherical particles are disposed (see Fig.7.8). The first and bottom layers are composed of light particles of a $7.5mm$ radius, and the upper and layer are composed of heavy particles of a $10mm$ radius.

It should be noted, as shown in Fig. 7.8 that our configuration does not contain any intermediate obstacle.

7.3.1.1 Configuration

The entire case comprises 2.35 million particles interacting with the column water in an XDEM-CFD coupling approach. The benchmark was originally chosen to highlight the benefit of using a multi-scale DEM-VOF method over the classical DEM-VOF method (Pozzetti and Peters, 2018).

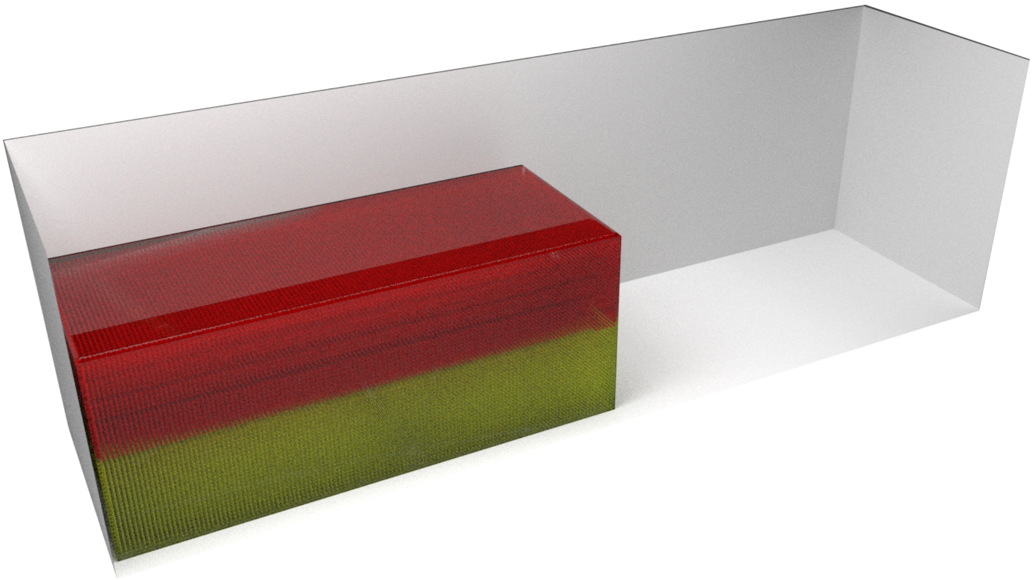


FIGURE 7.8: Dam break initial configuration. Light particles (bottom) in yellow and heavy particles in red (upper) are initially positioned within a column of water.

Our multi-scale DEM-VOF method uses a dual-grid multi-scale approach with a coarse grid that performs the coupling between CFD and DEM code at a bulk scale, while a finer and non-uniform grid is adopted to discretize the CFD equations. An interpolation strategy between the grids ensures the correct exchange of information between the bulk scale at which the inter-physics coupling is performed and the fine fluid scale at which the fluid equations are solved. It has been shown in (Pozzetti and Peters, 2018) that the approach produces grid-convergent results and provides a higher accuracy if compared to a standard DEM-VOF method.

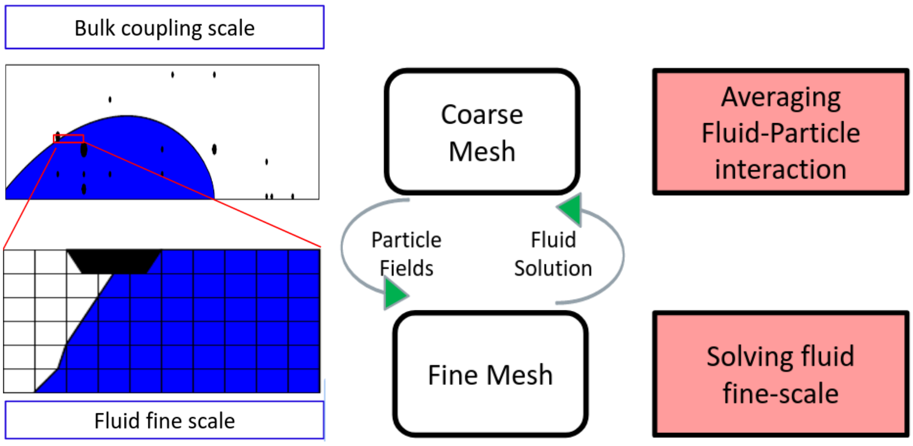


FIGURE 7.9: Different length scales in high-Stokes three-phase flows: bulk (coarse) scale and fluid fine scale (left figure). Schematic of the solution procedure for the bulk and fine length-scale in the simulation. The two boxes represent the different models adopted, while the arrows show the communication between the scales schematically. A coarse grid (top) is used to perform the volume averaging and to solve the fluid-particle interaction. Particle-related fields are mapped to the supporting domain (bottom) then, a finer grid is used to solve the fluid equations (Pozzetti and Peters, 2018).

We discretized the fine grid into $10M$ CFD cubic cells (identical) while the coarse grid is discretized into $500k$ cubic cells (see Fig. 7.10). The gas (light) phase has a density of $1kg/m^3$ and a viscosity of $10^{-5}Pas$. The density and viscosity of the liquid (heavy) phase are respectively $1000kg/m^3$ and $10^{-3}Pas$ (see Tab.7.3).

A linear dashpot impact model with a spring constant of $1200N/m$ is used for particle-particle and particle-wall collisions. We also used a restitution coefficient of 0.9 and a friction coefficient of 0.3.

Phase \ Property	*Density*	*Viscosity*
Gas	$1kg/m^3$	$10^{-5}Pas$
Liquid	$1000kg/m^3$	$10^{-3}Pas$

TABLE 7.3: Gas and liquid phases properties.

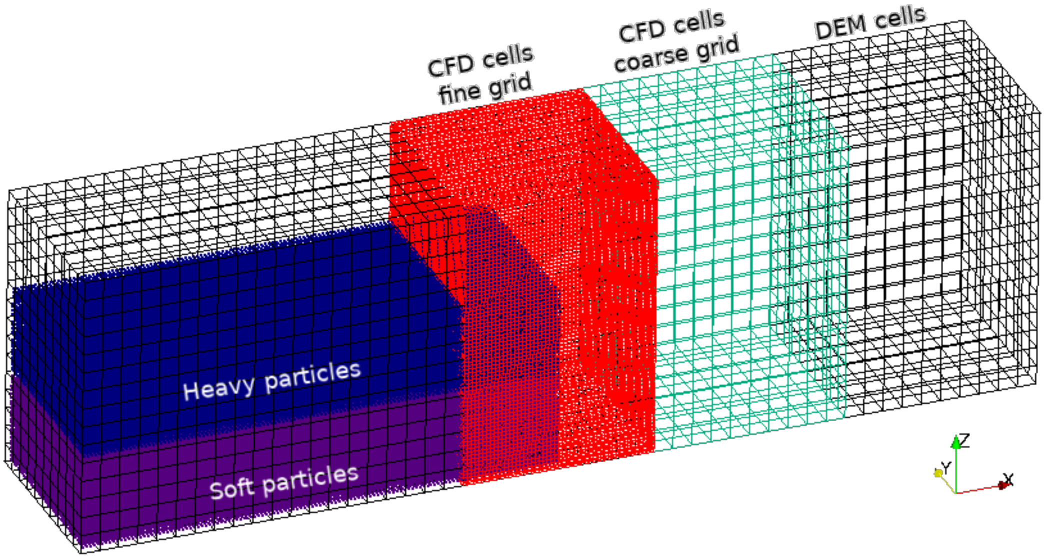


FIGURE 7.10: Three-phase dam-break multi-scale strategies.

7.3.1.2 Parallel set-up

The multi-scale DEM-VOF method is accomplished by coupling the XDEM and the OPENFOAM software, both of them having parallelization capabilities. As detailed in chapter 5, we now can perform hybrid *MPI+OpenMP* simulations within the XDEM framework. OPENFOAM, on the other hand, has only *MPI* parallelization approach allowing to discretize the fluid mesh (fine grid). Parallelization makes it possible to amortize the additional cost induced by choosing a fine grid for the fluid phase.

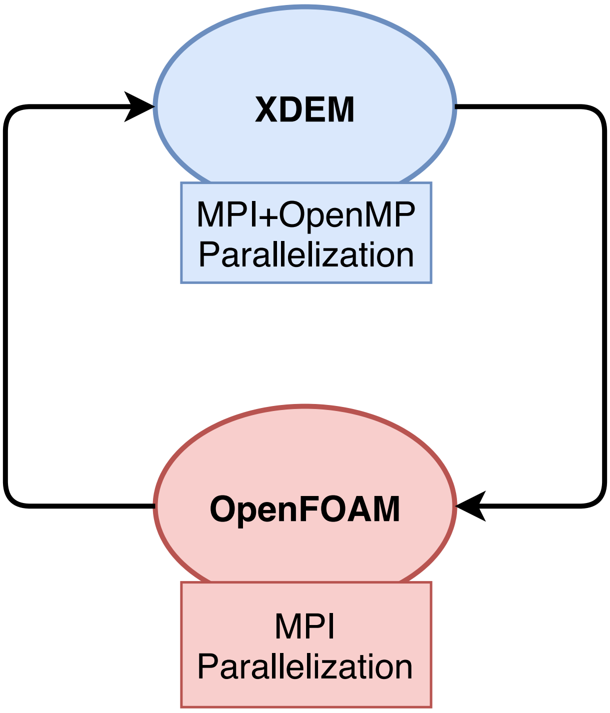


FIGURE 7.11: XDEM and OPENFOAM parallelization strategies. XDEM uses an hybrid MPI+OpenMP approach while OPENFOAM is partitioned using only MPI.

Fig. 7.11 describes the parallelization strategies of XDEM and OPENFOAM software. OPENFOAM is always used in full MPI parallel configuration as it is the only parallel strategy available. On the other hand, XDEM is used with both full MPI and hybrid MPI+OpenMP parallel strategies. We also use different configurations in the hybrid strategy by varying the number of MPI processes per node and the number of OpenMP threads per MPI process.

The experiments were carried out using the `Iris` cluster of the university of

Luxembourg (Varrette et al., 2014). The *Iris* cluster is the most powerful computing platform available within the University of Luxembourg (since 2017) running on **CentOS Linux** operating system (see Fig. 7.12). It's composed of 196 nodes, to say 5824 cores (28 cores/node) for a theoretical peak performance (**RPeak**) of $1.095 PFlops$. The nodes are distributed between CPU *Skylake and Broadwell* processors nodes (168) and GPU *NVIDIA Tesla V100 SXM2* nodes (28). The nodes are connected through an **Infiniband EDR (100Gb/s)** network and use three different parallel file systems for data storage: **GPFS** ($10GiB/s$ read and write), **Lustre** ($10GiB/s$ read and write) and **Isilon OneFS**.

FIGURE 7.12: The Iris cluster computing nodes.

7.3.2 Strong scalability and Speed Up

This section presents the speed up results performed on 85 nodes to say 2380 cores. The goal is to compare different hybrid configurations with the same number of cores to get the best strategy for running a dam-break case. Three different parallel configurations have been tested:

- Full MPI run. All 2380 cores are used for MPI processes to decompose the XDEM and CFD domains.
- Hybrid MPI/OpenMP run. This configuration uses 2 MPI processes per node, each process on a different socket, and 14 OpenMP threads per MPI process. The XDEM and CFD domains are, therefore, decomposed using 170 processes.
- Hybrid MPI/OpenMP run. This configuration uses 14 MPI processes per node, 7 processes on a socket, and 2 OpenMP threads per MPI process. The XDEM and CFD domains are, therefore, decomposed using 1190 processes.

In Fig. 7.13, we compute the speed ups relative to the simulation time on one node (for each configuration) rather than with the simulation time on one core (takes much time). We can see that the two different hybrid configurations (2MPI/14Threads and 14MPI/2Threads) have overall higher speed up than the full MPI configurations (on 2380 cores). Indeed, the latter configurations have respectively 1690 and 1380 of speed up over 2380 cores while the full MPI has only a speed up of 1142 (on 2380 cores). That is a gain of 32% of speed up between the best hybrid and the full MPI configurations.

We can also observe that below 1000 cores, the full MPI configuration outperformed the hybrid configuration with a better speed up. The configurations with more MPI processes performed better than the ones with more OpenMP threads. At a first look, it is surprising as we would expect the hybrid configurations always to perform better than the full MPI because they have fewer MPI processes and, thus, less communication. However, looking at the simulation times, it turns out that the full MPI is the slowest strategy and the hybrid configurations are faster. The full MPI has a better speed up only because it is the slowest strategy on one node, which simulation time is used to compute the speed up. From 1000 cores, the full MPI configuration speed up falls compared to the hybrid configurations. It can be explained by the fact that we have reached a performance wall. The communication between the MPI processes has an overhead that has a significant impact on the

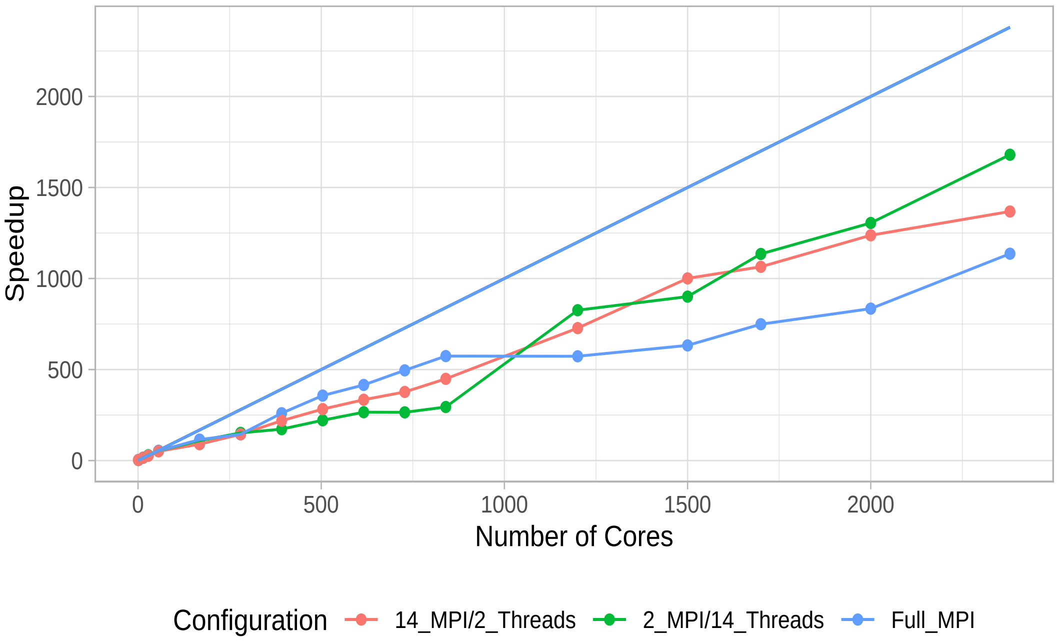


FIGURE 7.13: Speed up of hybrid MPI/OpenMP runs with different process/thread configurations. The x-axis represents the number of processes and the speed up on the y-axis.

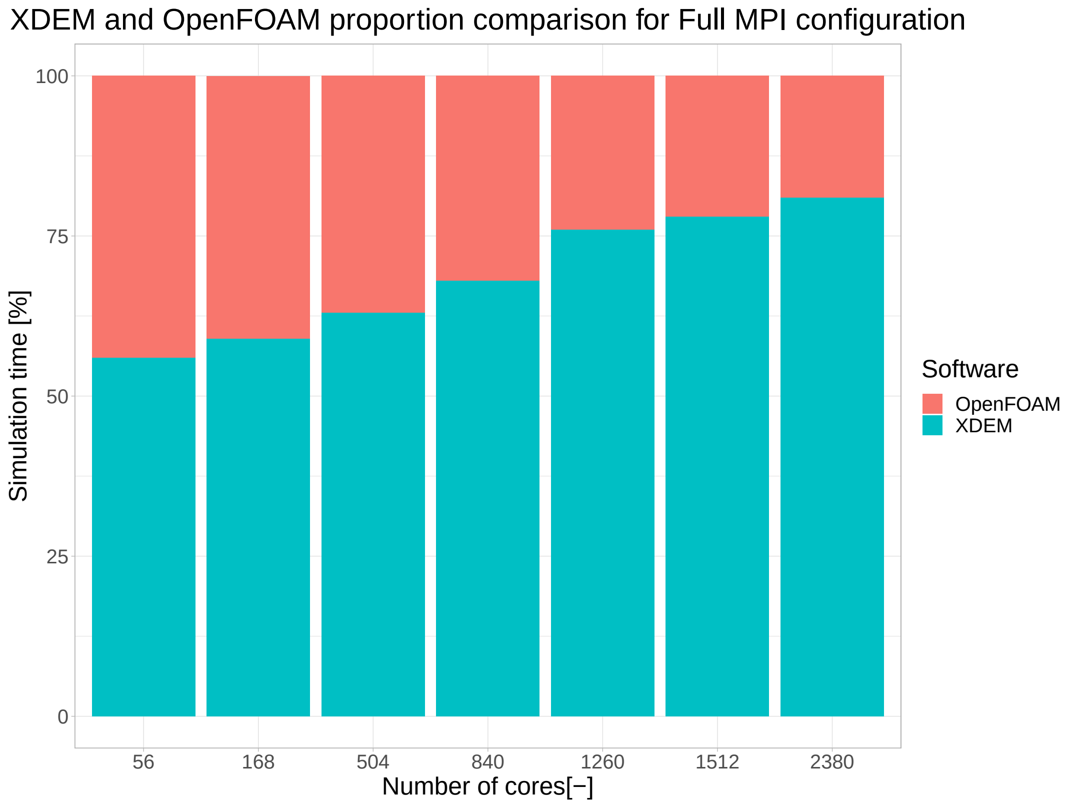
overall performance. On the other side, having a hybrid configuration allows us to minimize the MPI processes. Thus, we minimize the MPI inter-processes communications while using the same amount of resources and having a better performance rate.

As the dam-break test case uses a coupling CFD-DEM model with XDEM and OPENFOAM, it is imperative to understand and highlight the contribution of each software. Fig. 7.14 shows the simulation time proportion of XDEM and OPENFOAM for different number of cores. Fig. 7.14a shows the proportions in the full MPI parallel configuration of XDEM and OPENFOAM, and we can notice that XDEM's contribution (proportion in the simulation time) increases as the number of cores increases. With 56 cores, the contributions are even as we have 56% and 44% proportions for XDEM and OPENFOAM respectively. But the proportion of XDEM increases and reaches 81% with 2380 cores while the OPENFOAM proportion is down to 19%. XDEM is, therefore, a performance bottleneck in the coupling approach with OPENFOAM when using the full MPI parallel configuration. Fig. 7.14b shows the simulation time proportions in the hybrid (2 MPI processes per node and 14 OpenMP threads per MPI process) parallel configuration for XDEM and full MPI configuration for OPENFOAM, and in this configuration, we can notice that the two contributions do not evolve much as the number of cores increases. XDEM's contribution goes from 52% to 63% for 56 and 2380 cores, respectively, and those contributions are better (lower) than in the full MPI configuration highlighting the benefits of the hybrid configuration. The justification lies in the communication and computation load-imbalances defined as the uneven distribution of communication/computation works, respectively, across the MPI processes. Therefore, it is evident that our XDEM presents load imbalance issues as the number of MPI processes increases. That is why the hybrid configuration performs better as it has fewer MPI processes and less computational and communication load-imbalances.

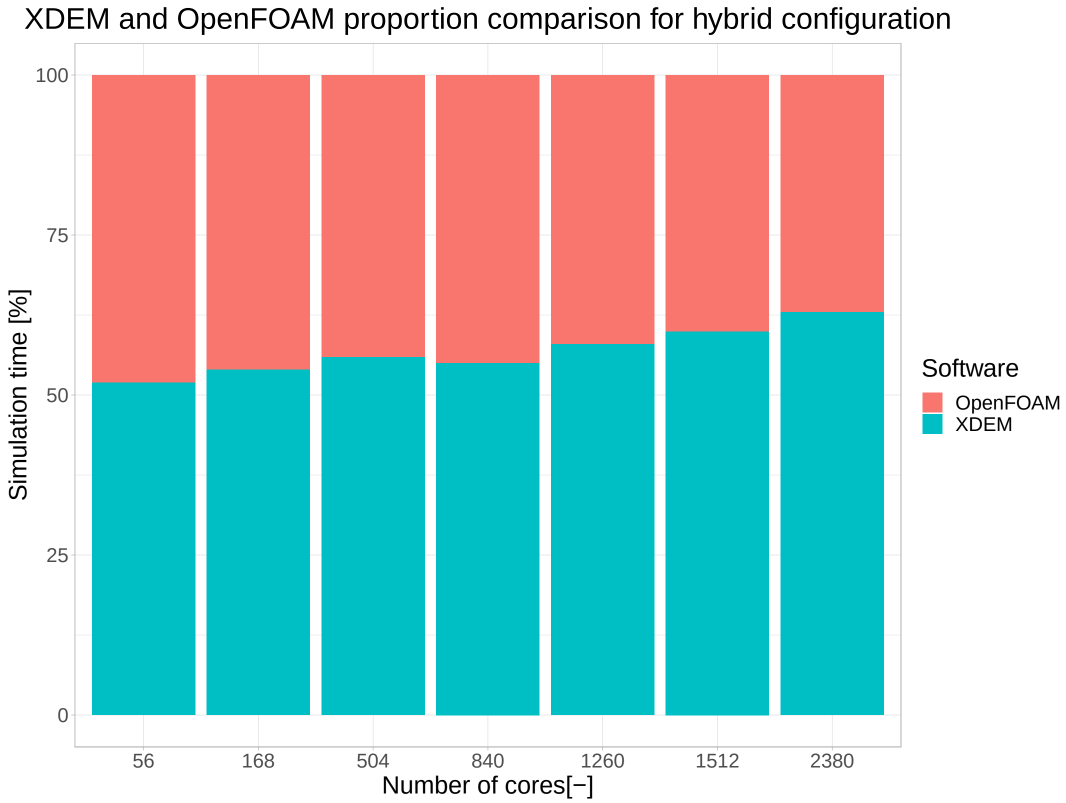
In Fig.7.15, we compare XDEM and OPENFOAM load-imbalances for the dam-break for different numbers of cores in the full MPI configuration.

We observe in Fig. 7.15, as it can be suspected, the more processes there are, the more computation imbalances there are. The OPENFOAM imbalance goes from 3% with 56 cores to almost 100% with 2380 cores. It also presents a steady and continuous increase. Typically, for OPENFOAM, with 2830 cores, there is a computation load difference of 100%, meaning there is 100% computation load difference between the MPI process with the lowest computation load and the MPI process with the highest computation load. On the other hand, the XDEM computation load-imbalance goes from 3% with 56 cores to almost 160% with 2380 cores. We observe a big load-imbalance jump between 840 and 1260 (from 25% to 90%) cores. It is in accordance with the scalability and speed up results presented in Fig. 7.13, where we noticed a fall down of the speed up around 1000 cores. With 2830 cores, for XDEM, there is a computation load difference of 160%, meaning there is 160% computation load difference between the MPI process with the lowest computation load and the MPI process with the highest computation load.

Fig. 7.16 presents the computation load imbalance for the two hybrid MPI/OpenMP strategies. Without any surprise, the hybrid strategy with 2 MPI processes per node and 14 OpenMP threads per MPI process has the lowest communication load imbalance as it has the least number of MPI processes. It imbalance goes from 0.15%



(A) Simulation time percentage for Full MPI configuration for XDEM and OPENFOAM.



(B) Simulation time percentage for Hybrid MPI/OpenMP.

FIGURE 7.14: XDEM and OPENFOAM simulation time proportion comparison. The left figure compares the proportions for a Full MPI configuration for XDEM. The right figure compares the proportions for a hybrid 2 MPI processes per node and 14 OpenMP threads per MPI process for XDEM.

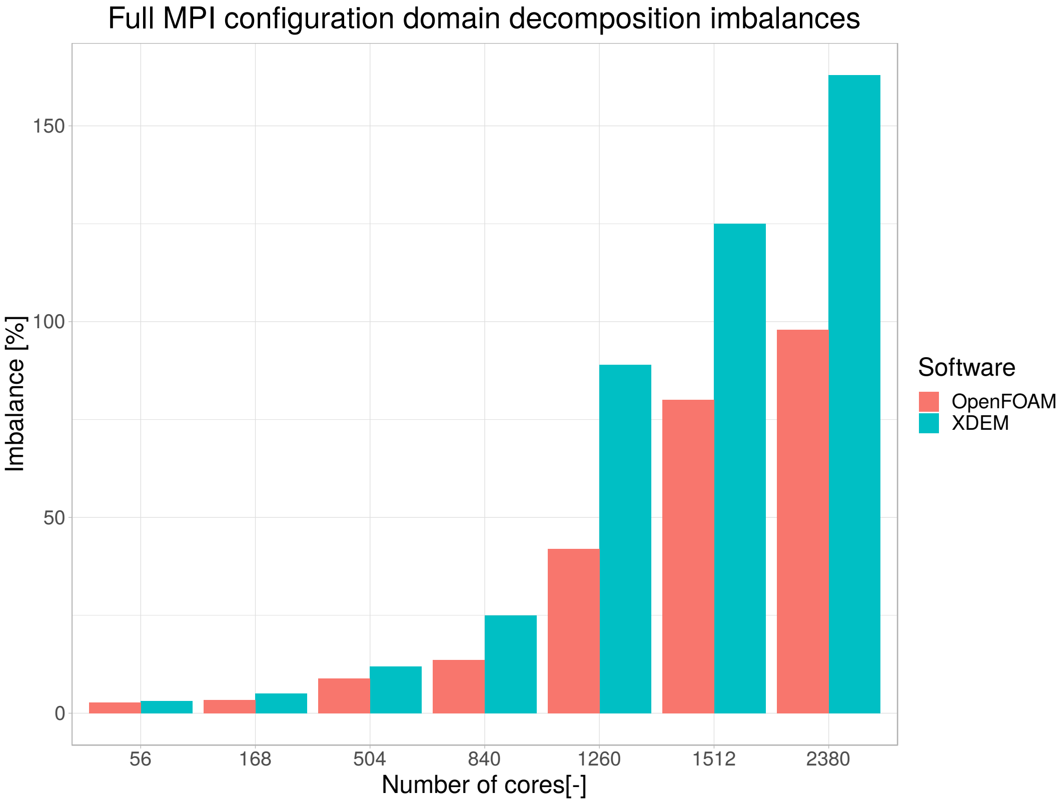


FIGURE 7.15: XDEM and OPENFOAM load-imbalances for the dam-break domain decomposition in the Full MPI parallel configuration.

with 56 cores to only 17% with 2380 cores (170 MPI processes). It is nine (9) times lower than the imbalance of the full MPI strategy on 2380 cores. The difference is as expected because this hybrid approach has fourteen (14) times less MPI processes than the full MPI approach. On the other hand, the hybrid strategy with 14 MPI processes per node and 2 OpenMP threads per MPI process imbalance goes from 1.2% with 56 cores to 94% with 2380 cores (1190 MPI processes). It is 1.7 times lower than the imbalance of the full MPI strategy on 2380 cores. The difference is as expected again because this hybrid approach has two (2) times less MPI processes than the full MPI approach. So with the hybrid parallelization, the XDEM domain decomposition presents even less load communication imbalance than OPENFOAM, which clearly benefit the coupling simulations.

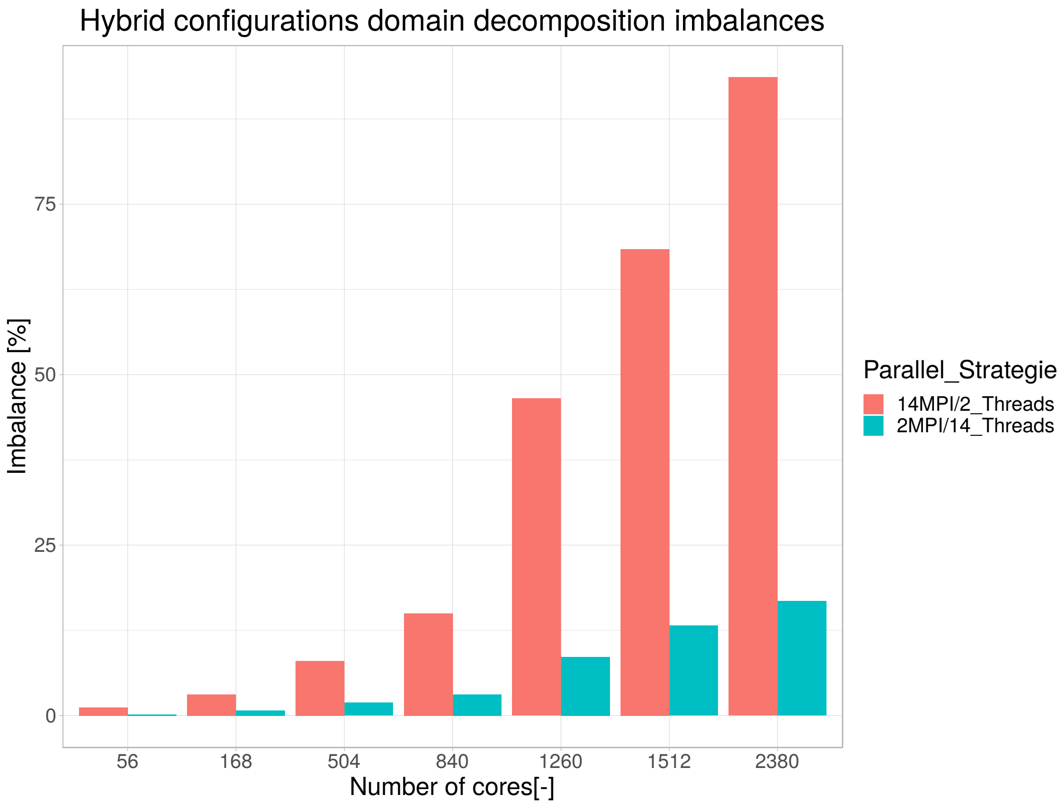


FIGURE 7.16: XDEM load-imbalances for the dam-break domain decomposition in the hybrid MPI/OpenMP parallel configuration.

We can conclude that the OPENFOAM software scales better than our XDEM (in full MPI parallel configurations) for this dam-break test case mainly because it presents a better load balance. However, the imbalance loads of XDEM in the hybrid strategies are better (than full MPI) as they use less MPI process and thus have less load imbalance. As shown in Fig. 7.10, the particles are located in the first half of the domain in the initial configuration of the dam-break, leaving the rest of the domain empty. As a result, despite our efficient partitioning algorithms (Zoltan), it generates load and communication imbalance among MPI processes that strongly impact the performance as it generates overhead and imbalance. It should be noted that only hundreds of steps were performed without dynamic load balance for the performance analysis. Therefore, the dam-break remains more or less in its initial configuration. The latter results state once again the benefit of using hybrid MPI/OpenMP simulation at a large scale, specially for coupling cases with OpenMP.

7.4 Conclusion

In this chapter, we presented the scalability results of the different implementation made to the XDEM code. The goal was to assess the gains brought by all the contributions conducted during the doctoral research. For this purpose, we introduced the dam-break test case with more than two million particles (2.35M) with two different piles of particles (light and heavy). We also presented our multi-scale DEM-VOF method that uses a dual-grid multi-scale approach with a coarse grid that performs the coupling between CFD and DEM code at a bulk-scale; a finer non-uniform grid is adopted to discretize the CFD equations.

The parallelization set up was to use a hybrid MPI/OpenMP approach to execute XDEM while using a pure MPI configuration for the CFD domain with OpenFOAM. The scalability tests were performed on the Iris cluster of the University of Luxembourg on 85 nodes, as to say 2380 cores. We then compare different parallel configurations: pure MPI approach and hybrid 2 MPI processes per node and 14 OpenMP threads per MPI process and 14 MPI processes per node, and 2 OpenMP threads per MPI process. We showed that our approach allows reaching a speed up of 1690 and a parallel efficiency of more than 70% on 2380 cores with the hybrid configuration of 2 MPI processes per node and 14 OpenMP threads, which is better than the pure MPI approach. We compared the communication load imbalance of three parallel strategies with OpenFOAM's and we noticed that the full MPI presented the lowest speed up because it also has the most communication imbalance. Our hybrid configurations has also lower communication imbalance than OpenFOAM highlighting the benefit of using hybrid strategies in XDEM+OpenFOAM coupling simulations. We also noticed that the best strategy in the dam-break case is different from the hopper case described in chapter 6. The main difference between these two cases is the load imbalances in the MPI processes. Therefore, when the MPI domain decomposition presents a high load (communication and computation) imbalance, we recommend using the maximum OpenMP threads possible in the hybrid MPI/OpenMP strategies.

We also compared the scalability and speed up of the early 03/2017 and later 10/2020 XDEM versions. The results compared the scalability and speed up of the main loop simulation time on one computing node. It appears that the full OpenMP parallel configuration is the best strategy to chose on one node. We then compared the main parts of the XDEM simulation loop: the broad and narrow-phases, the integration (dynamic and conversion), and the dynamic models. The results indicated a massive gain in the collision detection part due to the algorithm implementation introduced in chapter 3 and the new Verlet buffer developed in chapter 4.

Finally, we have presented the benefit of using the OpenMP new capability to perform hybrid MPI/OpenMP configurations. Apart from the OpenMP's performance gains, it also allows us to use more and more computing resources while bringing more performance.

Chapter 8

NUMERICAL ANALYSIS OF A GRATE FIRING COMBUSTION PROCESS

8.1 Abstract

Biomass as a renewable energy source continues to grow in popularity to reduce fossil fuel consumption for environmental and economic benefits. In the present contribution, the combustion chamber of a 16 MW geothermal steam super-heater, which is part of the Enel Green Power "Cornia 2" power plant, is being investigated with high-performance computing methods. For this purpose, the extended discrete element method (XDEM) developed at the University of Luxembourg is used to simulate the moving wooden bed and the combustion chamber above it in a high-performance computing environment. The XDEM simulation platform is based on a hybrid four-way coupling between the Discrete Element Method (DEM) and Computational Fluid Dynamics (CFD). In this approach, particles are treated as discrete elements coupled with heat, mass, and momentum transfer to the surrounding gas as a continuous phase. Besides the equations of motion for individual wood particles, the differential conservation equations for mass, heat, and momentum are solved, which describe the thermodynamic state during thermal conversion. The consistency of the numerical results with the actual system performance is discussed in this paper to determine the potentials and limitations of the approach [a].

Keyword

Biomass **Combustion** **XDEM** **CFD** **ENEL GREEN POWER SpA**

---

[a] This chapter was published as an article in Infub12 conference

8.2 Introduction

Grate firing is one of the fundamental techniques used for heat and power generation by combustion of biomass, as it allows the burning with little or no planning of a wide range of fuels, including waste. Grate systems can be classified into different categories depending on the manner the fuel transport is achieved, i.e., through just gravity (stationary sloping grates), conveyor belts (traveling grates), moving bars (forward-acting, reverse-acting, and reciprocating grates) or shaking movement (vibrating grates) (Yin et al., 2008; Liyan et al., 2013). The grate system helps the motion, mixing, and conversion of the fuel, thus improving the combustion rate and minimizing the presence of unburnt carbon and pollutant emissions. The numerical investigation by (Peters et al., 2005) was carried out using the Discrete Element Method (DEM) for the mix and segregation of biomass particles in a forward-acting grate. (Sudbrock et al., 2011) studied whether DEM simulations can quantitatively predict solid material mixing behavior on grates by analyzing the influence of operational parameters such as bar velocity, bar stroke, and moving patterns. (Simsek et al., 2009) studied the motion ($2D/3D$ DEM) and chemical conversion (heating, drying, pyrolysis, and char combustion) of solid fuels in a packed bed composed of polydisperse spherical particles moving on a forward-acting grate and coupled to the reacting flow above the combustion chamber. (Samiei and Peters, 2013) used a DEM model to outline the particle residence time on forwarding and backward-acting grates. (Sun et al., 2015) analyzed the effects of amplitudes and frequencies of moveable grates in reciprocating grates. The works cited above justify DEM's use in the study of particle motion and mix in reciprocating grate firing. The biomass combustion process on a moving grate involves multi-scale, multi-phase, and multi-species phenomena, which increase the difficulties of predicting the biomass conversion, ultimately altering the performances. There exist two main different numerical approaches categorizing a biomass simulation to tackle all these interacting phenomena: single-phase and multi-phase models. The single-phase directly solves the gas phase in the freeboard through Computational Fluid Dynamics (CFD) (Yin et al., 2008). The effect of the particle bed is taken into account by assigning the correct boundary conditions to the freeboard CFD model (Patronelli et al., 2017). In multi-phase, both solid and gaseous phases are taken into account by using an Eulerian-Eulerian (Yang et al., 2004; Kurz, Schnell, and Scheffknecht, 2012) or Eulerian-Lagrangian approaches. In the latter, the particle bed is treated by the DEM method for the motion and drying, devolatilization, and char oxidation for the conversion process. The particles are treated as discrete elements coupled with heat, mass, and momentum transfer to the surrounding gas as a continuous phase. Besides the equations of motion, the differential conservation equations for mass, heat, and momentum are solved for individual particles, which describe the thermodynamic state during thermal conversion. In the present work, we simulate the behavior of a large-scale, i.e. $15.7 MW_{th}$, reciprocating grate system, which is part of a hybrid plant, integrating biomass and geothermal energy, by applying a CFD-XDEM approach. The aim is to propose a numerical approach that can combine a low computational cost by the use of high performance computing, allowing the realistic use of the design with a sufficient accuracy of the results for industrial applications.

8.3 Numerical model and simulation conditions

The grate has three different moving sections to ensure good mixing of the biomass parts and an appropriate residence time. The primary air (PA) enters from below the grate and is split into four different zones (sections). Furthermore, a secondary air (SA) is injected at high velocity straight over the fuel bed through two circular nozzles. A Flue Gas Re-circulation (FGR) is present and partly injected through two jets along the vertical channel and partly from below the grate (see Fig. 8.1). Fig. 8.2a is a top view of the 3D representation of the grates and particle motion. Fig 8.2b shows the gas phase circulation through the combustion chamber with velocity arrows. The surface bed temperature is displayed, and the particles are colored according to their surface temperature distribution.

The geometric data and operating conditions of the biomass furnace are summarized in Tab. 8.1 and can be found in the master thesis (LUPI, 2017b).

In this model, the biomass fuel bed is composed of 80% wood-chips and 20% of agricultural residues corresponding to the biomass plant's real conditions. The

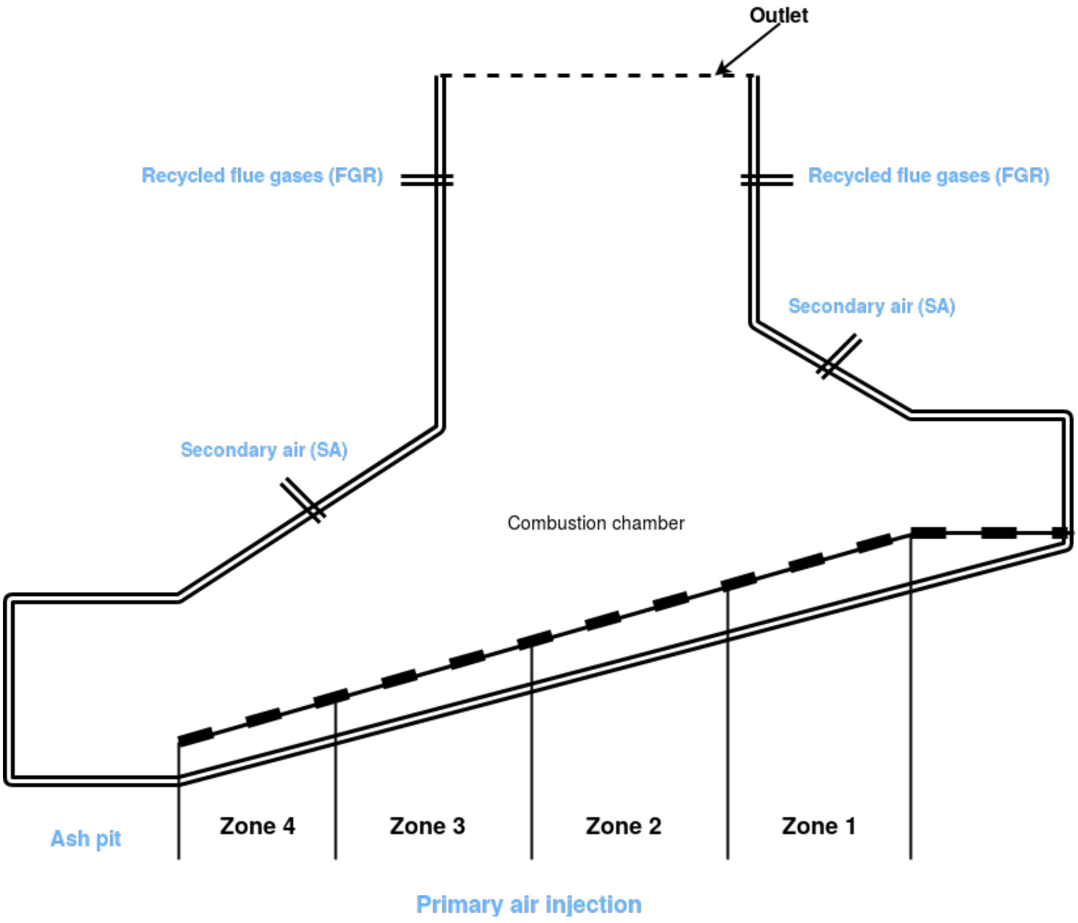


FIGURE 8.1: Biomass combustion chamber 2D design.

TABLE 8.1: Characteristics and operating conditions of the superheater. PA = primary air, SA = secondary air

Input Thermal Power [MW]	15.7
Fuel Mass Flow Rate [kg/h]	5433
PA Mass Flow Rate [kg/h]	20745
SA Mass Flow Rate [kg/h]	8890
FGR Mass Flow Rate [kg/h]	30000
PA Temperature [°]	200
Lower SA jet arrays	2×7
Upper FGR jet arrays	2×6
Grate Tilt [°]	15
Specific Heat Load [kW/m^2]	715
Number of Inlet Sections	4
Independent Groups of Mobile Steps	3

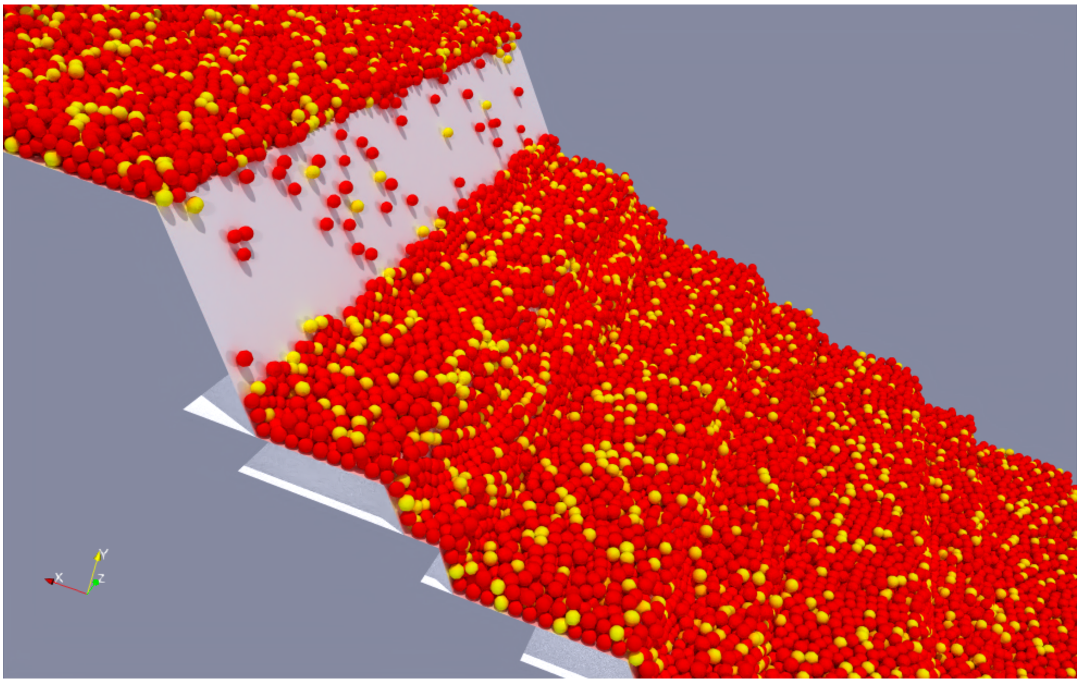
proximate and ultimate analysis of the biomass is provided in Tab. 8.2.

TABLE 8.2: Biomass analysis. ar = as received, daf = dry ash free

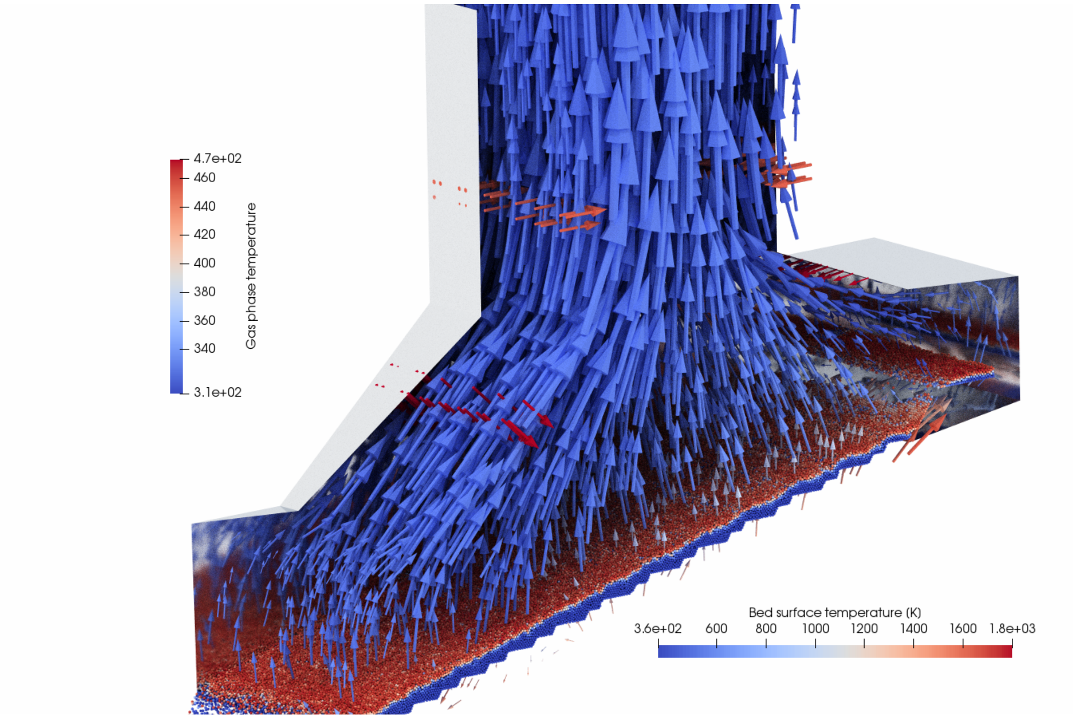
Properties	Woodchips	Residues	Mixture
Mix Fraction [%wt,ar]	80	20	100
Granulometry range [mm]	-	-	5÷400
Average particle size [mm]	-	-	30
Moisture [%wt,ar]	34.0	51.0	37.4
Volatiles [%wt,ar]	53.7	35.6	50.1
Fixed Carbon [%wt,ar]	11.3	8.3	10.7
Ashes [%wt,ar]	1.0	5.1	1.8
Carbon [%wt,daf]	49.60	51.15	49.82
Hydrogen [%wt,daf]	5.95	6.23	5.99
Oxygen [%wt,daf]	44.23	41.67	43.86
Nitrogen [%wt,daf]	0.22	0.95	0.33

The numerical model presented in this paper is based on a multi-phase approach. The biomass particles are taken into consideration via the XDEM (Peters, 2013), while the gaseous phase is described by CFD with OpenFoam.

XDEM is a novel and innovative numerical simulation technique that extends the dynamics of granular materials or particles as described through the classical discrete element method (DEM) by additional properties such as the thermodynamic state, stress/strain for each particle (Peters et al., 2015). Thus, the particles' combustion on the moving beds in the furnace is processed by XDEM through conduction, radiation, and conversion (Mahmoudi et al., 2016b) along with the interaction with the surrounding gas phase, accounted for by CFD. The coupling of CFD-XDEM as an Euler-Lagrange model is used in this paper, the fluid phase is a continuous phase handled with an Eulerian approach, and each particle is tracked with a Lagrangian approach. Energy, mass, and momentum conservation are applied for each particle. The interaction of particles with each other in the bed and the surrounding gas phase is considered. Hence, the sum of all particle processes represents the entire process, like a fixed bed. The full 2D/3D multi-phase CFD-XDEM model simulations of the biomass with the particle dynamics and conversion are performed using the XDEM code, while the gaseous phase with the primary air (PA), the secondary air (SA), and the flux gas recycled (FGR) is computed with CFD using the extend-OpenFOAM software. The current CFD-XDEM coupling is a complete model that especially



(A) Top view of the wood inlet and forward moving grates. The particles are colored according their composition: wood chips (red particles) and residues (yellow particles).



(B) Velocity arrows, surface bed temperature and gaseous phase temperature in 3D combustion chamber.

FIGURE 8.2: Biomass combustion chamber 3D views.

suits a biomass furnace, taking into account both particle motion and conversion, including the interaction with the surrounding gas. An individual particle can have solid, liquid, gas, or inert material phases (immobile species) at the same time. The different phases can undergo a series of conversion through various reactions that can be homogeneous, heterogeneous, or intrinsic. In the porous particles system, Darcy's law is applied for the chemical species transport (the gas captures in the porous structure is considered ideal) under boundary conditions specified by the interaction with the surrounding gas phase. The equilibrium model is used for the drying process, assuming that the water vapor within the particle's pores is in equilibrium with the liquid and the bound water in the biomass combustion process. The shrinking phase of the particle radius is taken into account in the current simulation. Further details about the model can be found in (Mahmoudi et al., 2015). Pyrolysis is described with three independent reactions expressing decomposition of wood to its main products (char, tar, and gas), as given:

$$wood \rightarrow char \tag{8.1}$$

$$wood \rightarrow tar \tag{8.2}$$

$$wood \rightarrow \nu CO + \nu CO_2 + \nu H_2O + \nu H_2 + \nu CH_4 \tag{8.3}$$

$$wood \rightarrow \nu_{CO} CO + \nu_{CO_2} CO_2 + \nu_{H_2O} H_2O + \nu_{H_2} H_2 + \nu_{CH_4} CH_4 \tag{8.4}$$

Tar may also be subjected to a secondary crack reaction and form light gases:

$$tar \rightarrow \gamma_{tar_{inert}} tar_{inert} + \gamma_{CO} CO + \gamma_{CO_2} CO_2 + \gamma_{H_2} H_2 + \gamma_{CH_4} CH_4 \tag{8.5}$$

Where in the above reactions, ν and γ are the mass fractions (Di Blasi, 2000; Wurzenberger et al., 2002).

In the whole process and the composition of gas products, homogeneous reactions during the gas phase play an essential role. During pyrolysis, the volatiles released can react with oxygen, generating heat. In this analysis, we are using the following four gas-phase reactions:

$$CO + 0.5O_2 \rightarrow O_2 \tag{8.6}$$

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O \tag{8.7}$$

$$2H_2 + O_2 \rightarrow H_2O \tag{8.8}$$

$$tar + 2.9O_2 \rightarrow 6CO + 3.1H_2 \tag{8.9}$$

The heterogeneous reactions, gasification, and combustion can occur to the remaining char from the wood's pyrolysis. These heterogeneous reactions are detailed in the following reactions:

$$\gamma C(s) + O_2 \rightarrow 2\left(\gamma - 1\right) CO + \left(2 - \gamma\right) CO_2 \tag{8.10}$$

$$C(s) + H_2O \rightarrow CO + H_2 \tag{8.11}$$

$$C(s) + CO_2 \rightarrow 2CO \tag{8.12}$$

where the partition coefficient γ is evaluated as (Johansson, Thunman, and Leckner, 2007):

$$\gamma = \frac{2\left[1 + 4.3e^{-\frac{3390}{T_P}}\right]}{2 + 4.3e^{-\frac{3390}{T_P}}} \tag{8.13}$$

The rate expression and the kinetic data of these reactions can be found in (Mahmoudi et al., 2016b). The char combustion and gasification reaction rates are based on the particle's oxidizing/gasifying agent's partial pressure. Particles are assumed to be isotropic in their scaling model and their properties to change along the radius. The distribution of temperature and chemical species within the particles is assessed through a solution of one-dimensional transient conservation equations describing particle heat-up, drying, pyrolysis, and char oxidation/gasification with boundary conditions at the particle surface deriving from the gas phase CFD solution (Mahmoudi et al., 2015). The gas flow through the bed's void space is modeled by applying the governing equations for a flow passing through a porous medium, which is solved using the finite volume method with OpenFOAM as a CFD tool. The Favre-averaged formulation is used by closing Reynolds stresses with the Boussinesq hypothesis and employing the standard $k - \epsilon$ model to determine the turbulent viscosity μ_t. The interaction of chemistry with turbulence in the gas phase is treated through the Partially Stirred Reactor (PaSR) model. Each computational cell is divided into a reacting part and a non-reacting part. The former is represented as a perfectly stirred reactor where all chemical species are assumed to be homogeneously mixed and reacted. After the reactions have taken place, the species are mixed due to turbulence for a mixing time τ mix and the resulting concentration represents the final one for the entire, partially stirred reactor (Kadar, 2015). The mixing time-scale depends on the local turbulence as:

$$\tau_{mix} = C_{mix}\sqrt{\frac{\mu_{eff}}{\rho\varepsilon}} \tag{8.14}$$

where C_{mix} is a constant, μ_{eff} is the effective dynamic viscosity (i.e., $\mu_{eff} = \mu + \mu_t$) and ε is the turbulent dissipation rate.

The radiative flux coming from the walls ensures the particle ignition. The coupling model enables applying a different flux to various bed surfaces, heat propagating from top to bottom of the particle bed through conduction, taking the different visual factors into account with the combustion chamber walls. Heat then propagates through conduction from the upper to the lower layers of the particle bed. The incident radiative flux is then estimated as:

$$q_{rad} = \alpha\sigma\left(T_{wall}^4 - T_b^4\right) \tag{8.15}$$

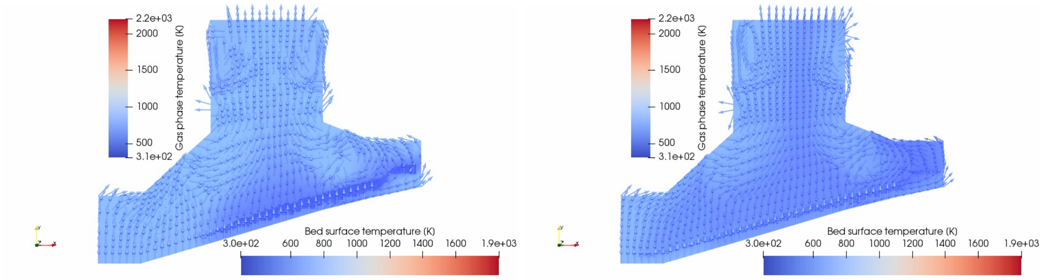
where $\alpha = 0.75$, $\sigma = 5.67 \cdot 10^{-8}$ W/ m^2 K^4 is the Boltzmann constant. The average wall temperature was set as $T_{wall} = 1200$ K as available from thermocouples positioned near the combustion chamber wall, while the average biomass temperature was that of the first two biomass layer (and thus changes with iterations).

As previously noted, in addition to PA, the SA and RFG injection nozzles were considered in the 2D/3D coupling model. The two types of biomass, wood-chips and agricultural residues, have been used to compose the furnace's particle bed. Such piles join the domain via a specific source that matches the volumetric system of the drawer. The grate has alternated longitudinal movement in the three mobile units, with roughly 80% being the advance and the remaining 20% being moved to the initial position. The movement of each series is changed by the next $30s$.

8.4 Results and performance analysis

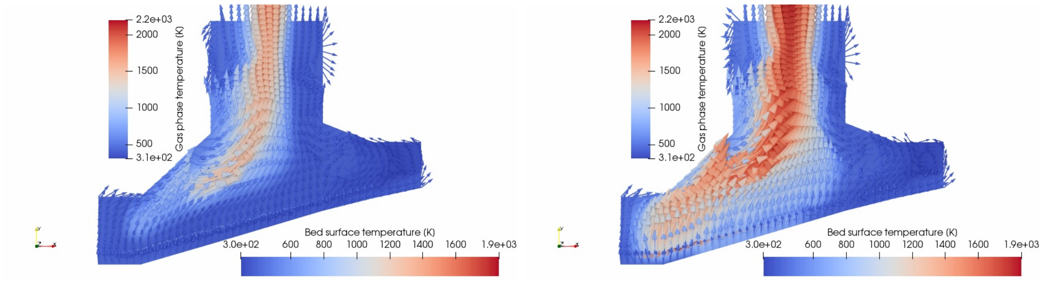
The Fig. 8.3 illustrates the biomass bed temperature evolution in time and the heat-up of the combustion chamber. The lower side colors bar represents the bed surface temperature while the left middle color bar represents the gas phase temperatures. After $5s$ simulation, both bed particles and gases are still cold and do not show any heat-up phase (Fig. 8.3a). However, at $50s$ of simulation time, the firsts layers of the bed start to warm up slightly even if the gas phase temperature appears relatively the same as previous (Fig. 8.3b). From then on, around $250s$, the ignition takes place in the combustion chamber of the furnace with some high-temperature gas streaks (around $1500K$) that are due to the oxidation of volatile gases coming from the fuel bed (Fig. 8.3c). After $600s$, the gas streak temperature substantially increases, and the combustion chamber appears to be characterized by those streaks with temperatures locally exceeding $2000K$ (Fig. 8.3d). The same gas streaks scheme occurs when simulating until $1200s$ suggesting that we reached a stable state after $600 \sim 700s$. During the pseudo steady-state, the position of the gas streaks can oscillate due to the grate movement.

The figures above show a change in shape as well as temperature when looking at the fuel surface. The pseudo-steady conditions state, achieved around a time simulation of $600 \sim 700s$, is confirmed in Fig. 8.4, which shows the average surface bed temperature as a function of time. The high-temperature gas-streaks that seem to govern the furnace's thermal are unlikely to be predicted by a freeboard-only approach (without fuel bed) but can be highlighted with a CFD-XDEM approach. Additionally, we can note that the more prominent streak with higher temperature is located in the grate area (4) close to the ash pit, suggesting that the significant release



(A) Combustion chamber bed temperature at time $t = 5s$.

(B) Combustion chamber bed temperature at time $t = 50s$.



(C) Combustion chamber bed temperature at time $t = 250s$.

(D) Combustion chamber bed temperature at time $t = 600s$.

FIGURE 8.3: Gas phase temperature distribution at different simulation time: 5s (upper left), 50s (upper right), 250s (lower left), 600s (lower right).

of volatile happens at the end of the biomass path, particularly close to the ash pit. Before that, the streaks are less pronounced and more consistent.

The average composition of all bed particles was calculated and reported in Fig. 8.5 to analyze the different phenomena occurring in the combustion chamber throughout the grating length. It can be noticed that devolatilization can complete only near the outlet (length 7.5 in the Fig. 8.5); a very negligible amount of organic matter is still present in this zone. At the outlet, the particles contain, on average, 8% of char and 92% of ashes by wt., meaning that there remain some unburnt carbon resulting from incomplete combustion. The 92% of ash at the outlet corresponds to the initial residual ash present in the dry biomass within the feeding (it represents 5% of the drying mass).

These results are consistent with actual data obtained from the residual solid's sampling in the industrial plant. From Fig. 8.5, we can notice the presence of little moisture in the particles up to length 6.0, signaling that the drying process is still not complete for many particles in these lengths. It can be explained that some particles

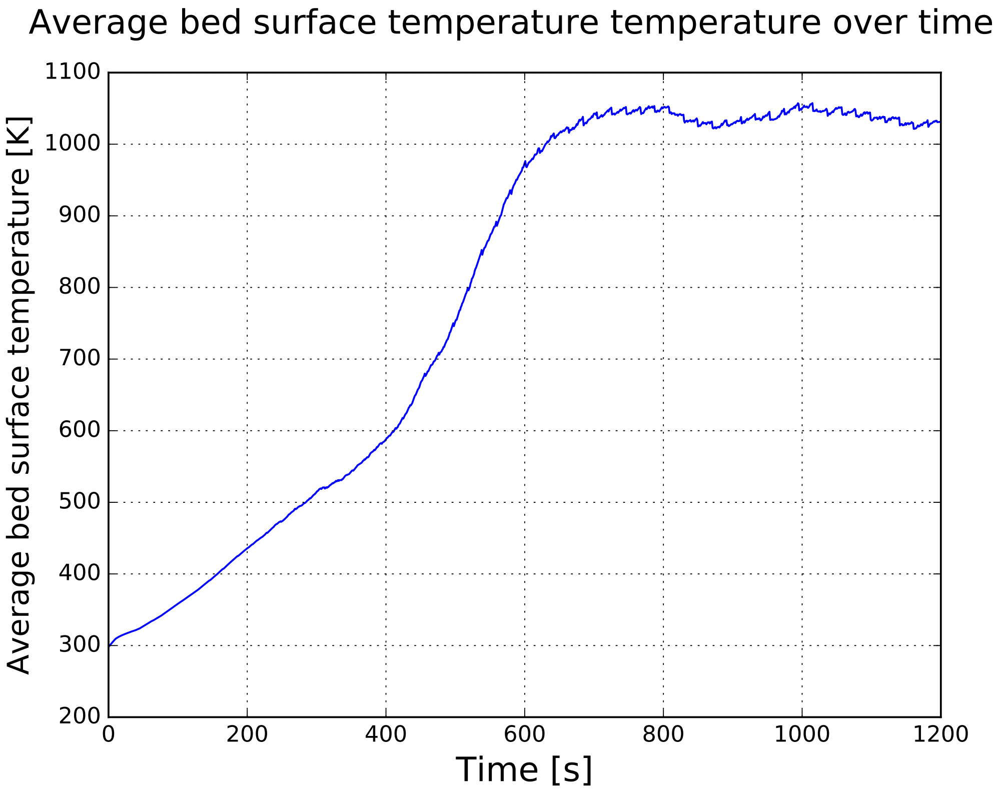


FIGURE 8.4: Surface bed temperature distribution.

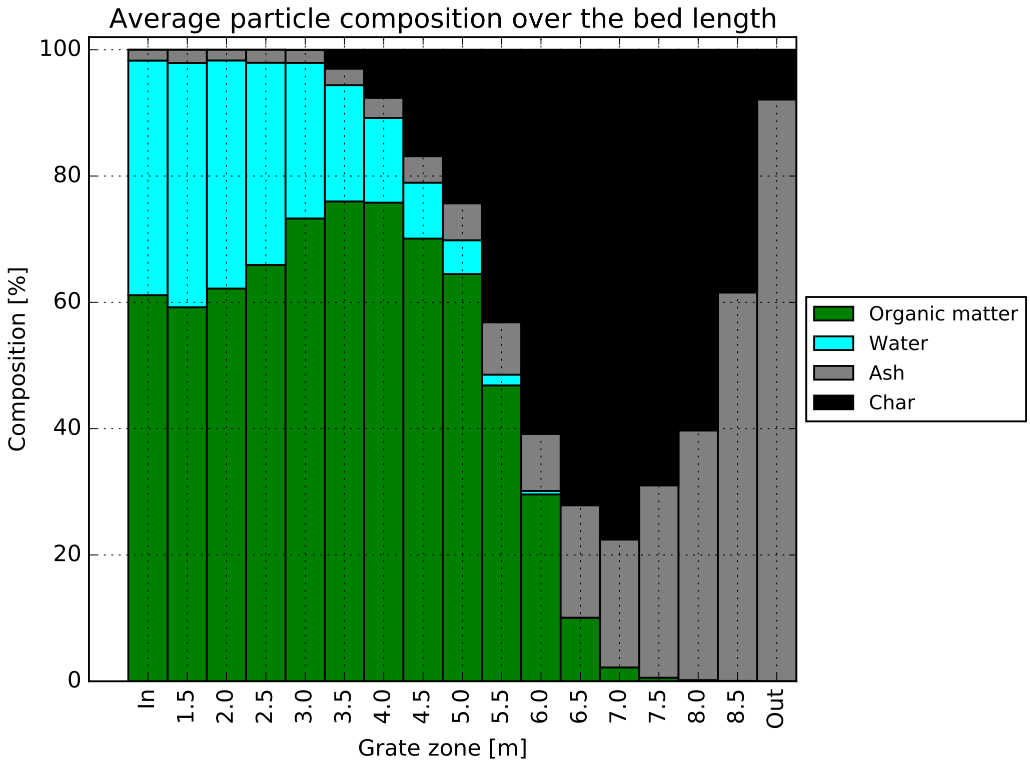


FIGURE 8.5: Particle composition in the inlet, outlet and along the grate length.

located at the bottom of the bed are being pushed upwards later on the grate, thus not able to complete their drying process.

Fig. 8.6 shows the average composition (wood, water, ash, and char) of particles alongside the grates.

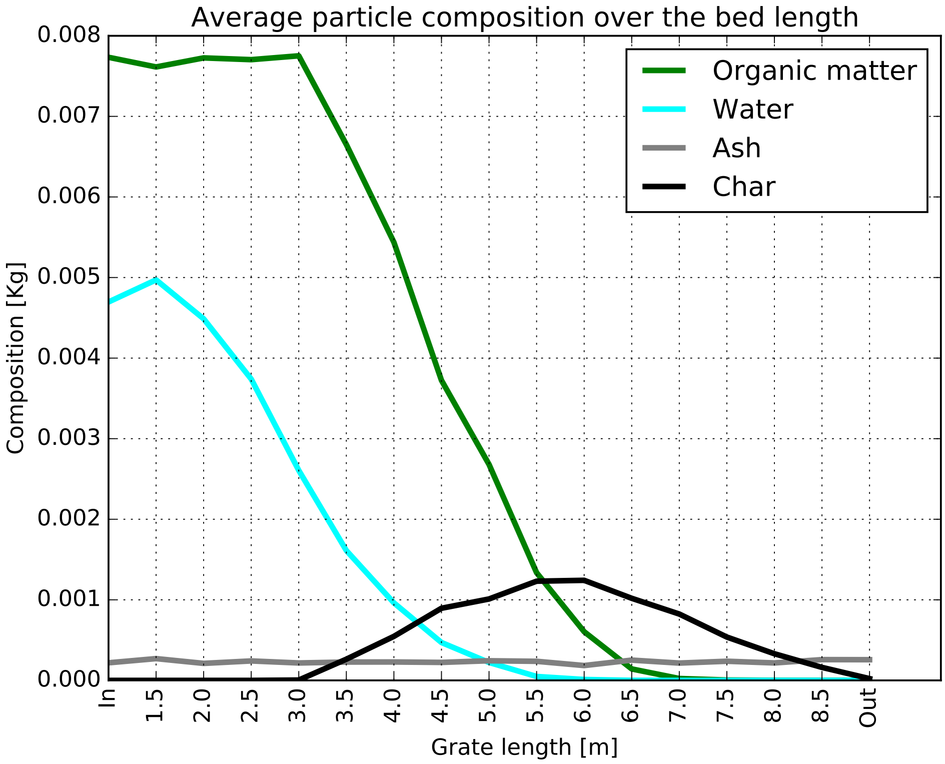


FIGURE 8.6: Particle average composition in the inlet, outlet and along the grate.

We can also notice how the moisture evaporates within the particles to a complete drying process in the length of 6.0. The amount of ash remains constant throughout the grating length and corresponds to the initial 5% ash introduced since the feeding. The whole simulation of the complete 2D case has been run for around 1200s in approximately five (5) hours thanks to advanced optimization techniques and the parallelization of XDEM using the OpenMP approach (Checkaraou et al., 2018a). The experiments were carried out using the Iris cluster of the University of Luxembourg, which provides 168 computing nodes for 4704 cores. The nodes used in this study feature a 128 GB of memory and have two Intel Xeon $E5-2680v4$ processors running at 2.4GHz; that is to say, a total of 28 cores per node. The nodes are connected through a fast, low-latency EDR InfiniBand ($100 GB/s$) the network is organized over a fat-tree topology. Fig. 8.7 compares the simulation times of OpenFoam and XDEM in a sequential and parallel coupling simulation. The proportion of XDEM in the coupling simulation time goes from 80% in sequential to 55% in parallel using 28 OpenMP threads, showing a speedup of 26 over 28 threads (93% compared to sequential).

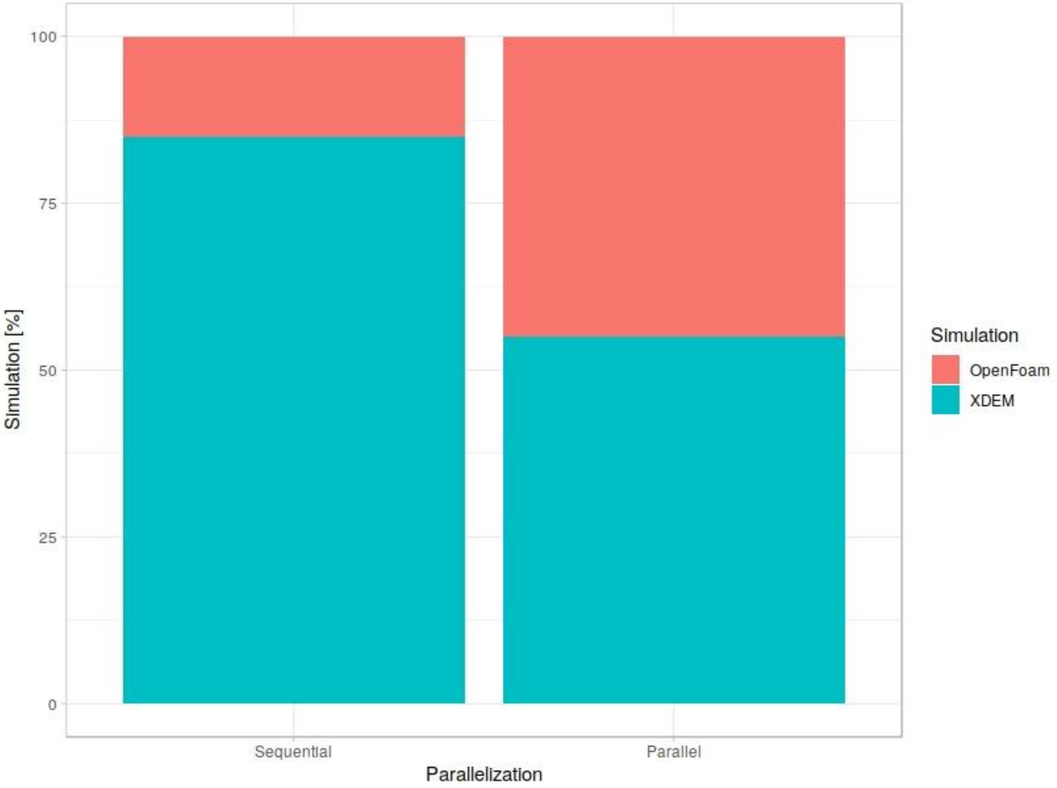


FIGURE 8.7: Time proportion of XDEM and OpenFoam in the coupling simulation.

8.5 Conclusion

In this paper, we presented a full 2D/3D CFD-XDEM model (given the plant's industrial size) to investigate biomass combustion in a large-scale reciprocating grate. In our coupling model, the XDEM software is used to simulate the granular flow along with the grates (with dynamic and conversion), and OpenFoam dealt with the surrounding gases. We showed that there exists a pseudo-steady state after $600 \sim 700 s$ allowing a deeper analysis of the composition of the particles through the grate lengths. Importantly, we were able to spot the level of unburnt carbon, i.e., approximately 8%, which was inconsistent with the evidence in the real plant. Therefore, the expected thermal field complied with the few provided experimental data, acknowledging a proper consideration of the interaction between chemical kinetics and turbulence. In particular, the model allows us to understand the effect of flue gas recirculation on the combustion process injection. First and foremost, the computational cost was relatively low due to the 2D/3D feature of the multi-phase CFD-XDEM model, the XDEM code's parallelization, and the high-performance computers. It was a fundamental aspect to suggest using the present numerical model for a real biomass plant's practical operation.

Part III

Conclusion

Chapter 9

Conclusion

9.1 Summary

In this thesis, an optimization process of the XDEM software to perform large scales and parallel simulations was discussed. The different contributions presented in this work were developed during the whole length of the Ph.D., and this thesis is just a collection of some of the most significant contributions presented and published in different international conferences and journals. The optimization process consists of evaluating XDEM software performances by conducting a series of profile using benchmarks. This step is crucial as it spots the code's weaknesses, the hotspots, and the part to be addressed and optimized. As a result, we have spotted particle collision detection as the most consuming computational time segment. We, therefore, developed a complete C++ framework for collision detection algorithms that were included within the XDEM software. Besides, we also proposed a new approach of the Verlet list technique that takes the particles' local flow regime conditions and enhances the algorithm's performance. We have developed a full OpenMP layer within the XDEM platform that unlocks new parallel simulation strategies targeting HPC systems.

The first contribution was introduced in chapter 3 with the development of a C++ framework for testing the broad-phase algorithms. We evaluate and compare ten different Broad-Phase Collision Detection algorithms (spatial partitioning and sorting, grids, and trees) while considering a large DEM test case. It appears that the choice of the best algorithm is a trade-off between many criteria, including the size of the search space, the number of particles, and memory usage. The new algorithms were validated through a series of tests on different cases. They were afterward integrated within the XDEM platform with the possibility to select a different type of algorithms depending on the running case.

The second and third contributions are the development of an original Verlet list implementation for DEM that takes the particle flow regime into account when selecting the skin margin to enhance the algorithm's efficiency further. The approach is presented with a performance comparison with the standard and usual Verlet approach. We also conducted an optimization study to determine which optimum skin margin gives the best computing performance depending on particle local flow regime parameters (velocity, solid fraction, number of particles, the ratio of particle size to cell size). Hundreds of simulations were performed using the DAKOTA software to solve the optimization problem (using genetic algorithms). Therefore, we proposed a polynomial function expressing the optimum skin margin as a function of the simulation parameters. The two contributions were introduced in chapters 4 and 5.

The fourth contribution is a complete implementation of an OpenMP strategy within the XDEM software. It was presented in chapter 6 and was aimed to target the High-Performance Computers (HPC) and systems by offering more parallel strategies. Adding an OpenMP layer needed a consequent code implementation and data reorganization, which required a vast code change effort as XDEM is legacy. It undoubtedly brings performance for parallel executions but also sequential executions. This contribution is a significant accomplishment of the doctoral candidate during his Ph.D. as it unlocks new possibilities and considerably speeds up the simulations (as shown in chapter 7). It allows new complexes and bulky cases to be simulated. 3D Blast-furnace and Biomass simulations (chapter 8) were therefore performed in a very reasonable time, accelerating the different research projects.

Finally, chapter 8 presents a biomass combustion test case. It a large scale application coupling XDEM-OPENFOAM with thousands of particles. The case investigates biomass combustion in a large-scale reciprocating grate. In the adopted coupling approach, the XDEM software is used to simulate the granular flow (particles) along with the grates (with dynamic and conversion), and OPENFOAM dealt with the surrounding gases. We were able to spot the level of unburnt carbon, i.e., approximately 5% that was in consistency with evidence in the real plant. This test case acknowledged a proper consideration of the interaction between chemical kinetics and turbulence in our coupling model as the expected thermal field complied with the few provided experimental data. In particular, the model allows for understanding the effect of flue gas recirculation on the combustion process injection. The proposed approach is thus strongly recommended as it also presents a relatively low computational cost, thanks to the 2D/3D feature of the multi-phase CFD-XDEM model, the XDEM code's parallelization, and the high-performance computer systems.

Overall, this thesis's different contributions developed during the doctoral research and not introduced in this thesis allow large scale and parallel simulations that were out of reach. Our contributions considerably speed up the XDEM simulation platform and enable new research to be conducted and the exploration of new fields, such as 3D models of biomass combustion chambers and complete Blast furnaces.

9.2 Future tasks

The research presented in this thesis was intended to optimize the XDEM software to run large scale and parallel DEM simulations using *HPC* capabilities and resources. For this purpose, the entire collision detection process was reviewed with the implementation of different algorithms. The Verlet was also enhanced with a newly developed approach that further consider the particle flow regime. A full OpenMP implementation of XDEM was designed to take full advantage of the shared memory resources and the NUMA configuration of the supercomputers computing nodes. Given these points already developed in this thesis, we propose to deepen and expand the following topics:

1. The Hierarchical grid is among the algorithm implemented in the C++ benchmark and presented a relatively good performance. Nevertheless, it presents an auspicious performance in the literature (Krijgsman, Ogarko, and Luding, 2014; Kroiss, 2013; Fan et al., 2011; Weinhart et al., 2020) for polydiverse simulation case. Therefore we recommend implementing a hierarchical grid algorithm within the XDEM framework. The new feature will add new capabilities as it will be possible to efficiently simulate polydiverse cases as the rotating drum (or mill charge) and granular flow for calibration and validation.

2. Using machine learning techniques to further predict the optimal skin distance in the Verlet buffer method is another essential task that future work should focus on. At the moment, we use a simple polynomial function to predict the optimal skin distance. With more simulation data, better techniques such as the *random forest*, *logistic regression*, or the *gaussian process* exist to predict the optimal skin distance accurately.

3. A major challenge when using a distributed memory parallelization strategy is to efficiently balance the load among the resources while maintaining the numerical solution's high accuracy throughout the computation. In DEM simulations, we often deal with particles dynamically moving through the entire domain going from to another region. There is, therefore, a demand

for a finer mesh in the regions where the particles go through. Thus, we recommend to implement the *A*daptative *M*esh *R*refinement (**ARM**) for DEM within the XDEM software. ARM is a method that adaptively refines the mesh in certain regions of the domain to increase the solution accuracy. An ARM can refine the mesh and the particle flow displacement for better collision detection and flow prediction. Such a high-performance computing technique is essential for computational efficiency in moving regions of interest for distributed-memory parallel computer architectures where domain decomposition is applied, especially with MPI. Moving regions of interest are dynamically deforming and migrating through the domain during simulations and require high spatial resolution of solution features (Rettenmaier et al., 2019). ARM coupled with *D*ynamic *L*oad *B*alancing (**DLB**) offer the benefit to effectively reduced computational effort on large scale and parallel simulations.

4. As presented in chapter 1, there is much existing software (open-sources and commercials) available in the literature (Granular Dynamics International and Software, 2020; "2.4 Theory Reference Guide, 2011"; Govender, Wilke, and Kok, 2016) using GPU and multi-GPU capabilities. They have been applied to many different applications with excellent performances. With the additional computing resources available with a GPU, such implementations make a massive simulation with complex shapes possible that were out of reach (non-convex polyhedra particles, particle breakage) (Wilke et al., 2016; Liu et al., 2020; Govender et al., 2018; Kasai et al., 2009; Chen et al., 2011). Moving the XDEM code from CPU to GPU will require an enormous amount of work, as the entire code would have to be rewritten from scratch. Faced with this constraint, we recommend instead only perform some parts on GPU. The collision detection process is an excellent candidate, as it has been demonstrated in the literature (Scott, 2020; Govender, Wilke, and Kok, 2015). An efficient data and results transfer between CPU and GPU coupled with the GPU computing, will surely bring performance benefits to the XDEM software.

Part IV

Bibliography

BIBLIOGRAPHY

Adams, Brian M et al. (2019). “DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.11 user’s manual”. In: *Sandia National Laboratories, Tech. Rep. SAND2010-2183*.

Allen, M. P. and D. J. Tildesley (1990). *Computer Simulation of Liquids*. Claredon Press Oxford.

Allen, Michael P and Dominic J Tildesley (2017). *Computer simulation of liquids*. Oxford university press.

Angeles, Luis and César Celis (2019). “Assessment of neighbor particles searching methods for discrete element method (DEM) based simulations”. In:

Awile, Omar et al. (2012). “Fast neighbor lists for adaptive-resolution particle simulations”. In: *Computer Physics Communications* 183.5, pp. 1073–1081.

Baraff, David (1992). *Dynamic simulation of non-penetrating rigid bodies*. Tech. rep. Cornell University.

Berger, R et al. (2015). “Hybrid parallelization of the LIGGGHTS open-source DEM code”. In: *Powder Technology* 278.

Bergstra, James and Yoshua Bengio (2012). “Random search for hyper-parameter optimization”. In: *Journal of machine learning research* 13.Feb, pp. 281–305.

Berry, Mike, George Cybenko, and John Larson (1991). “Scientific benchmark characterizations”. In: *Parallel Computing* 17.10-11, pp. 1173–1194.

Berryman, Sylvia (2004a). “Democritus”. In:

– (2004b). “Leucippus”. In:

Besseron, Xavier et al. (2013). “Unified Design for Parallel Execution of Coupled Simulations using the Discrete Particle Method”. In: *Proceedings of the Third International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*. Civil-Comp Press.

Boyd, Eric L et al. (1994). “A hierarchical approach to modeling and improving the performance of scientific applications on the KSR1”. In: *1994 International Conference on Parallel Processing Vol. 3*. Vol. 3. IEEE, pp. 188–192.

Chandra, Rohit et al. (2001). *Parallel programming in OpenMP*. Morgan kaufmann.

Checkaraou, Abdoul Wahid Mainassara et al. (2018a). “Hybrid MPI+ openMP Implementation of eXtended Discrete Element Method”. In: *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, pp. 450–457.

Checkaraou, Abdoul Wahid Mainassara et al. (2018b). “Predicting near-optimal skin distance in Verlet buffer approach for Discrete Element Method”. In: *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, pp. 450–457.

Chen, Jin et al. (2011). “Analysis of rice seeds motion on vibrating plate using EDEM.” In: *Nongye Jixie Xuebao= Transactions of the Chinese Society for Agricultural Machinery* 42.10, pp. 79–100.

Chialvo, Ariel A and Pablo G Debenedetti (1990). “On the use of the Verlet neighbor list in molecular dynamics”. In: *Computer physics communications* 60.2, pp. 215–224.

– (1991). “On the performance of an automated Verlet neighbor list algorithm for large systems on a vector processor”. In: *Computer Physics Communications* 64.1, pp. 15–18.

Clarke, Lyndon, Ian Glendinning, and Rolf Hempel (1994). “The Message Passing Interface standard”. In: *Programming environments for massively parallel distributed systems*. Springer, pp. 213–218.

Cohen, Jonathan D et al. (1995). “I-collide: An interactive and exact collision detection system for large-scale environments”. In: *Proceedings of the 1995 symposium on Interactive 3D graphics*, 189–ff.

Coumans, Erwin (2015). “Bullet Physics Simulation”. In: *ACM SIGGRAPH 2015 Courses*. SIGGRAPH ’15.

Cundall, P. A. and O. D. L. Strack (1979). “A discrete numerical model for granular assemblies”. In: *Geotechnique* 29, pp. 47–65.

DEM-Solutions, EDEM. “2.4 Theory Reference Guide, 2011”. In: *DEM Solutions: Edinburgh* ().

Devine, Karen et al. (2002). “Zoltan data management service for parallel dynamic applications”. In: *Computing in Science & Engineering* 4.2, pp. 90–97.

Di Blasi, Colomba (2000). “Dynamic behaviour of stratified downdraft gasifiers”. In: *Chemical engineering science* 55.15, pp. 2931–2944.

Doglio, Fernando (2015). *Mastering Python High Performance*. Packt Publishing Ltd.

Donoso, Alvaro Antonio Estupinan and Bernhard Peters (2018). “Exploring a Multiphysics Resolution Approach for Additive Manufacturing”. In: *JOM* 70.8, pp. 1604–1610.

Duran, Jacques (2012). *Sands, powders, and grains: an introduction to the physics of granular materials*. Springer Science & Business Media.

Ericson, Christer (2004). *Real-time collision detection*. CRC Press.

Erlangen Regional Computing Center, FAU (2019). *ECM Performance Model*. `https://hpc.fau.de/research/ecm/`.

Evans, Jason (2006). “A scalable concurrent malloc (3) implementation for FreeBSD”. In: *Proc. of the BSDCan Conference, Ottawa, Canada*.

Fabri, Andreas and Sylvain Pion (2009). “CGAL: The computational geometry algorithms library”. In: *GIC’09*. ACM.

Fan, Wenshan et al. (2011). “A hierarchical grid based framework for fast collision detection”. In: *Computer Graphics Forum*. Vol. 30. 5. Wiley Online Library, pp. 1451–1459.

Fang, X, J Tang, and H Luo (2007). “Granular damping analysis using an improved discrete element approach”. In: *Journal of Sound and Vibration* 308.1-2, pp. 112–131.

Furley, David J (1967). “Knowledge of atoms and void in Epicureanism”. In:

Gaede, Volker and Oliver Günther (1998). “Multidimensional access methods”. In: *ACM Computing Surveys (CSUR)* 30.2, pp. 170–231.

Gan, JQ, ZY Zhou, and AB Yu (2016). “A GPU-based DEM approach for modelling of particulate systems”. In: *Powder Technology* 301, pp. 1172–1182.

Gelsinger, P. (2004). In: *Intel Developer's Forum*. URL: `https://www.intel.com/pressroom/kits/events/idffall_2004/`.

Ghemawat, Sanjay and Paul Menage (2009). *Tcmalloc: Thread-caching malloc*.

Ghoroghi, Camellia and Tannaz Alinaghi. *An introduction to profiling mechanisms and Linux profilers*.

Glass, Kevin (2005). *Analysis of broad-phase spatial partitioning optimizations in collision detection*. Tech. rep. Technical Report. Grahamstown, South Africa: Rhodes University.

Govender, Nicolin, Daniel N Wilke, and Schalk Kok (2015). "Collision detection of convex polyhedra on the NVIDIA GPU architecture for the discrete element method". In: *Applied Mathematics and Computation* 267, pp. 810–829.

– (2016). "Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture". In: *SoftwareX* 5, pp. 62–66.

Govender, Nicolin et al. (2018). "A study of shape non-uniformity and poly-dispersity in hopper discharge of spherical and polyhedral particle systems using the Blaze-DEM GPU code". In: *Applied Mathematics and Computation* 319, pp. 318–336.

Granular Dynamics International, LLC Engineering Simulation and Scientific Software (2020). *Rocky Discrete Element Method Package*. (Visited on 2012).

Grindon, Christina et al. (2004). "Large-scale molecular dynamics simulation of DNA: implementation and validation of the AMBER98 force field in LAMMPS". In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 362.1820, pp. 1373–1386.

Haftka, Raphael T and Zafer Gürdal (2012). *Elements of structural optimization*. Vol. 11. Springer Science & Business Media.

Hockney, Roger W (1996). *The science of computer benchmarking*. Vol. 2. siam.

Jasak, Hrvoje, Aleksandar Jemcov, Zeljko Tukovic, et al. (2007). "OpenFOAM: A C++ library for complex physics simulations". In: *International workshop on coupled methods in numerical dynamics*. Vol. 1000. IUC Dubrovnik, Croatia, pp. 1–20.

Jiménez, Pablo, Federico Thomas, and Carme Torras (2001). "3D collision detection: a survey". In: *Computers & Graphics* 25.2, pp. 269–285.

Johansson, Robert, Henrik Thunman, and Bo Leckner (2007). “Influence of intraparticle gradients in modeling of fixed bed combustion”. In: *Combustion and Flame* 149.1-2, pp. 49–62.

June 2020 top500 poster. `https://www.top500.org/lists/top500/2020/06/`. Accessed: 2020-08-30.

Kabore, B et al. (2018). “Multi-scale modelling of snow mechanics”. In: *41st Solid Mechanics Conference (SOLMECH 2018)*. Warsaw, Poland.

Kadar, Ali Hussain (2015). “Modelling turbulent non-premixed combustion in industrial furnaces”. PhD thesis.

Kasai, Mio et al. (2009). “LiDAR-derived DEM evaluation of deep-seated landslides in a steep and rocky region of Japan”. In: *Geomorphology* 113.1-2, pp. 57–69.

Ketterhagen, William R, Mary T am Ende, and Bruno C Hancock (2009). “Process modeling in the pharmaceutical industry using the discrete element method”. In: *Journal of pharmaceutical sciences* 98.2, pp. 442–470.

Kloss, Christoph et al. (2012). “Models, algorithms and validation for opensource DEM and CFD–DEM”. In: *Progress in Computational Fluid Dynamics, an International Journal* 12.2-3, pp. 140–152.

Kockara, Sinan et al. (2007). “Collision detection: A survey”. In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, pp. 4046–4051.

Kozicki, Jan and Frederic V Donze (2009). “Yade-open DEM: an open-source software using a discrete element method to simulate granular material”. In: *Engineering Computations* 26.7, pp. 786–805.

Krijgsman, Dinant, Vitaliy Ogarko, and Stefan Luding (2014). “Optimal parameters for a hierarchical grid data structure for contact detection in arbitrarily polydisperse particle systems”. In: *Computational particle mechanics* 1.3, pp. 357–372.

Kroiss, Ryan Robert (2013). “Collision detection using hierarchical grid spatial partitioning on the GPU”. In: *ProQuest Dissertations and Theses, University of Colorado at Boulder* 45.

Kurz, D, U Schnell, and G Scheffknecht (2012). “CFD simulation of wood chip combustion on a grate using an Euler–Euler approach”. In: *Combustion Theory and Modelling* 16.2, pp. 251–273.

Lagrange, Joseph Louis de (1853). *Mécanique analytique*. Vol. 1. Mallet-Bachelier.

Li, Wan-Qing et al. (2010). “Comparison research on the neighbor list algorithms: Verlet table and linked-cell”. In: *Computer Physics Communications* 181.10, pp. 1682–1686.

Lilja, David J (2005). *Measuring computer performance: a practitioner's guide*. Cambridge university press.

Lin, Ming C (1997). “Fast and accurate collision detection for virtual environments”. In: *Scientific Visualization Conference, 1997*. IEEE, pp. 171–171.

Liu, Chun and NOEL J WALkINGTON (2001). “An Eulerian description of fluids containing visco-elastic particles”. In: *Archive for rational mechanics and analysis* 159.3, pp. 229–252.

Liu, Guang-Yu et al. (2020). “Study on the particle breakage of ballast based on a GPU accelerated discrete element method”. In: *Geoscience Frontiers* 11.2, pp. 461–471.

Liyan, Sun et al. (2013). “Simulation of motion of particles in reciprocating grates using DEM”. In: *Powder technology* 246, pp. 218–228.

Lubbe, Retief et al. (2020). “Analysis of parallel spatial partitioning algorithms for GPU based DEM”. In: *Computers and Geotechnics* 125, p. 103708.

LUPI, ALESSIO (2017a). “Numerical Modelling of Biomass Combustion on a Reciprocating Grate: Coupling of Computational Fluid Dynamics and Discrete Element Method”. In:

– (2017b). “Numerical Modelling of Biomass Combustion on a Reciprocating Grate: Coupling of Computational Fluid Dynamics and Discrete Element Method”. In:

Mahmoudi, Amir Houshang et al. (2015). “An experimental and numerical study of wood combustion in a fixed bed using Euler–Lagrange approach (XDEM)”. In: *Fuel* 150, pp. 573–582.

Mahmoudi, Amir Houshang et al. (2016a). “Modeling of the biomass combustion on a forward acting grate using XDEM”. In: *Chemical Engineering Science* 142, pp. 32–41.

– (2016b). “Modeling of the biomass combustion on a forward acting grate using XDEM”. In: *Chemical engineering science* 142, pp. 32–41.

Mahmoudi, Amir Houshang et al. (2016c). “Numerical modeling of self-heating and self-ignition in a packed-bed of biomass using {XDEM}”. In: *Combustion and Flame* 163, pp. 358–369. ISSN: 0010-2180. DOI: `10.1016/j.combustflame.`

2015.10.010. URL: `http://www.sciencedirect.com/science/article/pii/S0010218015003582`.

Mainassara Checkaraou, Abdoul Wahid et al. “Local Verlet buffer approach for broad-phase interaction detection in Discrete Element Method”. Submitted.

Maknickas, Algirdas et al. (2006). “Parallel DEM software for simulation of granular media”. In: *Informatica* 17.2, pp. 207–224.

Mattson, William and Betsy M Rice (1999). “Near-neighbor calculations using a modified cell-linked list method”. In: *Computer Physics Communications* 119.2-3, pp. 135–148.

Michael, M., F. Nicot, and B. Peters (2013). “Discrete Element Modeling of Inter-Granular Bonds between Snow Grains”. In: *Partec2013 Accepted Abstract*. Nuremberg, Germany.

Michael, M. and B. Peters (2013). “3D DEM – FEM Coupling to Analyse the Tractive Performance of Different Tire Treads in Soil”. In: *Coupled2013 Accepted Abstract*. Ibiza, Spain.

Mirtich, Brian Vincent (1996). *Impulse-based dynamic simulation of rigid body systems*. University of California, Berkeley.

Moore, Gordon E et al. (1965). *Cramming more components onto integrated circuits*.

Munjiza, Antonio, Jens H Walther, and Ivo F Sbalzarini (2009). “Large-scale parallel discrete element simulations of granular flow”. In: *Engineering Computations*.

Mytkowicz, Todd et al. (2010). “Evaluating the accuracy of Java profilers”. In: *ACM Sigplan Notices* 45.6, pp. 187–197.

Nambu, Yoichiro and Giovanni Jona-Lasinio (1961). “Dynamical model of elementary particles based on an analogy with superconductivity. I”. In: *Physical review* 122.1, p. 345.

Noske, Andrew (2004). “Efficient Algorithms for Molecular Dynamics Simulations and Other Dynamic Spatial Join Queries”. PhD thesis. Ph. D. Dissertation. http://www. andrewnoske. com/professional/publications …

Pabst, Simon, Artur Koch, and Wolfgang Straßer (2010). “Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library, pp. 1605–1612.

Páll, Szilárd and Berk Hess (2013). “A flexible algorithm for calculating pair interactions on SIMD architectures”. In: *Computer Physics Communications* 184.12, pp. 2641–2650.

Pan, Jia, Sachin Chitta, and Dinesh Manocha (2012). “FCL: A general purpose library for collision and proximity queries”. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3859–3866.

Patronelli, Stefania et al. (2017). “Experimental and numerical investigation of a small-scale fixed-bed biomass boiler”. In: *Chemical Engineering Transactions* 57, pp. 187–192.

Peters, B. and G. Pozzetti (2017). “Flow characteristics of metallic powder grains for additive manufacturing”. en. In: *EPJ Web of Conferences* 13001, p. 140. URL: `http://hdl.handle.net/10993/31734`.

Peters, Bernhard (2013). “The extended discrete element method (XDEM) for multi-physics applications”. In: *Scholarly Journal of Engineering Research*.

Peters, Bernhard and Algis Džiugys (2002). “Numerical simulation of the motion of granular material using object-oriented techniques”. In: *Computer methods in applied mechanics and engineering* 191.17-18, pp. 1983–2007.

Peters, Bernhard et al. (2005). “An approach to qualify the intensity of mixing on a forward acting grate”. In: *Chemical Engineering Science* 60.6, pp. 1649–1659.

Peters, Bernhard et al. (2015). “A discrete/continuous numerical approach to multi-physics”. In: *IFAC-PapersOnLine* 48.1, pp. 645–650.

Peters, Bernhard et al. (2019). “XDEM multi-physics and multi-scale simulation technology: Review of DEM–CFD coupling, methodology and engineering applications”. In: *Particuology* 44, pp. 176–193.

Plimpton, Steve, Paul Crozier, and Aidan Thompson (2007). “LAMMPS-large-scale atomic/molecular massively parallel simulator”. In: *Sandia National Laboratories* 18, pp. 43–43.

Pozzetti, Gabriele and Bernhard Peters (2018). “A multiscale DEM-VOF method for the simulation of three-phase flows”. In: *International Journal of Multiphase Flow* 99, pp. 186–204.

Ransing, RS et al. (2000). “Powder compaction modelling via the discrete and finite element method”. In: *Materials & Design* 21.4, pp. 263–269.

Rettenmaier, Daniel et al. (2019). “Load balanced 2D and 3D adaptive mesh refinement in OpenFOAM”. In: *SoftwareX* 10, p. 100317.

Richard, Patrick et al. (2005). “Slow relaxation and compaction of granular systems”. In: *Nature materials* 4.2, pp. 121–128.

Ritter, Jack (1990). “An efficient bounding sphere”. In: *Graphics gems* 1, pp. 301–303.

Rousset, Alban, Xavier Besseron, and Bernhard Peters (2017). “PARALLELIZING XDEM: LOAD-BALANCING POLICIES AND EFFICIENCY, A STUDY”. In:

Rousset, Alban et al. (2017). “Comparing Broad-Phase Interaction Detection Algorithms for Multiphysics DEM Applications”. In: *AIP Conference Proceedings ICNAAM 2017*. American Institute of Physics.

Rousset, Alban et al. (2018). “Comparing broad-phase interaction detection algorithms for multiphysics DEM applications”. In: *AIP Conference Proceedings*. Vol. 1978. 1. AIP Publishing LLC, p. 270007.

S., Godehard and Vladimir S. (2006). “Optimization of neighbor list techniques in liquid matter simulations”. In: *Journal of Molecular Liquids* 125.2-3, pp. 197–203.

Samiei, K and B Peters (2010). “The discrete particle method (DPM), an advanced numerical simulation tool for particulate applications”. In: *Proc. ECCM 2010 IV European Conference on Computational Mechanics, Paris, France*.

Samiei, Kasra and Bernhard Peters (2013). “Experimental and numerical investigation into the residence time distribution of granular particles on forward and reverse acting grates”. In: *Chemical engineering science* 87, pp. 234–245.

Schäling, Boris (2011). *The Boost C++ libraries*. Boris Schäling.

Schaller, Robert R (1997). “Moore’s law: past, present and future”. In: *IEEE spectrum* 34.6, pp. 52–59.

Schneider, Philip and David H Eberly (2002). *Geometric tools for computer graphics*. Elsevier.

Schreiberx (2020). *Bounding volume hierarchy — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Bounding%20volume%20hierarchy&oldid=921578869`. [Online; accessed 18-June-2020].

Scott, Le Grand (2020). *GPU Gems 3: Chapter 32. Broad-Phase Collision Detection with CUDA*. URL: `https://developer.nvidia.com/gpugems/gpugems3/part-v-`

physics-simulation/chapter-32-broad-phase-collision-detection-cuda (visited on 10/09/2020).

Simsek, E et al. (2009). "Numerical simulation of grate firing systems using a coupled CFD/discrete element method (DEM)". In: *Powder technology* 193.3, pp. 266–273.

Smith, James E. (1988). "Characterizing computer performance with a single number". In: *Communications of the ACM* 31.10, pp. 1202–1206.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*, pp. 2951–2959.

Spivey, J Michael (2004). "Fast, accurate call graph profiling". In: *Software: Practice and Experience* 34.3, pp. 249–264.

Stengel, Holger et al. (2015). "Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model". In: *Proceedings of the 29th ACM on International Conference on Supercomputing*, pp. 207–216.

Stepanov, Alexander and Meng Lee (1995). *The standard template library*. Vol. 1501. Hewlett Packard Laboratories 1501 Page Mill Road, Palo Alto, CA 94304.

Stewart, David B (2001). "Measuring execution time and real-time performance". In: *Embedded Systems Conference (ESC)*. Vol. 141.

Stewart, Graeme and Walter Lampl (Oct. 2017). "How to review 4 million lines of ATLAS code". In: *Journal of Physics: Conference Series* 898, p. 072013. DOI: 10.1088/1742-6596/898/7/072013.

Sudbrock, Florian et al. (2011). "Discrete element analysis of experiments on mixing and stoking of monodisperse spheres on a grate". In: *Powder technology* 208.1, pp. 111–120.

Sun, Liyan et al. (2015). "Prediction of configurational and granular temperatures of particles using DEM in reciprocating grates". In: *Powder Technology* 269, pp. 495–504.

Sun, Xian-He and John L Gustafson (1991). "Toward a better parallel performance metric". In: *Parallel Computing* 17.10-11, pp. 1093–1109.

Sutter, Herb (2005). "The free lunch is over: A fundamental turn toward concurrency in software". In: *Dr. Dobb's journal* 30.3, pp. 202–210.

Tikir, Mustafa M et al. (2007). “A genetic algorithms approach to modeling the performance of memory-bound computations”. In: *SC’07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. IEEE, pp. 1–12.

Tracy, Daniel J, Samuel R Buss, and Bryan M Woods (2009). “Efficient large-scale sweep and prune methods with AABB insertion and removal”. In: *2009 IEEE Virtual Reality Conference*. IEEE, pp. 191–198.

Truong, Nghia, Sreekanth Arikatla, and Andinet Enquobahrie (2019). *Octree-based Collision Detection in iMSTK*. URL: `https://blog.kitware.com/octree-collision-imstk/` (visited on 06/09/2020).

Ulrich, Thatcher (2000). “Loose octrees”. In: *Game programming gems* 1, pp. 434–442.

Varrette, S. et al. (2014). “Management of an Academic HPC Cluster: The UL Experience”. In: *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*. Bologna, Italy: IEEE, pp. 959–967.

Verlet, Loup (1967). “Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules”. In: *Physical review* 159.1, p. 98.

Wadleigh, Kevin R and Isom L Crawford (2000). *Software optimization for high-performance computing*. Prentice Hall Professional.

Weber, Lukas M et al. (2019). “Essential guidelines for computational method benchmarking”. In: *Genome biology* 20.1, p. 125.

Weinhart, Thomas et al. (2016). “Influence of coarse-graining parameters on the analysis of DEM simulations of silo flow”. In: *Powder technology* 293, pp. 138–148.

Weinhart, Thomas et al. (2020). “Fast, flexible particle simulations—An introduction to MercuryDPM”. In: *Computer physics communications* 249, p. 107129.

Welling, Ulrich and Guido Germano (2011). “Efficiency of linked cell algorithms”. In: *Computer Physics Communications* 182.3, pp. 611–615.

Wikipedia contributors (2019). *Enel Green Power — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Enel_Green_Power&oldid=912661233`. [Online; accessed 3-September-2019].

Wilke, Daniel N et al. (2016). “Computing with non-convex Polyhedra on the GPU”. In: *International Conference on Discrete Element Methods*. Springer, pp. 1371–1377.

Williams, John R and Alex P Pentland (1992). "Superquadrics and modal dynamics for discrete elements in interactive design". In: *Engineering Computations* 9.2, pp. 115–127.

Williams, Samuel (2009). "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore". In:

Wulf, Wm A and Sally A McKee (1995). "Hitting the memory wall: implications of the obvious". In: *ACM SIGARCH computer architecture news* 23.1, pp. 20–24.

Wurzenberger, Johann C et al. (2002). "Thermal conversion of biomass: Comprehensive reactor and particle modeling". In: *AIChE Journal* 48.10, pp. 2398–2411.

Yang, YB et al. (2004). "Modelling waste combustion in grate furnaces". In: *Process Safety and Environmental Protection* 82.3, pp. 208–222.

Yin, Chungen et al. (2008). "Mathematical modeling and experimental study of biomass combustion in a thermal 108 MW grate-fired boiler". In: *Energy & Fuels* 22.2, pp. 1380–1390.

Yoo, Andy B, Morris A Jette, and Mark Grondona (2003). "Slurm: Simple linux utility for resource management". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, pp. 44–60.

Zames, G et al. (1981). "Genetic algorithms in search, optimization and machine learning." In: *Information Technology Journal* 3.1, pp. 301–302.

Zhou, Kun et al. (2008). "Real-time kd-tree construction on graphics hardware". In: *ACM Transactions on Graphics (TOG)* 27.5, pp. 1–11.

Zomorodian, Afra and Herbert Edelsbrunner (2000). "Fast Software for Box Intersections". In: *SCG'00*. ACM. ISBN: 1-58113-224-7. DOI: `10.1145/336154.336192`. URL: `http://doi.acm.org/10.1145/336154.336192`.