9-3-2013

Parallel simulation of particle dynamics with application to micropolar peridynamic lattice modeling of reinforced concrete Structures

Hossein Honarvar Gheitanbaf

Hossein Honarvar Gheitanbaf
Candidate

Civil Engineering
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Walter H. Gerstle , Chairperson

Dr. Susan R. Atlas

Dr. Stewart A. Silling

Dr. Arup K. Maji

PARALLEL SIMULATION OF PARTICLE DYNAMICS WITH APPLICATION TO MICROPOLAR PERIDYNAMIC LATTICE MODELING OF REINFORCED CONCRETE STRUCTURES

by

HOSSEIN HONARVAR GHEITANBAF

B.S., Civil Engineering, University of Tehran, Iran, 2011

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Civil Engineering

The University of New Mexico
Albuquerque, New Mexico

May, 2013

To my family, for their love, support, and encouragement

ACKNOWLEDGMENTS

I owe my utmost gratitude to Prof. Walter H. Gerstle, my advisor and committee chair, for his guidance, attention, and understanding during my research.

I heartily acknowledge my co-advisor, Prof. Susan R. Atlas, of Department of Physics and Astronomy, for her support, patience and concern about my research.

It is my pleasure to thank Dr. Stewart A. Silling, of Sandia National Laboratories, whose kind concern about my research is acknowledged.

I am thankful to Prof. Arup K. Maji, member of my committee, for his comments and suggestions.

Also, I appreciate the Center for Advanced Research Computing for providing facilities and help in harnessing the supercomputers.

Lastly, I would like to express my gratitude to my family, friends, and all those who supported me in any aspect to complete my thesis.

Parallel Simulation of Particle Dynamics with Application to

Micropolar Peridynamic Lattice Modeling of

Reinforced Concrete Structures

by

Hossein Honarvar Gheitanbaf

B.S., Civil Engineering, University of Tehran, 2011

M.S., Civil Engineering, University of New Mexico, 2013

ABSTRACT

Particle dynamics simulations are used widely in various disciplines such as physics, engineering, and biology. To study complex systems consisting of a large number of particles, an efficient parallel particle dynamics code is necessary. Several such codes exist but each has been designed with specific applications in mind. For example, LAMMPS is designed mainly for atomistic modeling, GROMACS for biophysics applications, and EMU for peridynamic studies. With the goal of having a general purpose parallel particle dynamics code, in 2011, two UNM research groups collaborated on the redevelopment and generalization of the pdQ molecular dynamics code to jointly accommodate both molecular dynamics and peridynamics [Sakhavand 2011]. However, pdQ remained domain-specific, using an "#ifdef" coding style to select alternative molecular dynamics and peridynamic routes through the code at compile time. In addition, due to the data structures used, the implementation of "particle shuffling", in which particles' neighborhoods change, was challenging in this initial version of the

code. Finally, an extensive review of the message passing algorithm used in pdQ [Sakhavand 2011] revealed inefficiencies due to the sending of unnecessary messages.

In this thesis, we describe the re-architecting of pdQ as pdQ2. pdQ2 is completely non-domain-specific in that user files are clearly separated from non-user files and no #ifdefs exist in the code. Thus, it operates as a particle simulation *engine* that is capable of executing any parallel particle dynamics model. As in the original pdQ, users can customize their own physical models without having to deal with complexities such as parallelization, but the ease of extensibility has been significantly improved. This has greatly facilitated the implementation of particle shuffling, for example. Finally, the message passing is implemented by introducing the concepts of "core" and "skin" which define the central processing cube or "procCube". It is shown that pdQ2 is about four times as fast as pdQ using parallel supercomputers.

The particle dynamics model of particular interest in this work is *peridynamics*. Peridynamics was proposed by S.A. Silling to overcome deficiencies in the continuum mechanics formulation for modeling discontinuities in a material at different scales from micro to macro [Silling 1998]. Thus, it has the potential to be used for the engineering of reinforced concrete structures which show many discontinuities prior to failure. Gerstle *et al.* extended peridynamics to include particle rotations, terming their model the "micropolar peridynamic model" [Gerstle *et al.* 2007b]. However, both the original peridynamic and micropolar peridynamic models require that *ad hoc* discretization decisions be made to implement them computationally, and they do not explicitly relinquish the continuous topology describing the reference geometry [Gerstle *et al.* 2012].

In this thesis, we discard the continuum mechanics paradigm completely, and model reinforced concrete by introducing the "micropolar peridynamic lattice model (MPLM)". The MPLM models a structure as a close-packed particle lattice. In the MPLM, rather than viewing the structure as collection of truss or beam elements (as with traditional lattice models), the model is viewed as collection of particle masses (as with peridynamic models). The MPLM uses a finite number of equally-spaced interacting particles of finite mass. Thus, it does not need any *ad hoc* discretization and it is more straightforward to implement computationally. Also, the MPLM is conceptually simpler than both the lattice and peridynamic models [Gerstle *et al.* 2012]. After defining the MPLM, its application to reinforced concrete structures is investigated through several examples using pdQ2.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In this thesis, we describe a novel architecture for parallel particle dynamics which we refer to "particle dynamics Quickly (pdQ2)". In the second part of this thesis, pdQ2 is used to implement the micropolar peridynamic lattice model (MPLM) for simulation of reinforced concrete structures.

1.1. Motivation

Particle dynamics simulation methods such as peridynamics, molecular dynamics, and discrete element methods are widely used by scientists and engineers. The availability of a general-purpose and efficient parallel particle dynamics code is thus an important tool for many disciplines. Various parallel particle dynamics codes have been designed in recent years, such as LAMMPS [http://lammps.sandia.gov], GROMACS [Lindahl *et al.* 2001], and EMU [http://sandia.gov/emu/emu.htm], each with specific applications in mind. LAMMPS is designed mainly for molecular and atomistic modeling, GROMACS for biophysics, EMU for peridynamic studies, and so forth. Therefore, if users wish to use these codes, they must be familiar with domain-specific concepts. This is unnecessarily complicated and cumbersome for users in emerging disciplines.

In the second part of this thesis, we present results of the computational modeling of reinforced concrete structures using peridynamics within pdQ2. Concrete is a quasi-brittle material, and cracking is a challenging feature to model in the simulation of reinforced concrete structures. Reinforced concrete exhibits many discontinuities even

before failure mechanisms develop. A failure mechanism in a reinforced concrete column subject to earthquake loading is shown in Fig. 1.1.

Figure 1.1 – Failure mechanism at the bottom of a concrete column subject to earthquake loading (from [Jennings 1971]).

To this day, *ad hoc* approximate methods are used to model reinforced concrete structures. But these approximate methods are not general or simple enough to be used for many complex problems. Thus, current engineering designs are usually overly conservative in order to make up for the lack of modeling capability. Economic savings could be realized by more intelligent models of reinforced concrete structures.

With ongoing advances in computational power, it makes sense to use computational methods to model fracture problems. Continuum mechanics and finite elements methods work well for continuum problems, but they fail to model cracks efficaciously.

The *smeared crack* and the *discrete crack* approaches are among computational models based on fracture mechanics using continuum mechanics-based finite element methods. In the smeared crack models, the cracks are represented through changes of the material stress-strain constitutive equations instead of changes in the geometry as with discrete models. Sensitivity of the results to the finite element mesh is the major deficiency in smeared crack models [Nguyen *et al.* 2005].

In discrete crack models, fracture mechanics theories are used to predict the trajectories of discrete cracks. The problem geometry and the corresponding finite element mesh is incrementally altered as cracks propagate [Cusatis *et al.* 2006]. For 3D problems, simulation of discrete cracks is often not possible due to the complexity of the required geometry and mesh. Furthermore, the assumptions of fracture mechanics are insufficiently general to capture the behavior of reinforced concrete structures.

Molecular dynamics simulation is another computational approach that can be used for fracture modeling. With the most powerful supercomputers, molecular dynamics is an effective method to model fracture at the atomistic and nano scales. But applying molecular dynamics to macro-scale structures like reinforced concrete structures is not yet feasible due to huge computational requirements. For instance, a concrete beam with length of 4 m, depth of 0.4 m, and width of 0.3 m, has about 10^{30} silicon atoms [O'Mara *et al.* 1990]. Today's most powerful supercomputers can execute molecular dynamics simulations up to several billions (10^{9}) of particles [Kadau *et al.* 2006].

Peridynamics is a formulation that can model discontinuities more realistically than other existing methods. Peridynamics can be implemented as a particle dynamics

method which is applicable to macro-scale structures, unlike molecular dynamics. Therefore, we apply peridynamic theory to simulate reinforced concrete structures in this work.

1.2. Background

In light of the issues in attempting to use existing parallel particle dynamics codes, developing a general-purpose code that can perform explicit particle dynamics simulations is a worthy goal. Different users would be able to use such a code without the need to know about many irrelevant concepts from other science and engineering domains. To achieve this purpose, in 2009, S.R. Atlas, W.H. Gerstle, N. Sakhavand, and V. Janardhanam from the Physics and Astronomy and Civil Engineering departments at the University of New Mexico embarked on designing a parallel particle dynamics code called "particle dynamics Quantum (pdQ)". This code [Atlas 1999] was based on an earlier object-oriented parallel molecular dynamics code [Atlas *et al.* 1996], both originally developed by S.R. Atlas at the University of New Mexico. While pdQ was designed to accommodate both peridynamics and molecular dynamics in a single code, it remained domain-specific, with #ifdefs used to select alternative molecular dynamics or peridynamic routes through the code at compile time. Also, the data structures used in the code were sufficiently complex that it became obvious that a major re-write was necessary if "particle shuffle" was to be implemented. Particle shuffle is necessary when the particles undergo large relative displacements, which is particularly true for molecular dynamics simulations of liquids, for instance.

One of the most significant achievements of the current thesis is the development of a truly domain-independent particle dynamics code. We still call the code pdQ but now the acronym stands for “particle dynamics Quickly”. To distinguish the two codes in this thesis, we refer to Sakhavand’s version of pdQ as “pdQ”, and the new version as “pdQ2”. The new version, pdQ2, is truly independent of any physical discipline, and implements “particle shuffle”, as described in Chapter 3. In addition, due to a redesign of the message passing algorithm, pdQ2 is much more computationally efficient than pdQ, as shown in Chapter 3.

On the reinforced concrete modeling side, S.A. Silling, from Sandia National Laboratories, in order to overcome the deficiencies for solving the discontinuum problems, proposed the peridynamic model [Silling 1998]. Peridynamics is a reformulation of continuum mechanics that allows discontinuities to develop and propagate naturally in a material. In this reformulation, the governing equation of motion is an integral equation, in contrast to continuum mechanics, which is based on a differential equation that fails at displacement discontinuities. However, peridynamics does not explicitly completely relinquish the continuum mechanics paradigm [Gerstle *et al.* 2012].

Peridynamics is conceptually similar to molecular dynamics but it can be used at varying scales from micro to macro. It models the structure as interacting material particles analogous to molecular dynamics, which models molecules of materials as interacting atoms. Thus, peridynamics is applicable to civil engineering structures such as reinforced concrete bridges. Peridynamics is computationally intensive when applied to

large engineering structures and it requires that further *ad hoc* discretization decisions be made in order to implement it computationally.

In this thesis, a new model called the "micropolar peridynamic lattice model (MPLM)" is presented in Chapter 4. With the MPLM model, the concrete is modeled as a close-packed particle lattice, which discards the continuum mechanics paradigm completely. The MPLM is a good model for concrete which is inherently discontinuous, and it is more straightforward to implement computationally, because fewer *ad hoc* discretization decisions need to be made.

1.3. Objectives

The objectives of this thesis are twofold. First, with the goals of simplicity, efficiency, and extensibility, pdQ has been redesigned and rewritten (called pdQ2 in this thesis). As with pdQ, pdQ2 is designed with the philosophy of hiding computational complexities such as parallelization from the user. However, in contrast to pdQ, which "inlined" physical models of particle interactions, pdQ2 utilizes programmable user files for implementation of physical models. Thus, pdQ2 is an engine that can run any explicit parallel particle dynamics simulation, and users can easily implement arbitrary physical models with any degree of complexity.

Second, to address the problems with existing peridynamic and other computational models, the "micropolar peridynamic lattice model (MPLM)" is implemented. With the MPLM, the number of computations is decreased with respect to the original peridynamic model, and it is conceptually simpler than existing continuum models [Tuniki 2012].

1.4. Scope

This thesis includes five chapters. Chapter 2 provides a literature review of existing particle dynamics codes and the peridynamic models for reinforced concrete structures. Chapter 3 describes the design and performance analysis of pdQ2. Chapter 4 describes the MPLM and provides several examples to demonstrate the method using pdQ2. In Chapter 5, conclusions and future work are suggested. Finally, a concise user manual for pdQ2 is provided in the Appendix.

2. LITERATURE REVIEW

This chapter has three main sections. In the first section, several computational algorithms for parallel particle dynamics simulation are studied. In the second section, several existing parallel particle dynamics codes are investigated. The advantages and disadvantages of these codes are explained. In the third section, the peridynamic theory is described and variations of this model are explained.

2.1. Computational algorithms for parallel particle dynamics

Particle dynamics refers to a system of particles interacting with each other in a specific physical domain. The basic steps of a particle simulation are particles' definition, domain decomposition, force interactions, integration, and particle shuffling.

In real-life particle dynamics problems, millions or billions of particles interact with each other. In order to solve such realistic problems, exploiting the capabilities of parallel computers is necessary. To use parallel computers efficiently, developing appropriate algorithms is important. Several parallel algorithms exist mainly for molecular dynamics simulation, but they may also be adopted for use in other particle dynamics simulations in other domains, such as astrophysical simulation or peridynamics. *Atom decomposition*, *force decomposition*, and *spatial decomposition* are the three main parallel algorithms that were pioneered in the early days of parallel computing.

In the atom decomposition method, an equal number of particles are distributed among processors regardless of their positions. Then, the new positions are computed on

each processor after the all-to-all communication is done among all the processors [Bruck *et al.* 1994].

In the force decomposition method, a subset of pairwise forces is assigned to each processor [Hendrickson and Plimpton 1992].

In the spatial decomposition method, the entire geometrical domain is decomposed across several subdomains containing particles. Each subdomain is allocated to a specific processor and consequently the particles are assigned to those processors. The associated processor does the computations for its particles. But the particles residing on processor boundaries necessarily interact with the particles residing on adjacent processors. Therefore, to fulfill this requirement, information about particles residing on the boundaries need to be exchanged between processors [Finchman 1987]. Like pdQ, pdQ2 uses a spatial decomposition algorithm (see Chapter 3).

The atom and force decomposition methods are load-balanced. In other words, they divide the computations between processors equally. For these methods, the inter-processor communication scheme is global, in contrast to the spatial decomposition algorithm which has a local communication scheme [Plimpton 1995].

Choosing a parallel algorithm is problem-dependent. It can depend upon the number of particles in the processor or the speed of the processor itself. For example, if the number of particles per processor is high, the spatial decomposition algorithm is more efficient than others [Brown and Miagret 1999]. Due to increases in the amount of RAM per core in modern supercomputers, this is the regime considered in pdQ and pdQ2.

In the following subsections, several techniques for implementation of parallel particle dynamics algorithms are reviewed, including Plimpton's method of message [Plimpton 1995], which is implemented in pdQ2.

2.1.1. The multi-cell method

The multi-cell method was first used by Beazley and Lomdahl for molecular dynamics simulations [Beazley and Lomdahl 1994]. In this method, the geometry of the problem is divided into cuboids that are assigned to a processor. Each of these cuboids is subdivided into cells. Thus, each processor has a certain number of cells, and particles are located within the cells. A particle interacts with other particles that are in its neighborhood (material horizon) and it does not interact with the particles that are beyond the material horizon. By introducing the concept of cell, particles can quickly identify the neighbors with which they interact. By choosing the cell dimension to be somewhat larger than the material horizon, interaction between particles within the material horizon is guaranteed. A particle within a specific cell and processor can interact with other particles in the same cell and processor, or other particles in different cells but the same processor, or other particles in different cells and processors. If the particles need to interact with the particles on other processors, communication between processors is necessary. Like pdQ, pdQ2 uses the multi-cell method of Beazley and Lomdahl combined with the concepts of walls and procCubes, but we introduce the notions of "core" and "skin", implemented using FORTRAN arrays as described in Chapter 3, in order to significantly reduce inter-processor communication.

2.1.2. Plimpton's method of processor communication

Parallel simulation of particle dynamics requires communication between processors. Information is sent and received between processors as messages. This is also called *message passing* between processors.

In the spatial decomposition algorithm, a cuboid needs to obtain information from the twenty-six adjacent cuboids, at most. Thus, each cuboid should receive twenty-six messages per time step. In Fig. 2.1, a cuboid (shown in gray), and its 26 adjacent cuboids are shown.

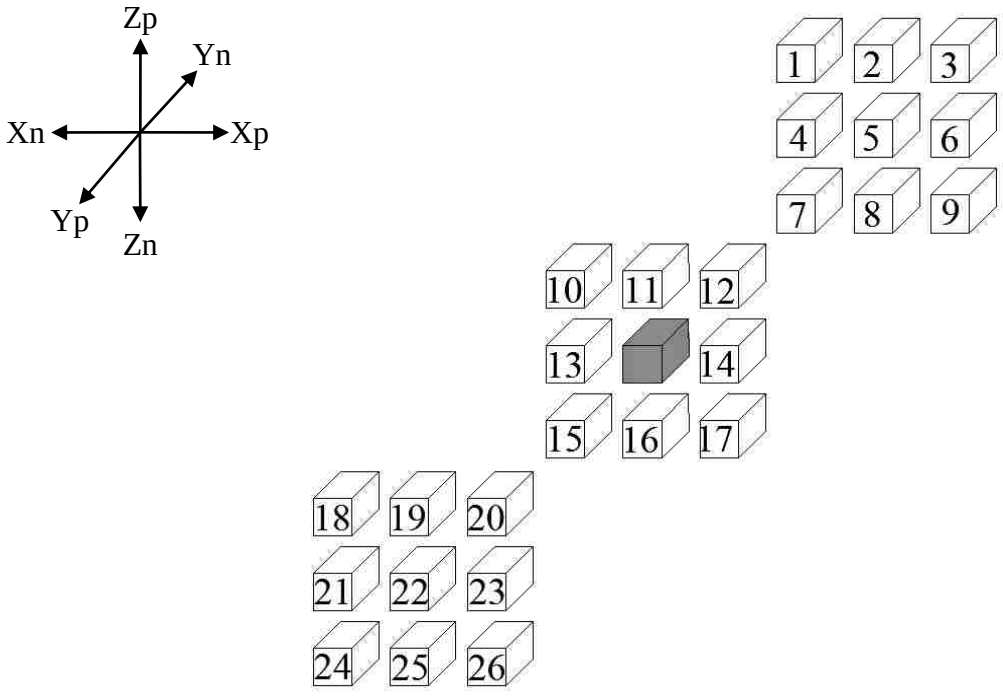
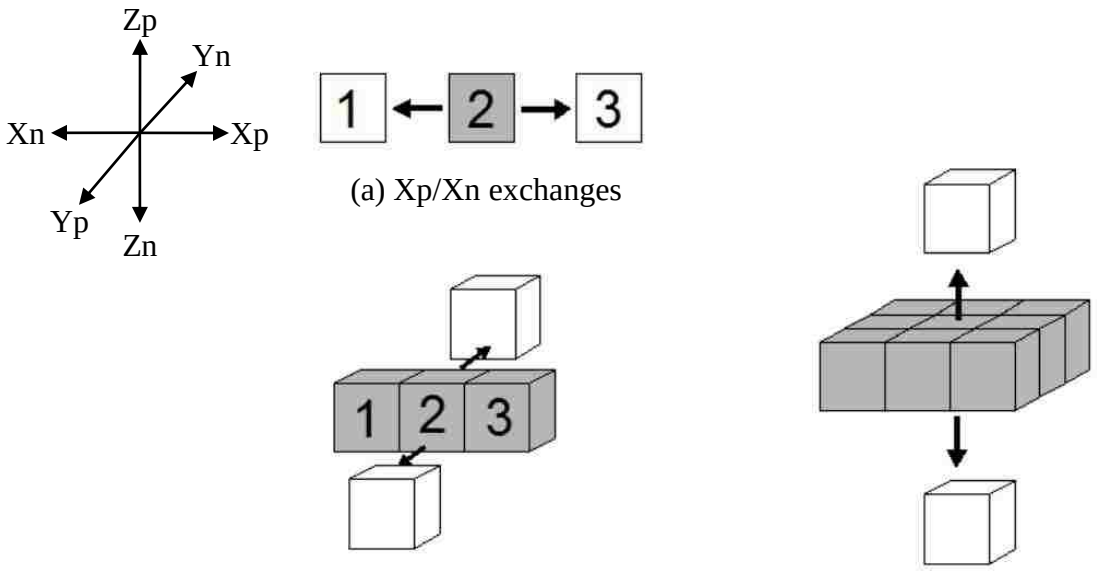


Figure 2.1 – Exploded view of a cuboid (shown in gray) and its 26 adjacent cuboids. "p" and "n" refers to positive and negative directions respectively.

In 1993, Plimpton introduced a novel method for communication between processors in particle dynamics simulation [Plimpton 1995]. With Plimpton's method of

message passing, the number of the messages received by a cuboid reduces to six. In Fig. 2.2, Plimpton's method of message passing is illustrated. In the first step, each processor sends and receives information in the Xp direction simultaneously. For example, processor 2 sends information to processor 3 and also receives the data from processor 1. In the second step, each processor sends and receives the information in the Xn direction concurrently. In the third step, each processor sends and receives the data in the Yp direction; note that this includes information previously passed in the Xp/Xn exchanges. In the fourth step, each processor sends and receives the data in the Yn direction, and this contains information communicated in the Xp/Xn exchanges. In the fifth step, every processor sends and receives information in the Zp direction including data from the Xp/Xn and Yp/Yn exchanges.



(a) Xp/Xn exchanges

(b) Yp/Yn exchanges

(c) Zp/Zn exchanges

Figure 2.2 – Plimpton's method of message passing (after [Plimpton 1995]).

In the sixth step, each processor sends and receives data in the Zn direction, containing information exchanged in the Xp/Xn and Yp/Yn exchanges. With these six steps, each cuboid has now obtained the information from its twenty-six adjacent cuboids. Using FORTRAN arrays, we implement Plimpton's method elegantly, as described in Chapter 3.

2.1.3. Zonal methods

Zonal methods have been used in several molecular dynamics research codes. Zonal methods are generalized spatial decomposition methods for efficient parallelization of range-limited N-body problems [Bowers *et al.* 2006]. Zonal methods are based on the concept of *zones* which are spatial shaped regions. They enable the tailoring of communication between processors and ensure that all near interactions are computed. The main purpose of zonal methods is to reduce the communication time in order to avoid redundant force calculations. Zonal methods can result in optimized scaling for biophysical systems [Bowers *et al.* 2006]. Like pdQ, to improve the efficiency, zonal methods can be implemented in pdQ2 because of using hybrid of cells and procCubes.

In the next section, several existing parallel particle dynamics codes are described.

2.2. Parallel particle dynamics codes

Dynamical simulation of large-scale particle systems is impossible without a powerful parallel computer. Even though computational power has tended to increase by approximately a factor of two every 18 months following Moore's law [Moore 1965], it is still necessary to take advantage of advanced computational tools to model the largest

systems of particles. Thus, it is important to develop a simple, efficacious, and extensible parallel particle dynamics code. In the following subsections, several existing particle dynamics codes are reviewed and evaluated.

2.2.1. SPaSM (Scalable Parallel Short-range Molecular dynamics)

In the early 1990s, D. Beazley and P. Lomdahl collaborated at Los Alamos National Laboratory to develop the SPaSM code. It was primarily designed to simulate the behavior of materials [Zhou *et al.* 1998]. SPaSM uses the multi-cell spatial decomposition method, and it was originally developed for the Thinking Machine CM-5 [Beazley and Lomdahl 1994]. In 2006, SPaSM was implemented on the BlueGene/L architecture to enable large molecular dynamics simulations with billions of particles. It has shown good scaling and performance [Kadau *et al.* 2006]. Recently, the communication data structures in the SPaSM have been rewritten for implementation on the Roadrunner supercomputer, achieving excellent speedup [Germann *et al.* 2009]. However, SPaSM is domain-specific for molecular dynamics models, and is proprietary, and therefore could not be used as a research platform in the present work.

2.2.2. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator)

In 1990, two laboratories (Sandia and LLNL) and three companies (Cray, Bristol-Myers Squibb, Dupont) began to develop a large-scale parallel classical molecular dynamics code called LAMMPS. It was originally developed in FORTRAN 77 and was rewritten in FORTRAN 90 in 2001. In 2004, an open source version for LAMMPS was released in C++ and the code is updated continuously [http://lammps.sandia.gov].

LAMMPS is a classical molecular dynamics code that models particles in a liquid, solid, or gaseous state. It is designed mainly for parallel simulations; however, it can also efficiently run on a single-processor machine. It can run simulations with millions or billions of particles. For computational efficiency, LAMMPS uses neighbor lists to keep track of nearby particles. On parallel machines, it uses the spatial decomposition technique to partition the simulation domain and assign the partitions to processors. LAMMPS also implements GPU coding (CUDA and OpenCL) and OpenMP for further code acceleration.

LAMMPS is able to model diverse force fields and statistical ensembles, and implements diverse constraints, boundary conditions, and integration schemes. Furthermore, a peridynamic module has been added recently to LAMMPS [Parks *et al.* 2008]. However, as LAMMPS is not a pure particle dynamics engine, it cannot be readily adapted to implement other particle dynamics models.

2.2.3. GROMACS (GROningen MAchine for Chemical Simulations)

The GROMACS project was originally designed by the Biophysical Chemistry Department of the University of Groningen as a parallel computer system for molecular dynamics simulations in FORTRAN 77. Later, it was rewritten in the C programming language. Now, it is a free open-source code [Lindahl *et al.* 2001].

GROMACS was primarily designed for simulating biochemical molecules such as proteins, lipids and nucleic acids, although it is now also applicable to non-biological systems such as polymers. It benefits from novel optimization methods such as the automatic generation of inner loops in either C or FORTRAN at compile time. On

parallel machines, it uses MPI communication and a spatial decomposition algorithm. Currently, GROMACS claims to be the fastest algorithm for parallel molecular simulations [Lindahl *et al.* 2001]. Recently, zonal methods were implemented in GROMACS to improve the communication time and to avoid redundant force computations [Hess *et al.* 2008]. As designed, GROMACS is intended principally for biophysical simulations and it appears to be cumbersome to use it for other physical models or domains.

2.2.4. Desmond

Desmond is a software package developed by the D. E. Shaw research group, as a free, open-source code specifically for performing parallel molecular dynamics simulations of biological and chemical systems [Bowers *et al.* 2006]. Thus, it cannot be easily used for other domain applications.

Desmond can compute energies and forces for many standard fixed-charged force fields used in biomolecular simulations [Lindorf-Larsen *et al.* 2010]. It implements an integrated version of the force decomposition and spatial decomposition algorithms. This hybrid of spatial and force decompositions utilizes a spatial decomposition of particles into cuboids, and the creation of a computational object for calculating interactions for every pair of interacting cuboids [Bhatele *et al.* 2008]. Since Desmond is mainly designed for molecular dynamics and it is domain-specific, it appears to be hard to use it for other particle dynamics methods

2.2.5. NAMD (Not (just) Another Molecular Dynamics program)

In 1995, NAMD was introduced as a molecular dynamics program utilizing the CHARM++ parallel programming layer for high performance simulations of biomolecular systems on parallel supercomputers. It was developed at the University of Illinois at Urbana-Champaign [Nelson *et al.* 1996]. It implements an object-oriented design using C++ and is freely available. NAMD is not general enough to be used for arbitrary particle dynamics problems.

NAMD uses spatial decomposition combined with a multithreaded message-driven design to provide a highly scalable program tolerant of communication latency [Nelson *et al.* 1996].

2.2.6. EMU

EMU has been developed mainly for in-house peridynamic simulations at Sandia National Laboratories [http://sandia.gov/emu/emu.htm]. EMU is the first code designed specifically for peridynamic simulations, to predict the deformation and failure of solids. However, EMU is a research code and is not generally available. We found that it was difficult to extend EMU to incorporate micropolar peridynamics and user-specified state-based damage models.

2.2.7. pdQ (parallel dynamics Quantum)

In 2009, a group of faculty and students from the University of New Mexico (S.R. Atlas, W.H. Gerstle, N. Sakhavand, and V. Janardhanam) decided to write a parallel particle dynamics code called pdQ with the purpose of introducing a versatile parallel

code that could be used for both molecular dynamics and peridynamic simulations [Sakhavand 2011]. This code was based on a parallel molecular dynamics code originally developed by S.R. Atlas at the University of New Mexico [Atlas 1999]. pdQ was written for molecular dynamics and peridynamic simulations using #ifdefs to determine a route through the code at compile time depending on whether a molecular dynamics or peridynamic simulation was required. This limited pdQ to these specific particle simulation domains.

In addition, pdQ was written using a number of complex data structures that did not lend themselves to modification or extensibility. For example, "particle shuffle" or the state-based peridynamics could not be simply implemented. Also, the message passing algorithm used in pdQ was inefficient, sending many unnecessary messages between processors [Sakhavand 2011]. In light of these issues, in 2011, we resolved to rewrite pdQ with the purpose of simplicity, efficiency, and enhanced extensibility. The new version, pdQ2, is described in Chapter 3 of this thesis. pdQ2 is completely non-domain-specific and it is an engine that can run any parallel particle dynamics simulation. Also, pdQ2 is sufficiently simple that "particle shuffle" can be easily implemented, for example. Finally, message passing efficiency has been improved by introducing the "core" and "skin" concepts as discussed in Chapter 3 and it is shown that pdQ2 is about four times faster than pdQ for the problems benchmarked in this thesis.

2.3. Peridynamics

In 2000, S.A. Silling from Sandia National Laboratories introduced the peridynamic model that can be used at different scales, from micro to macro and for both

continuous and discontinuous media, to overcome the deficiencies of the continuum mechanics formulation [Silling *et al.* 1998]. In this model, the structure is modeled as particles interacting with surrounding particles via specified force functions. The peridynamic equation of motion is an integral formulation, unlike continuum mechanics, which fails at discontinuities; thus, the peridynamic formulation allows cracks to emerge naturally.

To computationally model continuous structures using peridynamics, further *ad hoc* discretization is needed; however, no mesh generation is required. In the peridynamic model, the material horizon is defined as a spherical continuous region around a particle *i* that interacts with other particles *j* within this region. In Fig. 2.3, two particles *i* and *j* with volumes dV_i and dV_j are shown in a domain R. The equation of motion of the particle *i* (Newton's second law) for the peridynamic model is

$$\int_R \vec{f}_{ij}(\vec{\eta}, \vec{\xi}) dV_j + \vec{b}_i = \rho \ddot{\vec{u}}_i, \tag{2.1}$$

where $\vec{f}_{ij}$ is the pairwise force function between particles *i* and *j*. The pairwise force function depends on the relative position $(\vec{\xi})$ and relative displacement $(\vec{\eta})$ between the two particles *i* and *j*. $\vec{x}$ and $\vec{u}$ are the reference position and displacement fields respectively. $\vec{b}$ is the external force density, in units of force per unit volume. ρ is the mass density.

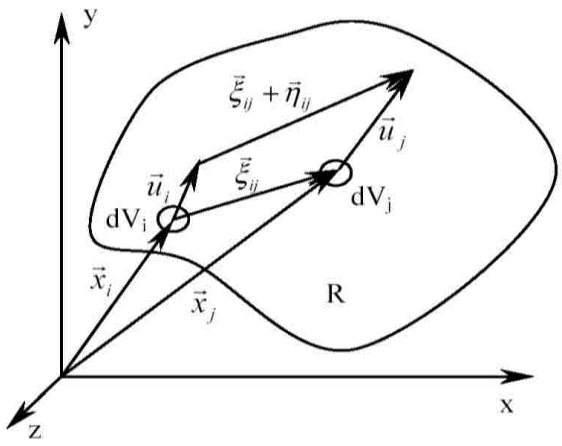


Figure 2.3 – Terminology for the peridynamic model (from [Gerstle *et al.* 2007b]).

In the following subsections, we discuss further variations of the peridynamic model.

2.3.1. Micropolar peridynamic model

In 2005, Gerstle *et al.* extended the peridynamic model by adding moments and rotations to the original peridynamic formulation [Gerstle *et al.* 2007b]. They called their model the "micropolar peridynamic model". The terminology for this model is shown in Fig. 2.4. The governing equations of motion (Newton's second law) for the micropolar peridynamic model for the particle *i* are

$$\int_R \vec{f}_{ij}(\vec{\eta}, \vec{\xi}, \vec{\theta}) dV_j + \vec{b}_i = \rho \ddot{\vec{u}}_i, \tag{2.2}$$

and

$$\int_R \vec{m}_{ij}(\vec{\eta}, \vec{\xi}, \vec{\theta}) dV_j + \vec{c}_i = \gamma \ddot{\vec{\theta}}_i, \tag{2.3}$$

where $\vec{m}_{ij}$ is the pairwise moment function between particles *i* and *j* with units of moment per unit volume squared. $\vec{\theta}$ is the relative rotation between the two particles. $\vec{c}$ is the external moment density, in units of moment per unit volume. γ is the mass moment of inertia per unit volume. All other parameters are the same as in the original peridynamic model.

The micropolar peridynamic model generalizes the peridynamic model to more easily include materials with Poisson's ratio other than $\frac{1}{4}$, and axial or plate structures [Gerstle *et al.* 2007b]. For any central force model, it can be proved that there is such a limitation on Poisson's ratio. In Chapter 4, the micropolar peridynamic model is specialized to the micropolar peridynamic lattice model (MPLM) with the purpose of avoiding *ad hoc* discretization and achieving greater computational efficiency.

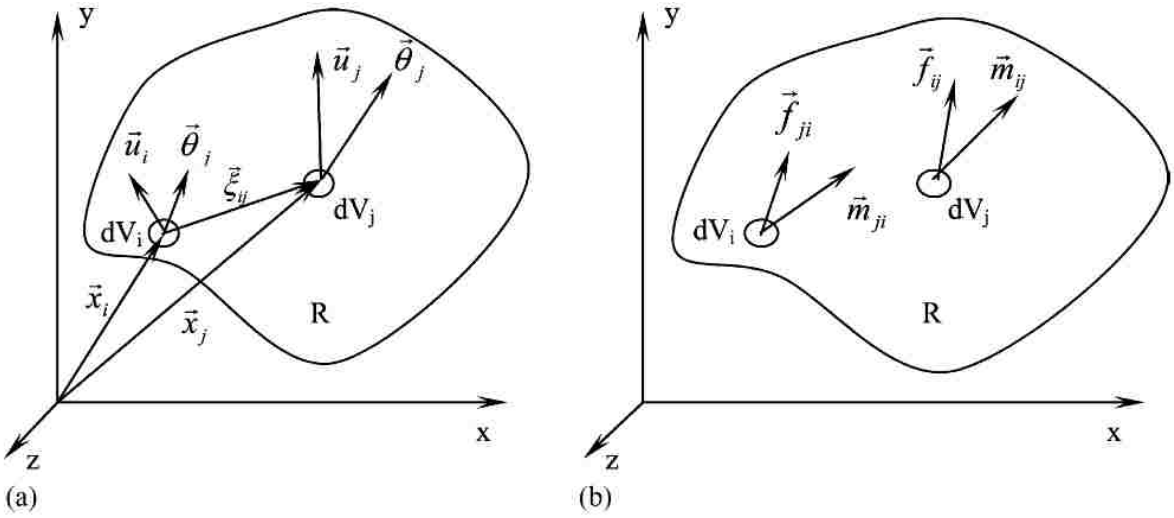


Fig. 2.4 – Terminology for micropolar peridynamic model: (a) kinematics; (b) kinetics (from [Gerstle *et al.* 2007b]).

2.3.2. Peridynamic states

The original peridynamic model included only a "bond-based" model for materials. "Bond-based" means that the pairwise force function between two particles depends only upon the states of these two particles. This is an oversimplification which prevents some material behaviors from being modeled, for example plasticity [Silling *et al.* 2007].

In order to resolve these difficulties, Silling *et al.* introduced the concept of peridynamic states to generalize the original bond-based peridynamic model [Silling *et al.* 2007]. The state-based peridynamic constitutive model is a relationship between the force state and the deformation state. The force state contains the forces within bonds of all lengths and orientations. The deformation state is the deformation field within the material horizon of a particle. The state-based equation of motion is

$$\int_{\mathrm{H}_x} \left\{ \underline{T}[\vec{x}, t] \left\langle \vec{x}' - \vec{x} \right\rangle - \underline{T}[\vec{x}', t] \left\langle \vec{x} - \vec{x}' \right\rangle \right\} dV_{x'} + \vec{b}(\vec{x}, t) = \rho(\vec{x})\ \ddot{\vec{u}}(\vec{x}, t)\ , \tag{2.4}$$

where H_x is a spherical region centered at $\vec{x}$ with the radius of the material horizon. $\underline{T}$ is the force vector state field.

State-based peridynamics is computationally more intensive than bond-based peridynamics. However, state-based peridynamics is capable of modeling broader material behaviors such as plasticity [Silling *et al.* 2007]. As discussed in Chapter 5, pdQ2 is designed so that state-based peridynamics can be easily implemented in pdQ2 later on.

2.3.3. Multi-physical peridynamics

Multi-physical modeling involves the simulation of multiple physical phenomena simultaneously. This multiplicity of physics adds to the complexity of the problem. Gerstle *et al.* has applied the peridynamic model to multi-physical problems [Gerstle *et al.* 2008]. In particular, the peridynamic model has been used to simulate mechanical, thermal, electrical, and atomic diffusion processes which often account for the failure of the integrated circuits (electromigration).

In the peridynamic model for electromigration, the constitutive relations of solid mechanics, heat conduction, electric conduction, and atomic diffusion correspond to integrations over a finite neighborhood of a point [Gerstle *et al.* 2008]. The factors accounted for in the multi-physics constitutive model are fluxes of force, heat energy, electrical charge, and atoms. Peridynamic kernels are used to represent the physical constitutive behavior of these processes.

The peridynamic model of electromigration enables four coupled physical phenomena to be modeled concurrently: mechanical deformation, heat transfer, electrical potential distribution and charge flow, and vacancy diffusion. The conceptual simplicity of the model paves the way for the multi-physical simulation of microchips, enabling electromigration, thermomechanical crack formation, and fatigue crack formation to be analyzed in a systematic manner [Gerstle *et al.* 2008].

2.4. Summary

In this chapter, several parallel particle dynamics codes have been reviewed. As described, they are neither general-purpose nor easy to extend. Thus, it is not straight-

forward for users to implement their own physical models. In this thesis, the main motivation for developing pdQ2 was to further develop pdQ into a general-purpose parallel particle dynamics code that is efficient, easy to use, simple and extensible.

On the peridynamic side, peridynamics and its variations have been described. The original peridynamic model can only model materials having Poisson's ratio $\frac{1}{4}$. Also, it requires that further *ad hoc* discretization decisions be made in order to be implemented computationally. In this thesis, the micropolar peridynamic lattice model is introduced to address the difficulties with the original bond-based peridynamic model for modeling reinforced concrete.

In the next chapter a novel architecture for the pdQ2 parallel particle dynamics code is presented.

3. NOVEL ARCHITECTURE FOR A PARALLEL PARTICLE DYNAMICS CODE

3.1. Introduction

The development of parallel particle dynamics simulation algorithms has been an active topic of research in recent years. Particle dynamics simulation codes have been developed in various disciplines, such as molecular dynamics, peridynamics, discrete element methods, and so forth. To solve realistic problems, a large number of particles and time steps is usually required. Single-processor machines are too slow or do not have sufficient memory; thus it is advantageous to use parallel-processor machines to decrease simulation times and increase the number of particles and time steps that can be simulated.

While parallelization has resulted in a considerable reduction in simulation time, to date, existing codes typically couple their parallelization approaches to domain-specific features. Therefore, the design of a general-purpose, simple, efficient and extensible parallelizable particle dynamics code is an important issue for contemporary particle dynamics research.

Most parallel particle dynamics codes in existence have been developed with a specific application in mind. Thus, LAMMPS is fraught with ideas for biophysical and materials systems, GROMACS is designed for biophysics, EMU is loaded with peridynamic concepts, and so on. In our work, on the other hand, we are careful to keep the physical modeling aspects of the code out of the fundamental code architecture. This

allows users to more easily get started with their particular physical model, without having to first learn about unrelated modeling domains.

The requirements for designing and building parallel programs can be summarized in the following steps [Foster 1995]:

1. Partitioning: divide the computation to be performed and the data operated on by the computation into small tasks. The focus here should be on identifying tasks that can be executed in parallel.
2. Communication: determine what communication needs to be carried out among the tasks identified in the previous step.

Regarding these requirements, pdQ2, like pdQ, uses the spatial decomposition algorithm for partitioning the problem. Communication between processors is likewise based upon Plimpton's message passing approach like pdQ [Plimpton 1995]. To reduce the number of required messages and their sizes, both codes use the concepts of procCubes, cells, and walls. In this chapter, we introduce the new concepts of *cores*, *blocks*, and *skins* for communication, which result in higher efficiency with respect to pdQ.

pdQ2 performs the particle dynamics simulation; however, preprocessing and post- processing are two other important aspects that should be discussed. Preprocessing is the step in which the problem is defined and input files are created for processing. Processing is the step in which physics is simulated. Postprocessing is the step in which the user analyzes and interprets output data.

In Sections 3.2, 3.3, and 3.4, we discuss preprocessing, processing, and postprocessing respectively. In Section 3.5, verification and efficiency of the code are investigated.

3.2. Preprocessing

Preprocessing is typically neither time- nor memory-intensive, and can be easily performed on a single processor machine using an interpreted language like MATLAB. MATLAB, with its easy syntax and rich graphical functionality, is a very effective language for preprocessing the particle dynamics simulations.

In pdQ2, as in the original pdQ, particles possess two types of attributes: *alterable* and *fixed*. Alterable attributes of particles may change during the simulation. They may include current positions, temperatures, and velocities, for example. At the beginning of the simulation, the alterable attributes are set equal to the initial conditions of the problem. On the other hand, fixed attributes of particles do not change during the course of the simulation. The minimal set of alterable attributes that must be defined are the global ID and the initial position for each particle. The minimal set of fixed attributes that must be defined are the global ID and the reference position for each particle. Users can add other physical attributes to these essential ones. The preprocessor saves the particle alterable and fixed attributes in files called *ptclAlterAttrs.dat* and *ptclFixedAttrs.dat* respectively. It is up to the user to define the additional fixed and alterable attributes for a particular simulation.

The preprocessor must also create two additional input files, *pdQInput.dat*, *pdQUserInput.dat*, containing all additional input data required for the simulation. The

philosophy is to have two separate input data files. One is for general simulation requirements (*pdQInput.dat*), required by all pdQ2 simulations, and the other (*pdQUserInput.dat*) is for additional user specifications.

General input data that is needed by all particle dynamics simulations include decomposition, force field, and integration parameters. Decomposition parameters specify the number of processors in the X, Y, and Z directions. The force field parameter is the minimum cell size chosen for the domain decomposition (explained further in Section 3.3). The sole integration parameter specified at this stage is the number of time steps.

The additional inputs that are specific to the user's physical model are saved in the file *pdQUserInput.dat.* More detailed information about how to use and set up particle attributes and input data are provided in the pdQ2 User Manual in the Appendix. We next describe the core of the simulation: the pdQ2 engine, or processing stage.

3.3. Processing

In this stage, parallelization, time loop, particle shuffling and physical modeling are performed. Parallelization and particle shuffling are common to all simulations; however, the physics varies from simulation to simulation. Parallelization includes spatially partitioning of the problem and communicating between processors. Particle shuffling is necessary for simulations where particles can move from cell to cell for example in molecular dynamics. As physical modeling is defined by the user, several user files such as *userSetup.F*, *userIntegrate.F*, *userForce.F,* and *userModule.F* are

provided for this purpose. Thus, pdQ2 handles the parallelization and particle shuffling and the user needs only define the physical behavior of the particles.

In Sections 3.3.1, 3.3.2, 3.3.3, 3.3.4 and 3.3.5, partitioning, time loop, communication, particle shuffling, and user files are described.

3.3.1. Partitioning

In this section, we will discuss domain decomposition, multi-dimensional arrays, blocks, procCube, cell, core and skin concepts, and particle allocation to processors.

Domain decomposition

In pdQ2, the spatial simulation domain is decomposed into required subdomains. The domain extents are determined by finding the maximum and minimum reference (fixed) coordinate of all particles in each direction: X_{min}, X_{max}, Y_{min}, Y_{max}, Z_{min}, and Z_{max}. This is accomplished in the subroutine *ComputeDomainParams*.

With the bounding coordinates of particles, a 3D cuboid is established. In 2D, this is the dashed rectangle as indicated in Fig. 3.1. The cuboid or rectangle is expanded by a margin, Δ. Δ is the user-defined margin provided for particle movement. The expanded cuboid or rectangle is defined as the domain of the problem.

Having established the problem domain, the domain is partitioned into *cores* as shown in Fig. 3.2. A core is a 3D cuboid assigned to a processor. The extent of each core in each direction is equal to the length of the domain in that direction divided by the

number of processors in that direction. For instance, the length of a core in the X direction is $\frac{X_{max} - X_{min} + 2\Delta}{\text{Number of processors in X direction}}$.

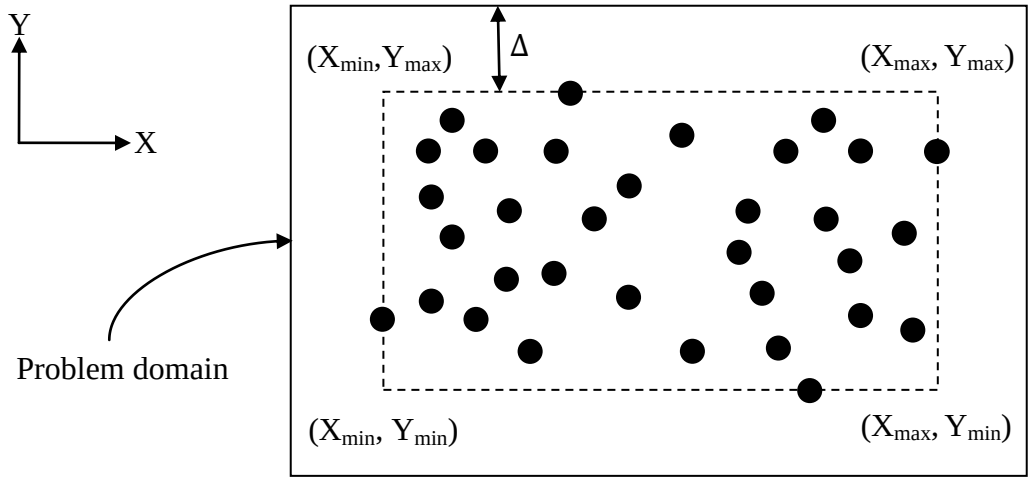


Figure 3.1 – 2D view of domain boundaries.

The material horizon or force radius, δ, is the radius around a particle beyond which the particle does not interact with surrounding particles. As particles need only to interact with neighboring particles closer than δ, the cores are further subdivided into *cells* [Beazley et al. 1994]. The cells provide a structure that allows for simple and efficient searches for surrounding particles by examining only adjacent cells. Cells are 3D cuboids. The dimensions of a cell are chosen to be slightly larger than δ so that interacting particles are guaranteed to have a reference or current location in adjacent cells. In MD-type problems, where particle motions can be large, particles can be "shuffled" from cell to cell. In this case, "current" particle positions are used. In each core, 3D indices of cells vary from one to the number of cells in each respective direction. Cell layout in two adjacent cores is shown in Fig. 3.3.

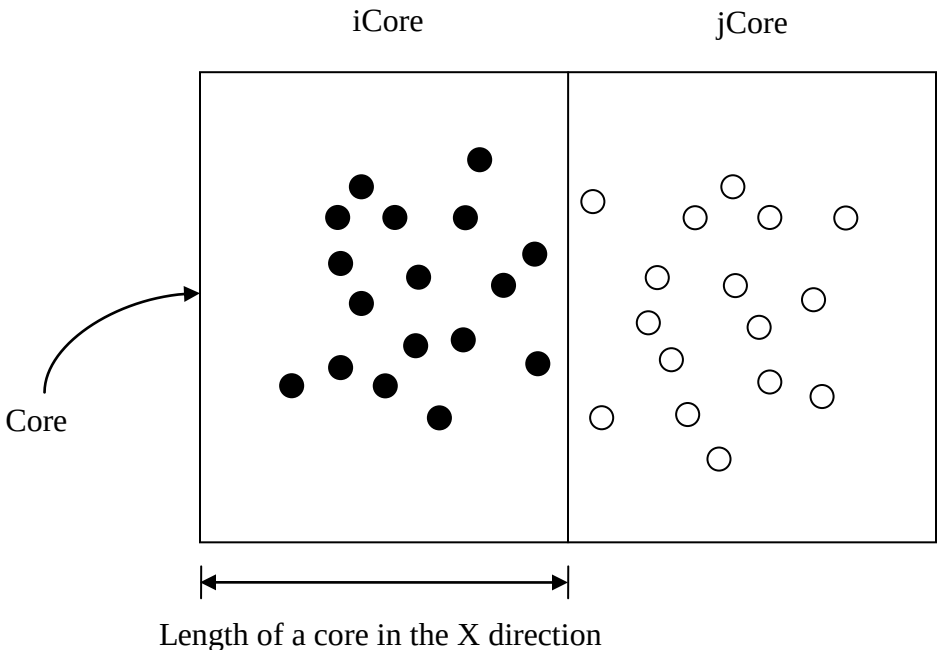


Figure 3.2 – Cores in the X direction.

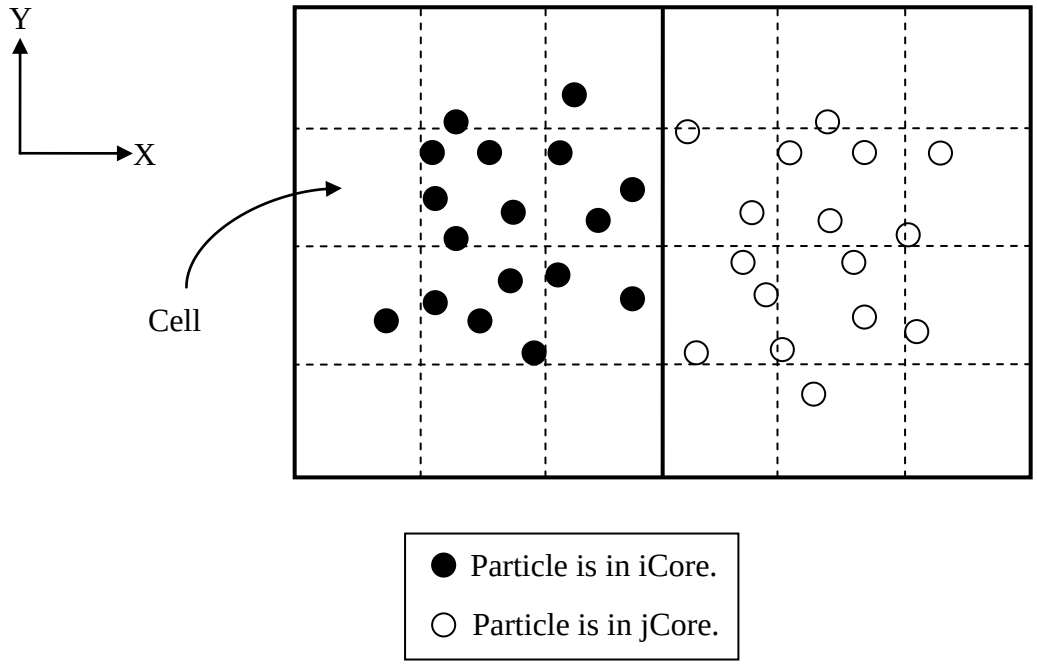


Figure 3.3 – Cells in the cores.

Multi-dimensional arrays

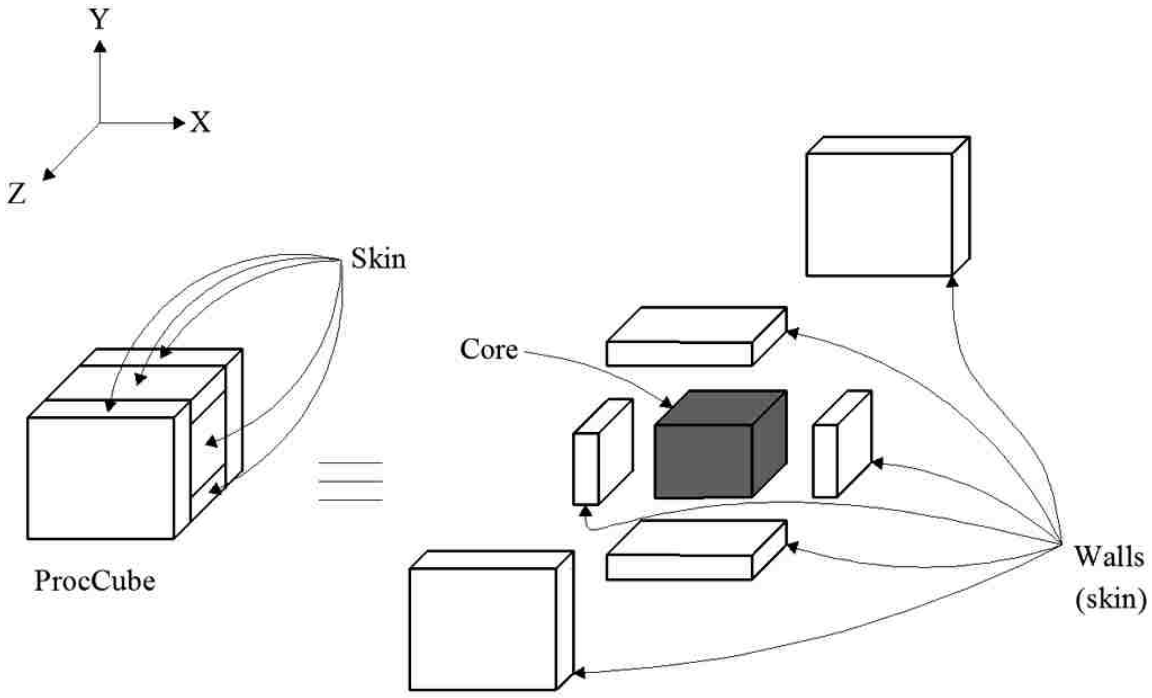
In particle dynamics simulations, particles can be identified or labeled by various methods. Particles are identified in pdQ2 based on their 3D cell indices and a local cell particle index with respect to each cell; however, they also have a global unique integer ID that is defined by the preprocessor as an attribute. This makes it possible to take advantage of the multi-dimensional array capability provided by FORTRAN, as pdQ2 is written in FORTRAN 90, and the use of up to seven-dimensional arrays is possible. Hence, arrays identifying particles can be multi-dimensional. In our design, particles are indentified by the five-dimensional arrays *ptclAlterAttrs*, *ptclFixedAttrs*, and *ptclIntegAttrs*, which are real (generally double precision). *ptclAlterAttrs* array corresponds to particle alterable attributes. *ptclFixedAttrs* array corresponds to particle fixed attributes. The *ptclIntegAttrs* array contains the integrable particle attributes. These arrays are dimensioned as follows:

```
Real ptclAlterAttrs (iAttr, iPtclCell, iCellX, iCellY, iCellZ)

Real ptclFixedAttrs (iAttr, iPtclCell, iCellX, iCellY, iCellZ)

Real ptclIntegAttrs (iAttr, iPtclCell, iCellX, iCellY, iCellZ)
```

where *iAttr* is the particle attribute index, *iPtclCell* is the particle index with respect to each cell (varying from one to the maximum number of particles in a cell), and *iCellX*, *iCellY*, and *iCellZ* are 3D indices of the cell with respect to the procCube which is described in the following section. In each core, 3D indices of cells vary from one to the number of cells in each respective direction.

Particle allocation to processors

To allocate particles to their processors, it is necessary to define a *procCube* for each core. To define the procCube, the concept of *skin* is introduced. The skin is a single layer of cells surrounding each core in all directions as shown in Fig. 3.4. Together, the core and the skin form a procCube. Each skin has six *walls*.



.

Figure 3.4 – Layout of a procCube with core, skin, and walls.

As an illustration, two adjacent procCubes are shown in Fig. 3.5. A procCube is defined for each processor. When the program runs simultaneously on all processors, all particles are read by each processor, but the processor stores only the particles that belong to its procCube. In the subroutine *ComputeProcParams*, particles are assigned to the procCube that they are in. Also, they are assigned cell indices. Particles in a specific

cell and procCube are saved in a 3D integer array called *numPtclsCell*, with indices *(iCellX, iCellY, iCellZ)*. nCX, nCY, and nCZ are the number of the cells in the X, Y, and Z directions in the core. In each procCube, when defining core and skin cells, the cell indices vary between 0 and nCX+1, nCY+1, nCZ+1 in each direction.

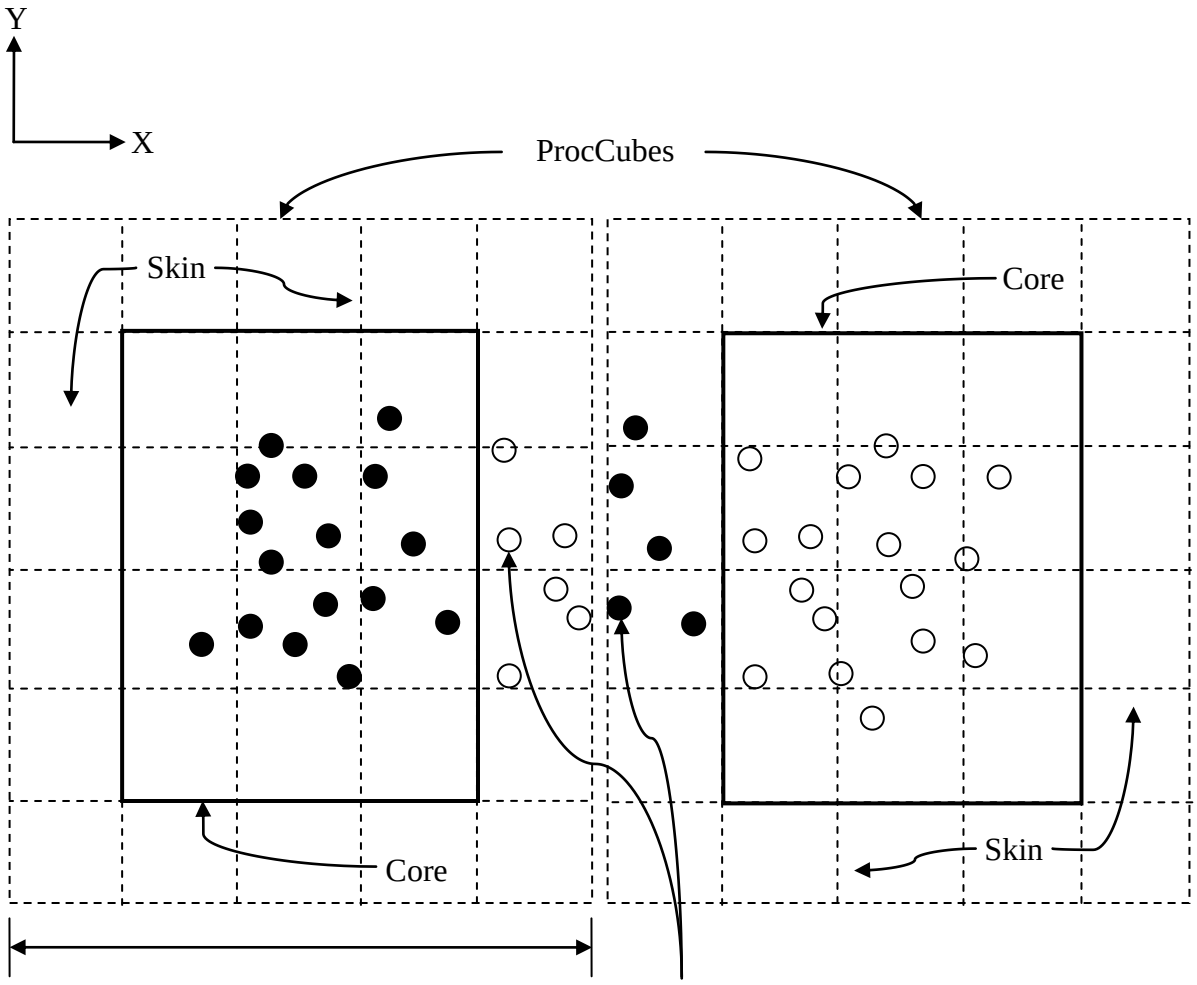


Figure 3.5 – Two adjacent procCubes in the X direction.

3.3.2. Time loop

After partitioning the problem, the time loop is initiated. The time loop is the mechanism by which the particle alterable attributes evolve. It is up to the user to decide how many time steps the time loop should be repeated and when it should be exited. The time loop includes the communication between processors, force computation, integration, and shuffling. The communication between processors is described in Section 3.3.3. The force computation and integration are defined as user subroutines in the user files as described in Section 3.3.5 and in the Appendix. Particle shuffling is explained in Section 3.3.4. In pdQ2, the time loop pseudocode is as follows:

```
Do loop until the stopping criterion
      Force computation
      Integration
      Interprocessor communication
      Shuffling
End loop
```

Interprocessor communication is associated by the subroutine *ExchangePtclAlterAttr*. Force computation and integration are accomplished by the subroutines defined in the *userForce.F* and *userIntegrate.F* files respectively. Shuffling is accomplished in the subroutine *shuffle.*

3.3.3. Communication between processors

As shown in previous section, the problem has been decomposed into an array of procCubes. We focus now on a single reference processor and its procCube, which we refer to as the *home procCube*. Particles residing in a boundary cell in the core of the

home procCube can potentially interact with neighboring particles that are within its material horizon but also lie in the cores of an adjacent procCube. The necessary particle data from cores of adjacent procCubes must be sent to the skin of home procCube. These communications are executed in the code by the Message Passing Interface (MPI) subroutines. MPI is a library of communication routines for sending and receiving data (messages) between processors. These routines are callable in FORTRAN [Pacheco 2011].

The information that must be passed between processors at each time step consists of particle alterable attributes, which are stored in the array *ptclAlterAttrs*. Before entering the time loop, particle fixed attributes are read into the core and skin cells in the array *ptclFixedAttrs* by the subroutine *SetupProblem*.

The home procCube is potentially surrounded by twenty-six adjacent procCubes. Using Plimpton's method of message passing as described in Section 2.1.2, messages are sent and received for each procCube by the subroutine *ExchangePtclAlterAttrs*, and the message passing is accomplished in six steps per time step. Before describing the six steps of message passing as implemented within pdQ2, we explain the naming convention for wall and core cells. These are defined as *cell blocks*. A cell block is a cuboidal collection of cells that are responsible for sending and receiving particle alterable attributes (*ptclAlterAttrs* array) between processors. Cell blocks can lie either in the skin or in the core. Cell blocks in the skin are shown in Fig. 3.6.

The core and the walls in the skin are defined according to cell blocks as follows. nCX, nCY, and nCZ are the number of cells in the core of a procCube in the X, Y, and Z directions:

```
iCore = Cell block [1:nCX, 1:nCY, 1:nCZ]

iXnSkin = Cell block [0, 1:nCY, 1:nCZ]

iXpSkin = Cell block [nCX+1, 1:nCY, 1:nCZ]

iYnSkin = Cell block [0:nCX+1, 0, 1:nCZ]

iYpSkin = Cell block [0:nCX+1, nCY+1, 1:nCZ]

iZnSkin = Cell block [0:nCX+1, 0:nCY+1, 0]

iZpSkin = Cell block [0:nCX+1, 0:nCY+1, nCZ+1]
```

The sending walls in the core are defined as follows:

```
iXnCore = Cell block [1, 1:nCY, 1:nCZ]

iXpCore = Cell block [nCX, 1:nCY, 1:nCZ]

iYnCore = Cell block [0:nCX+1, 1, 1:nCZ]

iYpCore = Cell block [0:nCX+1, nCY, 1:nCZ]

iZnCore = Cell block [0:nCX+1, 0:nCY+1, 1]

iZpCore = Cell block [0:nCX+1, 0:nCY+1, nCZ]
```

If the 3D indices of the home procCube are [iPX, iPY, iPZ], then the convention for naming adjacent procCubes is as follows:

```
iXn procCube = [iPX-1, iPY, iPZ]

iXp procCube = [iPX+1, iPY, iPZ]
```

```
iYn procCube = [iPX, iPY-1, iPZ]

iYp procCube = [iPX, iPY+1, iPZ]

iZn procCube = [iPX, iPY, iPZ-1]

iZp procCube = [iPX, iPY, iPZ+1]
```

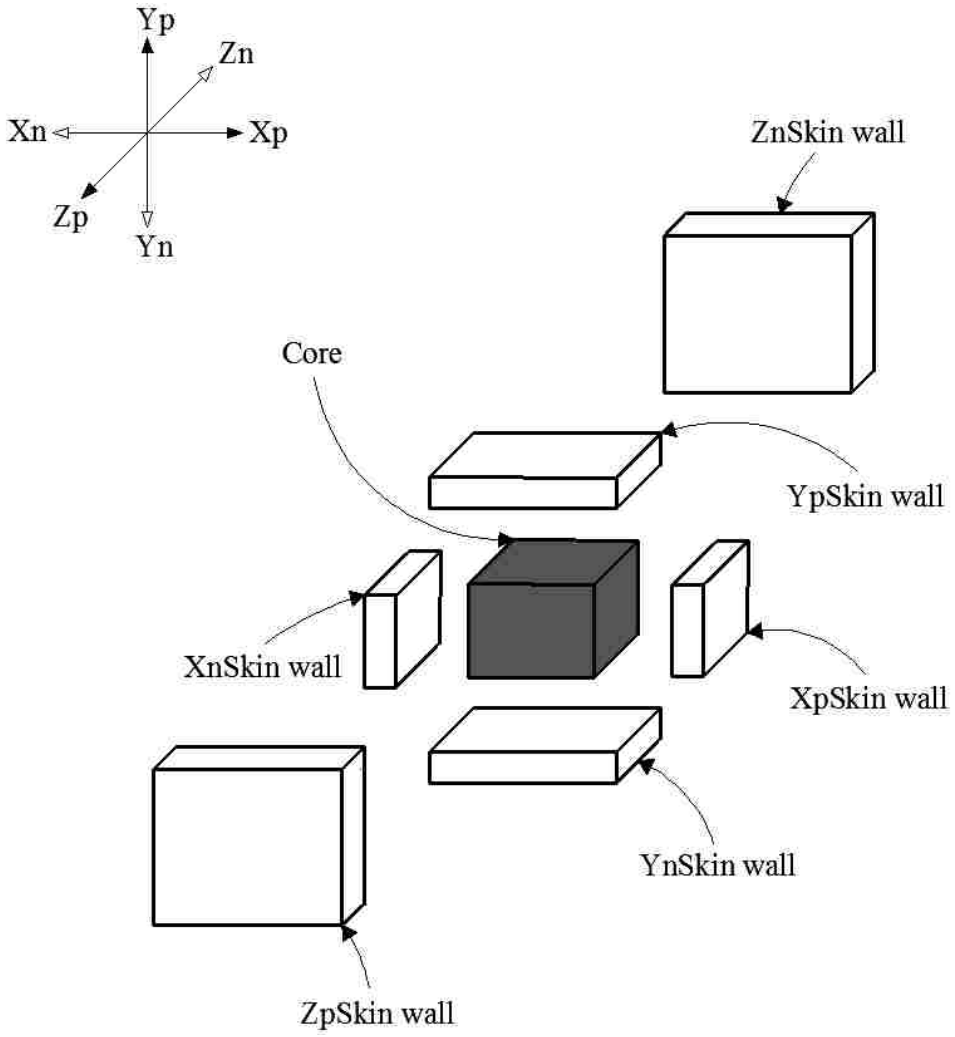


Figure 3.6 – Cell blocks in the skin of a procCube.

With the definitions in hand for the walls, core, and adjacent procCube coordinates, we now illustrate the six message passing steps. Note that these six steps are executed in sequence. Also, prior to this step, it is assumed that particles in the core have been updated (shown in yellow) by user-defined subroutines in the *userIntegrate.F* file, which are described in Section 3.3.

Step 1. Message passing in the Xp direction:

In the first step, as shown in Fig. 3.7, the home procCube sends *ptclAlterAttrs* from its iXpCore wall to the iXnSkin wall of the iXp procCube. At the same time, the home procCube receives *ptclAlterAttrs* from the iXpCore wall of the iXn procCube into its iXnSkin wall. This is accomplished using the MPI command "MPI_SENDRECV"

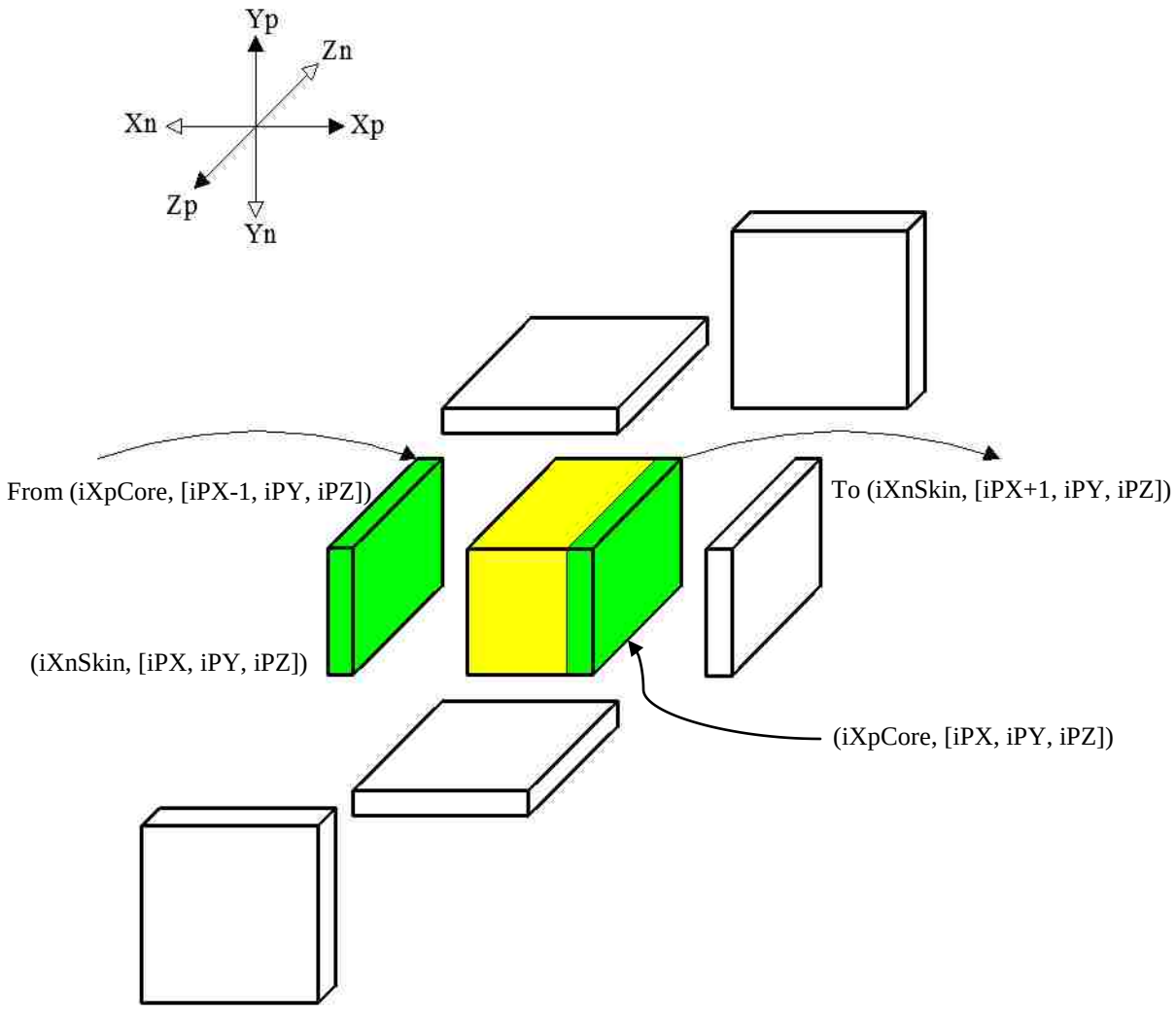


Figure 3.7 – Message passing in the Xp direction, with focus on home procCube. Send *ptclAlterAttrs* from (iXpCore, [iPX, iPY, iPZ]) to (iXnSkin, [iPX+1, iPY, iPZ]). Receive *ptclAlterAttrs* from (iXpCore, [iPX-1, iPY, iPZ]) to (iXnSkin, [iPX, iPY, iPZ]).

Step 2. Message passing in the Xn direction:

In the second step, as shown in Fig. 3.8, the home procCube sends *ptclAlterAttrs* from its iXnCore wall to the iXpSkin wall of its iXn procCube. Simultaneously, the home procCube receives *ptclAlterAttrs* from the iXnCore wall of its iXp procCube and places them in its iXpSkin wall.

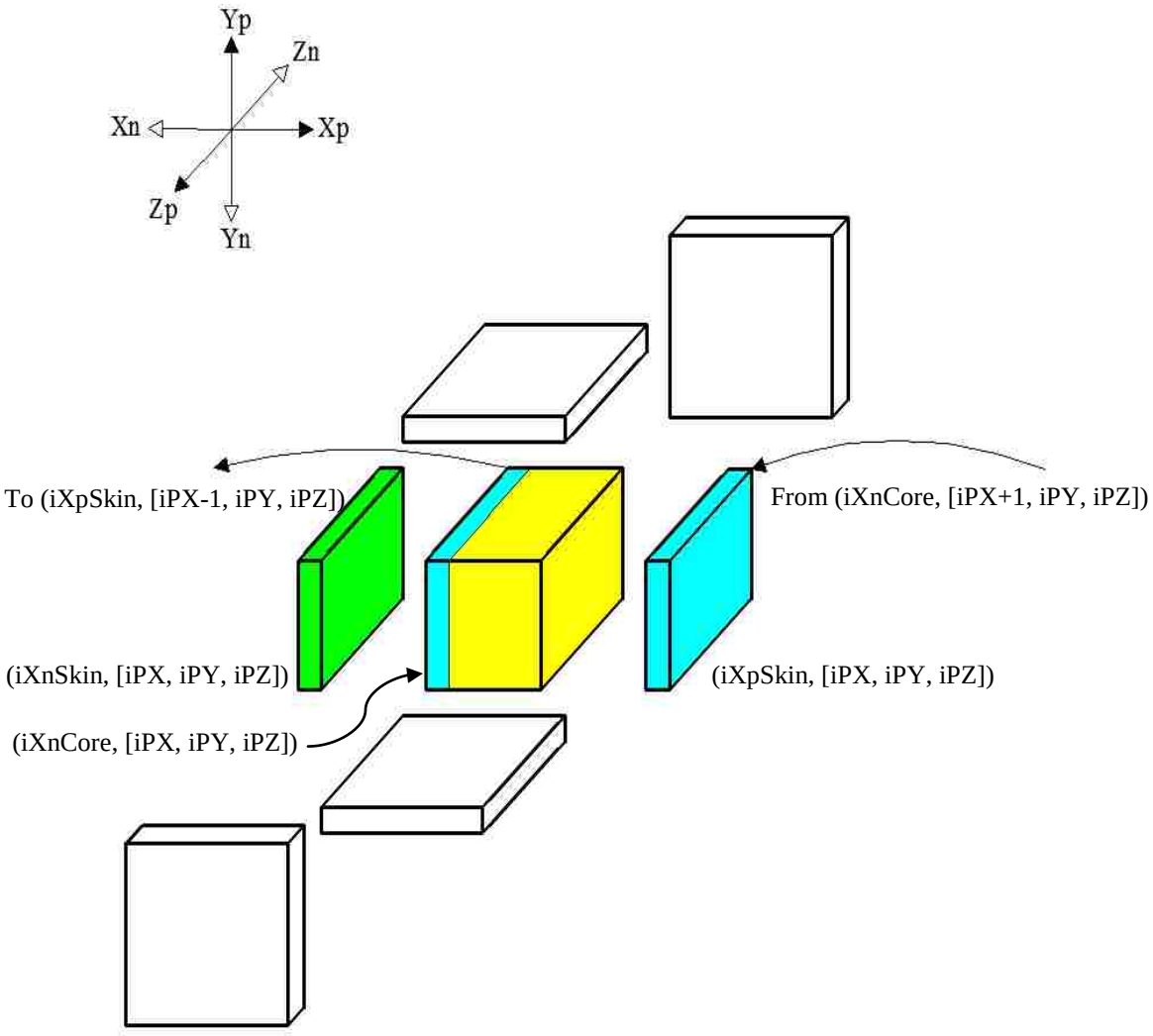


Figure 3.8 – Message passing in the Xn direction, with focus on home procCube. Send *ptclAlterAttrs* from (iXnCore, [iPX, iPY, iPZ]) to (iXpSkin, [iPX-1, iPY, iPZ]). Receive *ptclAlterAttrs* from (iXnCore, [iPX+1, iPY, iPZ]) to (iXpSkin, [iPX, iPY, iPZ]).

Now after these two steps, message passing has been completed in the X direction and the core and skins that have been updated are shown in color in Fig. 3.9.

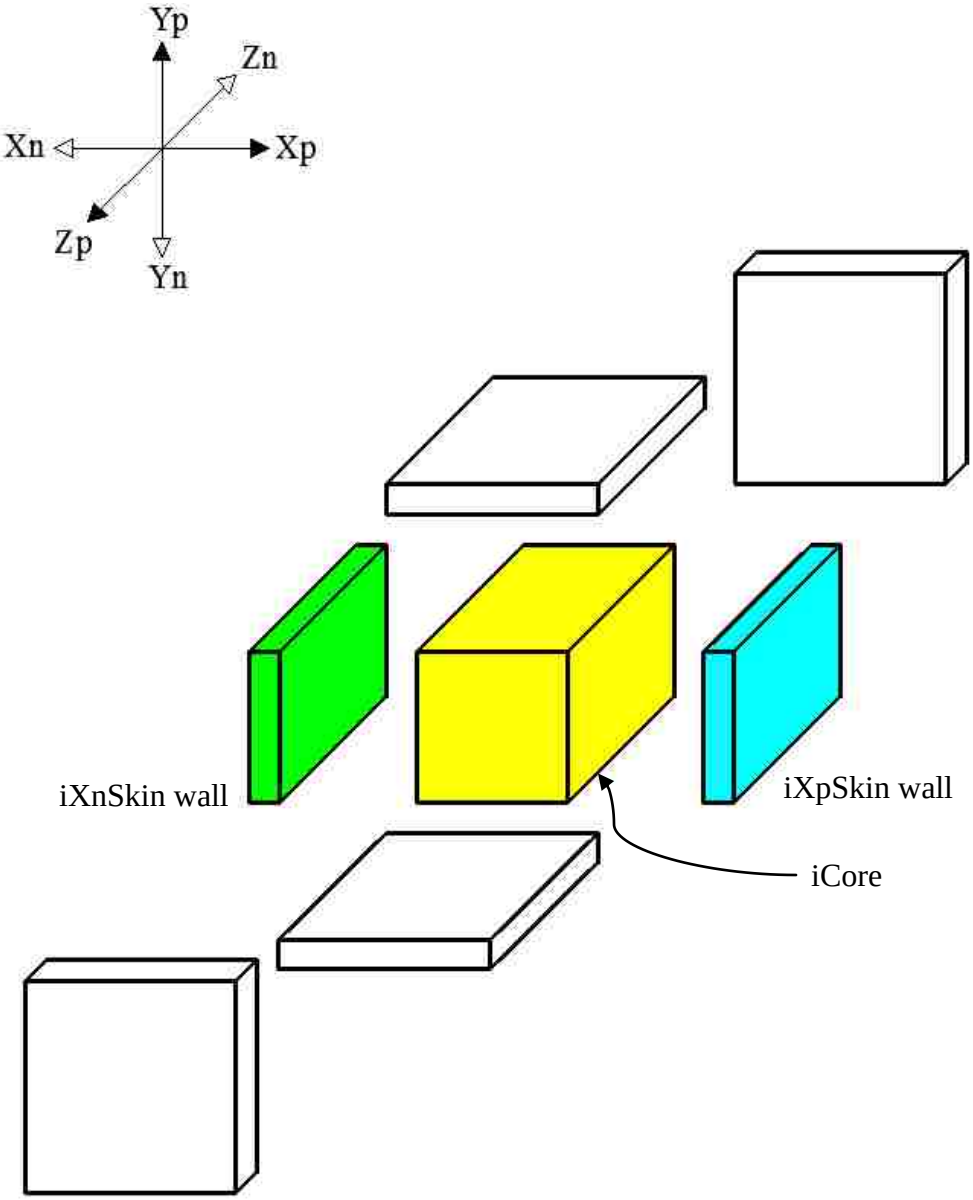


Figure 3.9 – Walls in the skin that have been updated (shown in green and blue) after completion of message passing in the X direction. The core that has already been updated (shown in yellow) before message passing.

Step 3. Message passing in the Yp direction:

In the third step, as shown in Fig. 3.10, the home procCube sends *ptclAlterAttrs* from its iYpCore wall, including needed cells that were received as part of the X direction message passing, to the iYnSkin wall of the iYp procCube. In the meantime, the home procCube receives *ptclAlterAttrs* from the iYpCore wall of its iYn procCube into its iYnSkin wall.

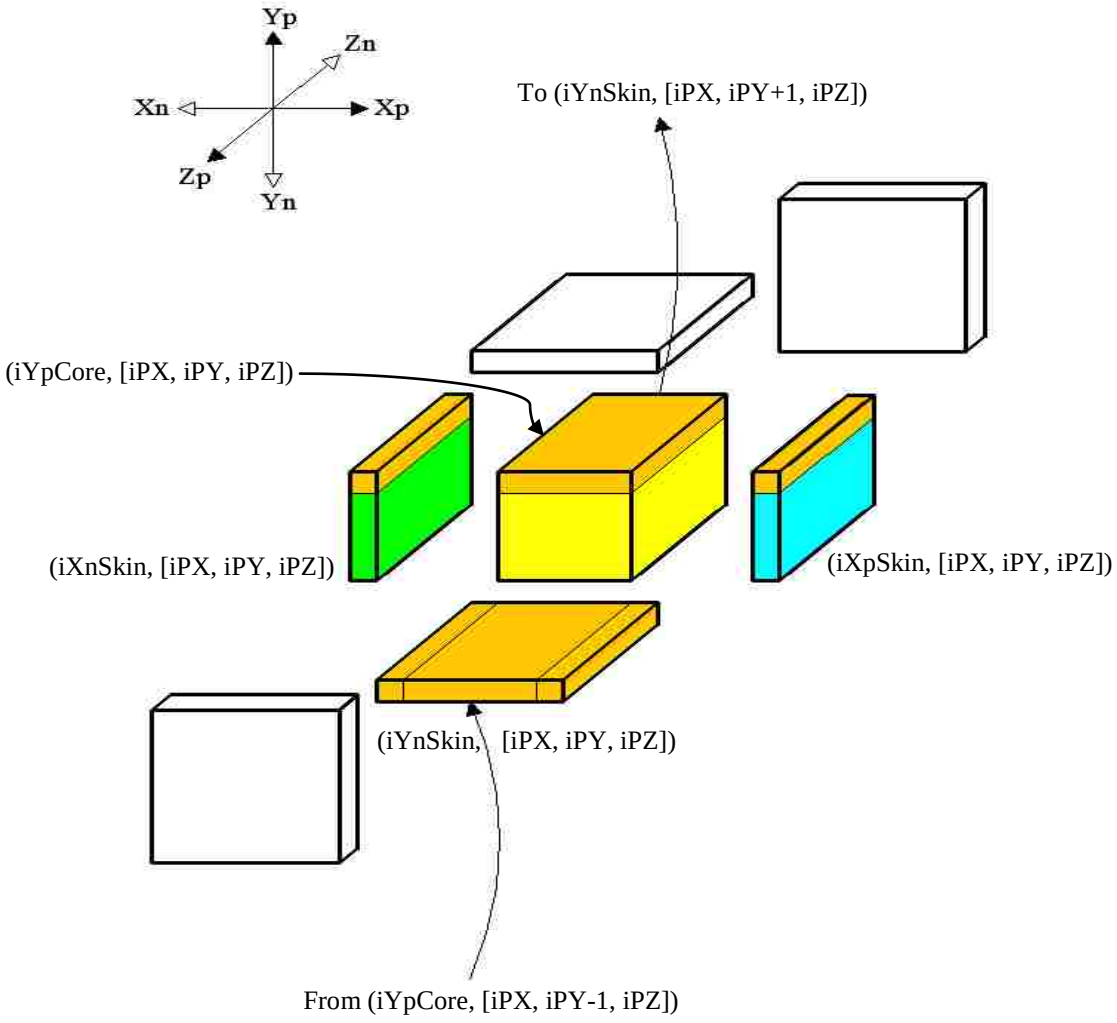


Figure 3.10 – Message passing in the Yp direction, with focus on home procCube. Send *ptclAlterAttrs* from (iYpCore, [iPX, iPY, iPZ]) to (iYnSkin, [iPX, iPY+1, iPZ]). Receive *ptclAlterAttrs* from (iYpCore, [iPX, iPY-1, iPZ]) to (iYnSkin, [iPX, iPY, iPZ]).

Step 4. Message passing in the Yn direction:

In the fourth step, as shown in Fig. 3.11, the home procCube sends *ptclAlterAttrs* from its iYnCore, including previously-received cells from the X direction message passing, to the iYpSkin wall of its iYn procCube. Concurrently, the home procCube receives *ptclAlterAttrs* from the iYnCore wall of its iYp procCube into its iYpSkin wall.

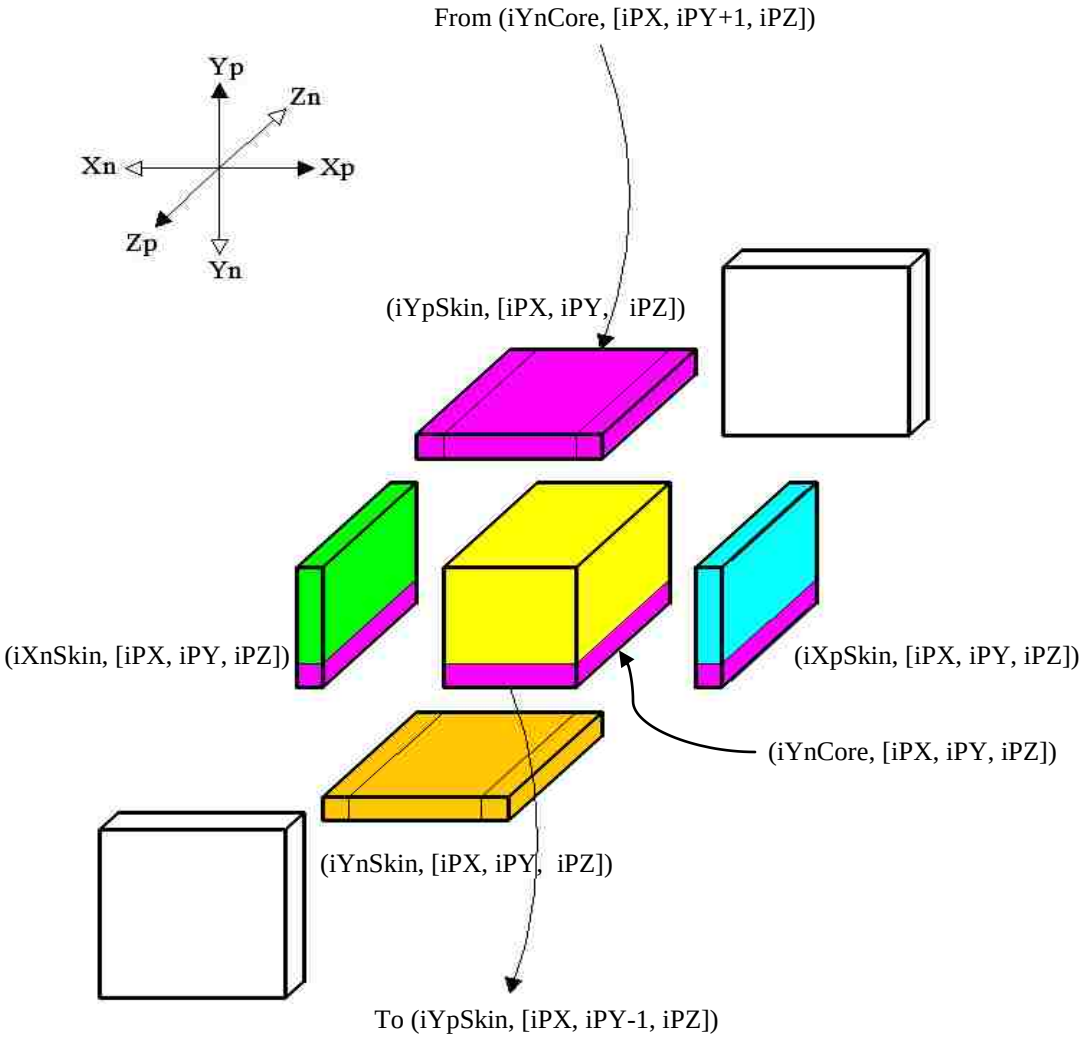


Figure 3.11 – Message passing in the Yn direction, with focus on home procCube. Send *ptclAlterAttrs* from (iYnCore, [iPX, iPY, iPZ]) to (iYpSkin, [iPX, iPY-1, iPZ]). Receive *ptclAlterAttrs* from (iYnCore, [iPX, iPY+1, iPZ]) to (iYpSkin, [iPX, iPY, iPZ]).

After completion of the four steps of message passing in the X and Y directions, received messages in the home procCube are indicated in color in Fig. 3.12.

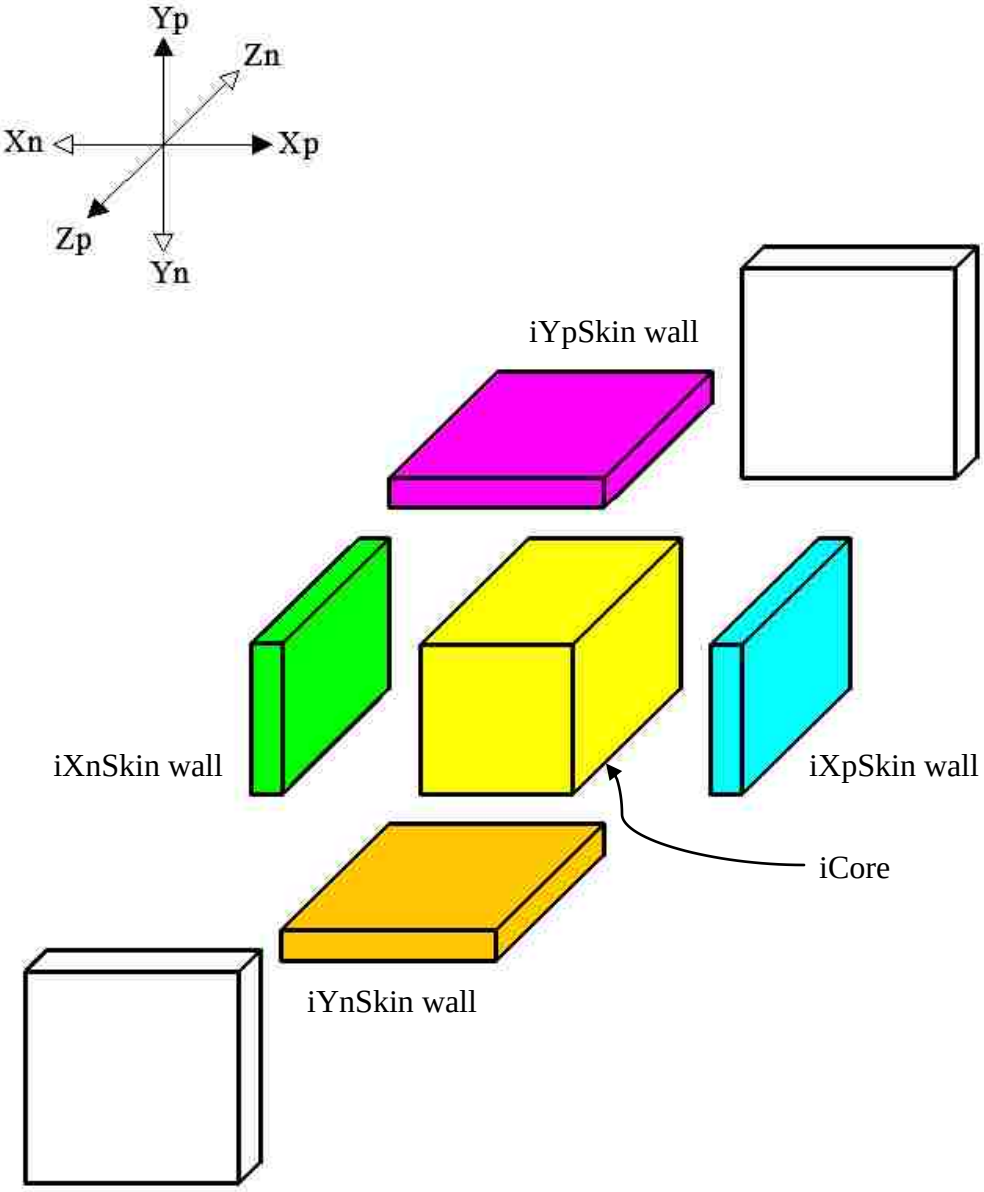


Figure 3.12 – Walls in the skin that have been updated (shown in color) after completion of message passing in the X and Y directions.

Step 5. Message passing in the Zp direction:

In the fifth step, as shown in Fig. 3.13, the home procCube sends *ptclAlterAttrs* from its iZpCore wall, including received messages in X and Y directions from previous steps, to the iZnSkin wall of its iZp procCube. Simultaneously, the home procCube receives *ptclAlterAttrs* from the iZpCore wall of its iZn procCube into its iZnSkin wall.

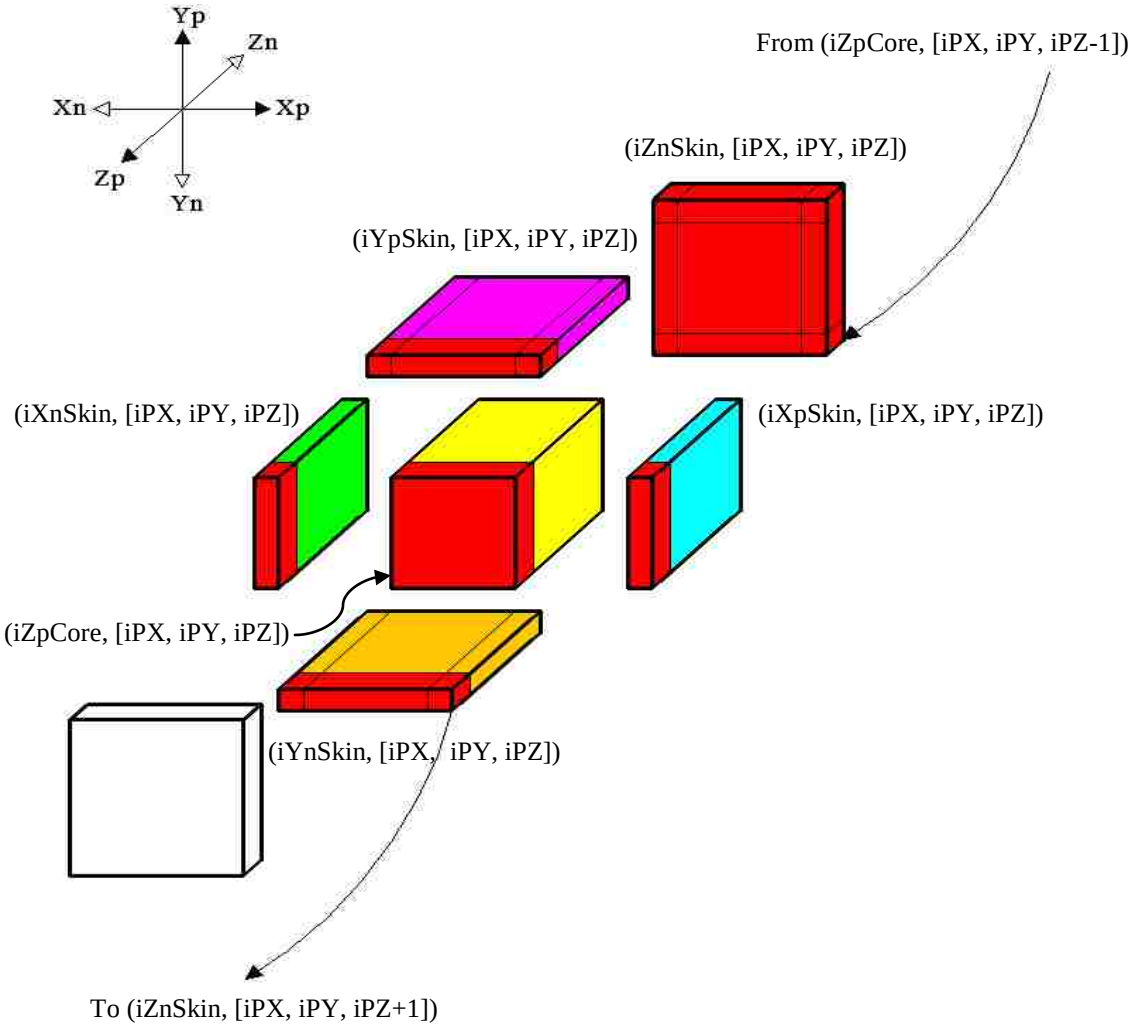


Figure 3.13 – Message passing in the Zp direction, with focus on home procCube. Send *ptclAlterAttrs* from (iZpCore, [iPX, iPY, iPZ]) to (iZnSkin, [iPX, iPY, iPZ+1]). Receive *ptclAlterAttrs* from (iZpCore, [iPX, iPY, iPZ-1]) to (iZnSkin, [iPX, iPY, iPZ]).

Step 6. Message passing in the Zn direction:

In the sixth step, the home procCube sends *ptclAlterAttrs* from its iZnCore wall including previously-received messages received on the skin from the message passing in X and Y directions, to the iZpSkin wall of its iZn procCube. At the same time, the home procCube receives *ptclAlterAttrs* from the iZnCore wall of its iZp procCube and place them in its iZpSkin wall.

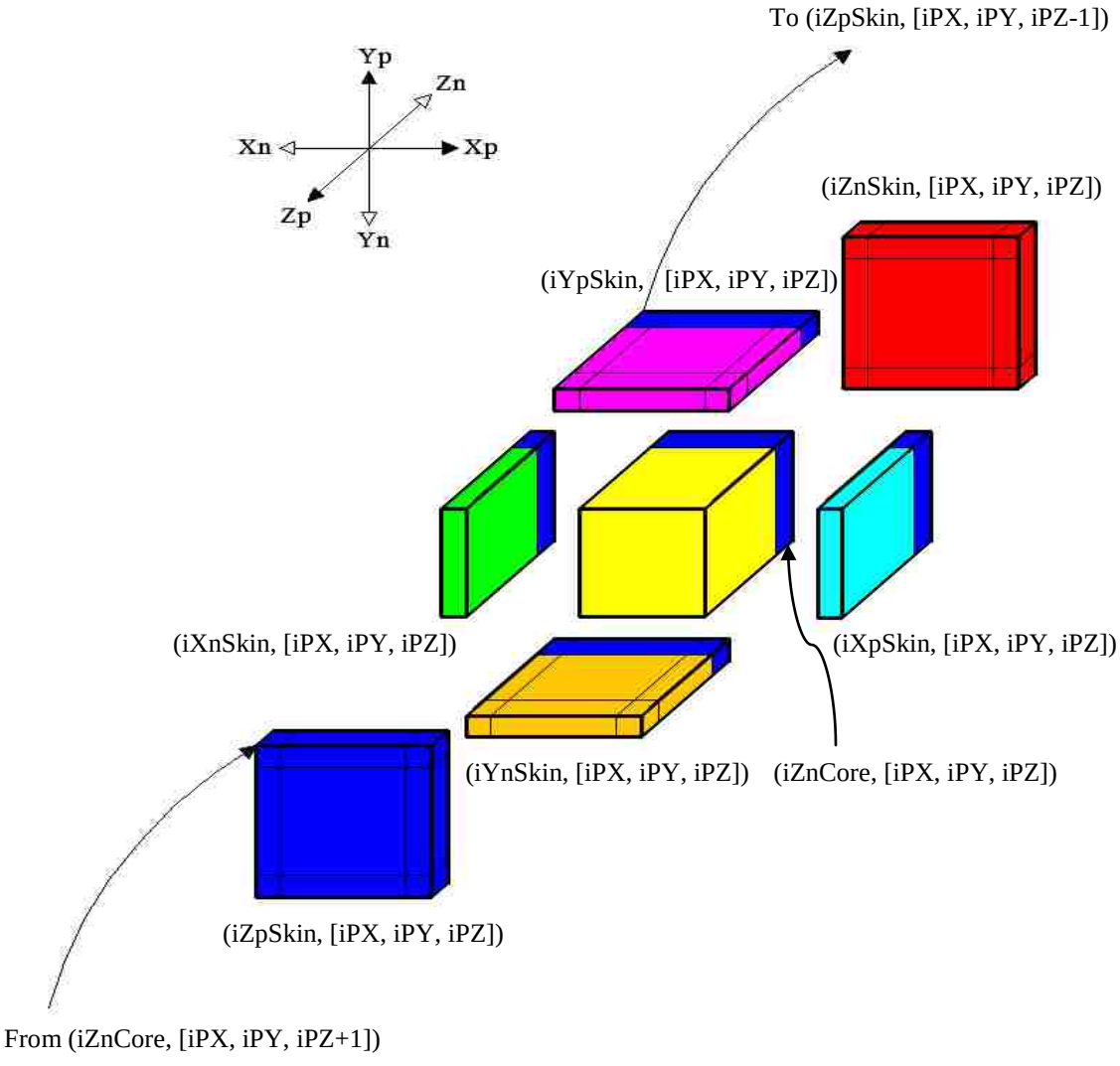


Figure 3.14 – Message passing in the Zn direction, with focus on home procCube. Send *ptclAlterAttrs* from (iZnCore, [iPX, iPY, iPZ]) to (iZpSkin, [iPX, iPY, iPZ-1]. Receive *ptclAlterAttrs* from (iZnCore, [iPX, iPY, iPZ+1]) to (iZpSkin, [iPX, iPY, iPZ]).

Finally, message passing is now complete in the six different directions and all the necessary particle alterable attributes data have been received in the skins of the home procCube. Received messages are indicated in color in Fig. 3.15. Now, forces on all particles in the core of the home procCube can be correctly computed using particle information which resides on the home procCube.

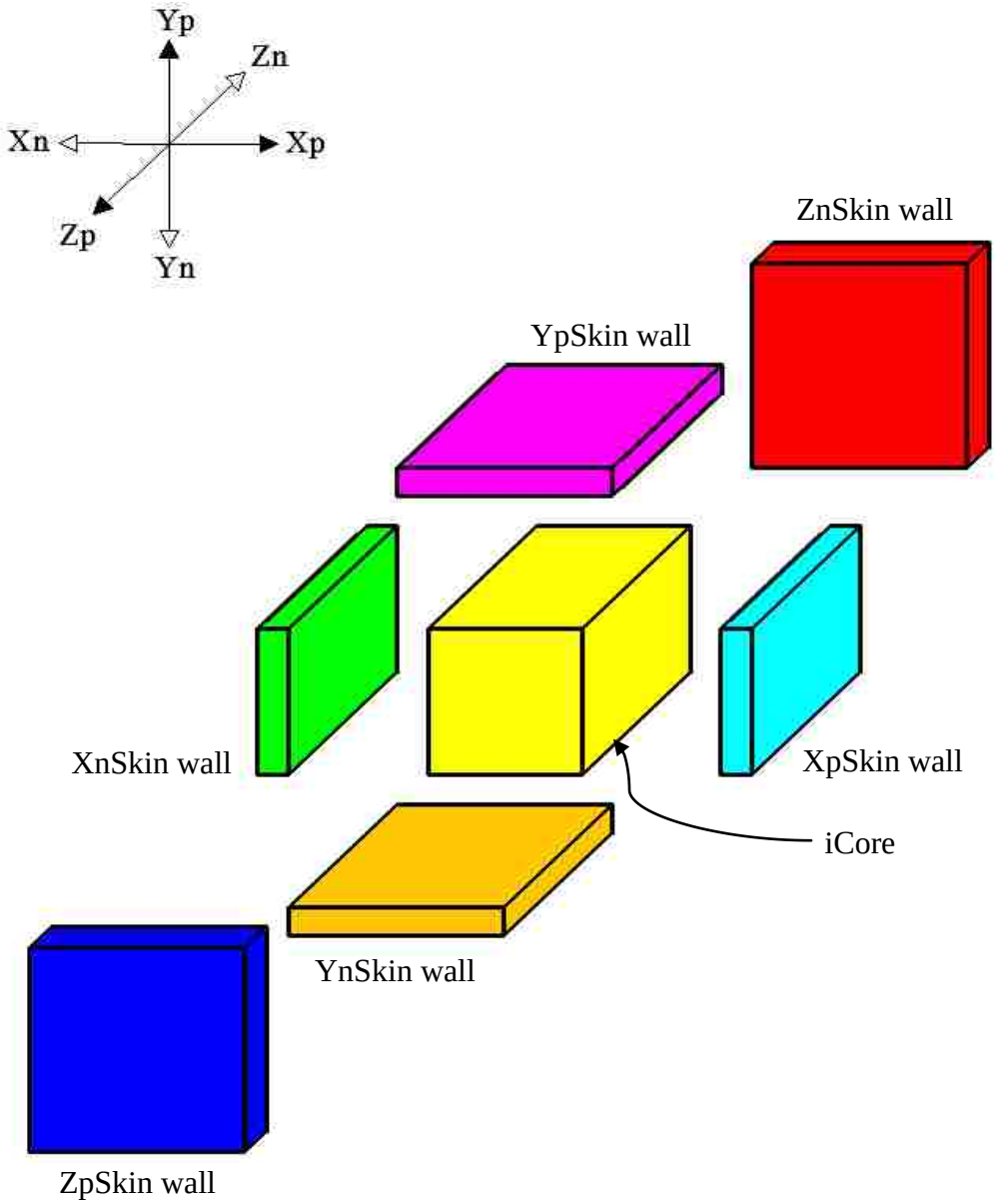


Figure 3.15 – Walls in the skin after completion of message passing in the X, Y, and Z directions.

3.3.4. Moving particles among cells and procCubes: particle shuffle

Cell indices are used as a mechanism to sort particles into geometric buckets, in an effort to avoid needless zero-force computations and to thus limit the number of force interactions that must be computed at each time step. The cell size is set somewhat larger than the force radius (δ), and thus a particle in the cell with address (iCX, iCY, iCZ) will only potentially interact with particles within cells within cell blocks (iCX−1 : iCX +1, iCY−1 : iCY +1, iCZ−1 : iCZ +1).

In simulations of solids composed of particles undergoing small relative deformations, particles do not change their non-zero-force neighbors as the solid deforms (even if absolute deformations are large), and therefore it is unnecessary to alter the particles' cell addresses during the simulation. However, for simulations of solids undergoing large relative deformations, and for liquids, gases, molecules, and atoms, particles may undergo large relative deformations, and the neighbors with which they interact may change during the course of the simulation.

Each particle has a current geometric location (x, y, z) as well as a cell address (iCX, iCY, iCZ). Each cell is defined by an unchanging geometric location defined by the cuboid with limits (iCX_{min}, iCX_{max}; iCY_{min}, iCY_{max}; iCZ_{min}, iCZ_{max}). At each time step, as the particle's current location (x, y, z) is updated, a flag is raised if the particle's current location (x, y, z) is no longer geometrically located within its containing cell's cuboid. This flag indicates that it is necessary to shuffle the particle to a neighboring cell. (The assumption is made that the magnitude of particle motion, Δd, in each time step is

sufficiently small (say, $\Delta d < 0.1 \times cell\ size$) that particles need only be shuffled between immediately adjacent cells.)

Two operations are necessary to shuffle a particle from iCell to jCell in a procCube: delete the particle from iCell and add the particle to jCell, as shown in Fig. 3.16. To delete the particle K from iCell with N_{iC} particles, the N_{iC}^{th} particle is copied to the location of the K^{th} particle in the *ptclAlterAttrs* array, and then the number of particles is reduced by 1: $N_{iC} = N_{iC} - 1$ as shown in Fig. 3.16(c). Thus, gaps in the *ptclAlterAttrs* array can never occur. To add a particle K to jCell with N_{jC} particles, we must first check to make sure that $N_{jC} < N_{max}$ (if $N_{jC} = N_{max}$, either an "out of memory" error message is issued and the simulation is terminated, or the *ptclAlterAttrs* array is extended to allow for more particles). Then particle K is copied to the particle address $N_{jC}+1$ in the *ptclAlterAttrs* array, and the number of particles is increased by 1: $N_{jC} = N_{jC}+1$ as shown in Fig. 3.16(c).

Consider shuffling a particle from iProcCube to jProcCube. Let us assume that at the beginning of the time step, each particle's position (x, y, z) and attributes are up-to-date and the particle is located within the geometrical limits of its containing cell.

Next, internal forces acting upon particles within the core are computed by the user-defined subroutine in the file *userForce.F* as shown in Fig. 3.17(a). These internal forces depend upon particle states within both core and skin cells in the home procCube. As particle positions, attributes, and cell addresses are all up-to-date, the forces acting upon core particles will be correctly computed.

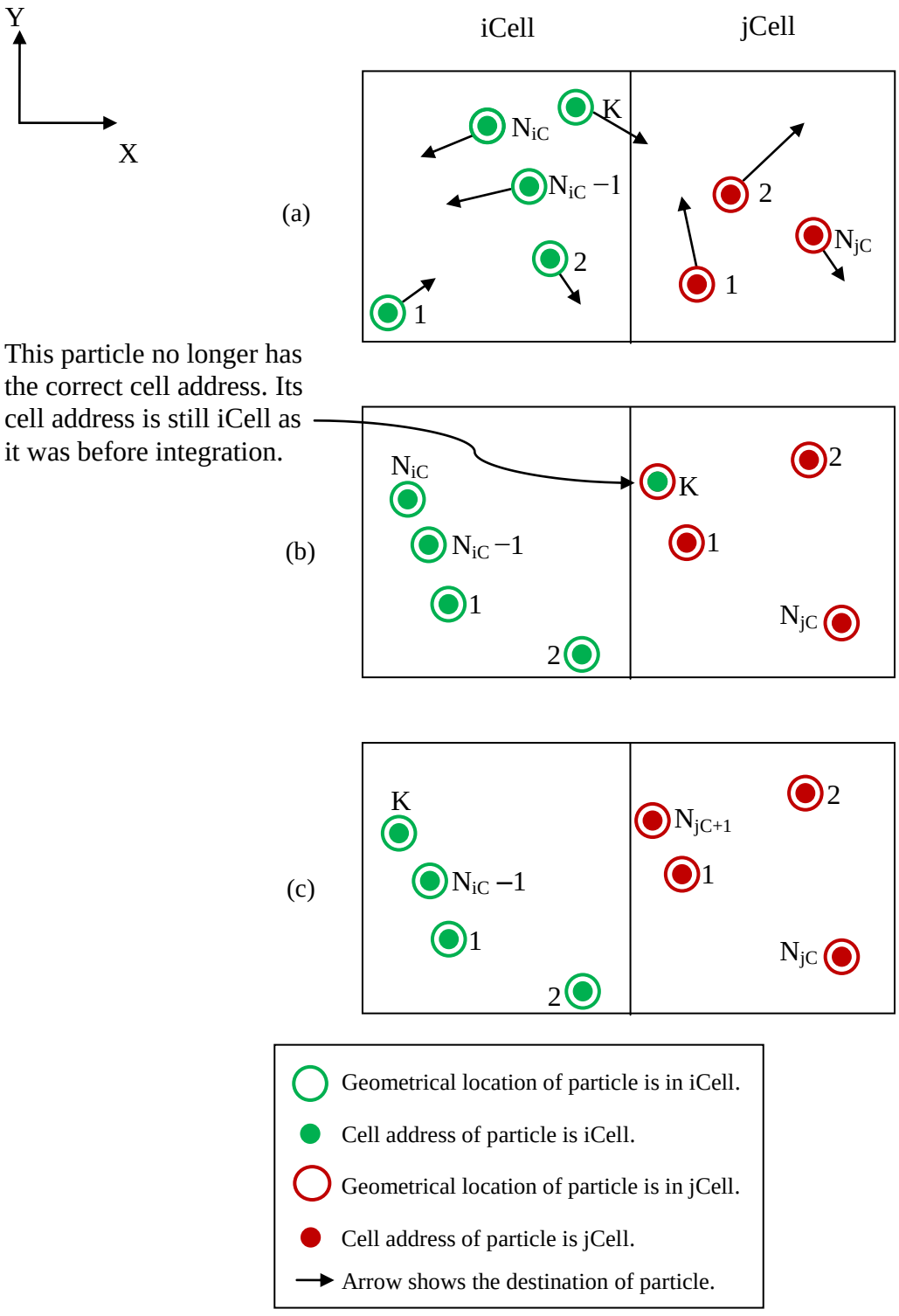


Figure 3.16 – Particle shuffling between iCell and jCell in one time step in a procCube. (a) Configuration of particles before integration. (b) Configuration of the particles after integration. Particle K moves geometrically from iCell to jCell, but it still has wrong cell address (iCell). (c) Configuration of the particles after shuffling.

In the next step, the positions of particles in the core (only) are updated by the user-defined subroutine in the file *userIntegrate.F* as shown in Fig. 3.17(b). Some of these particles may now need to be shuffled, perhaps into a skin cell. But at this stage, particles are not yet shuffled.

In the next step, particles in the core are copied, via message-passing, into the corresponding skins of adjacent procCubes using the subroutine *ExchangePctlAlterAttrs* as shown in Fig. 3.17(c). Note that some of these copied particles may have geometric locations that are not within their containing cells. At the end of this step, all particles in both skin and core cells have correct geometric locations, but their cell addresses may be obsolete because the particle geometric locations may no longer be within their cell boundaries.

Next, if the shuffle flag is set, a shuffle step is performed in the subroutine *Shuffle* within each procCube separately, as shown in Fig. 3.17(d). In this step, all particles no longer within their containing cell are moved to appropriate adjacent cells. Particles contained by core cells may move into adjacent skin cells, and vice versa. At the end of this step, some particles in skin cells may be missing, but all of the skins will be over-written in the next step. Importantly, the core cells are guaranteed to have up-to-date particle positions and up-to-date cell addresses.

At the end of the time step, all core cells and all skin cells contain particles with up-to-date particle positions and cell addresses, and the condition necessary to start with the next time step is fulfilled.

In Fig. 3.17, the particle shuffling between two adjacent procCubes, iProcCube and jProcCube, is shown. We focus on two cells, iCell and iCell′ of iProcCube, and two cells, jCell and jCell′ of jProcCube. This illustrates the case where particle M moves from the core to the skin in iProcCube. At the same time, particle M′ moves from the core to the skin in jProcCube. In Fig. 3.17(a), the particles are shown before integration. In Fig. 3.17(b), the particles are shown after integration and particle M has the wrong cell address. In Fig. 3.17(c), the message-passing between two procCubes is done so that the skins of two procCubes become up-to-date. In Fig. 3.17(d), the wrong cell addresses of particles M and M′ are corrected by the shuffle step.

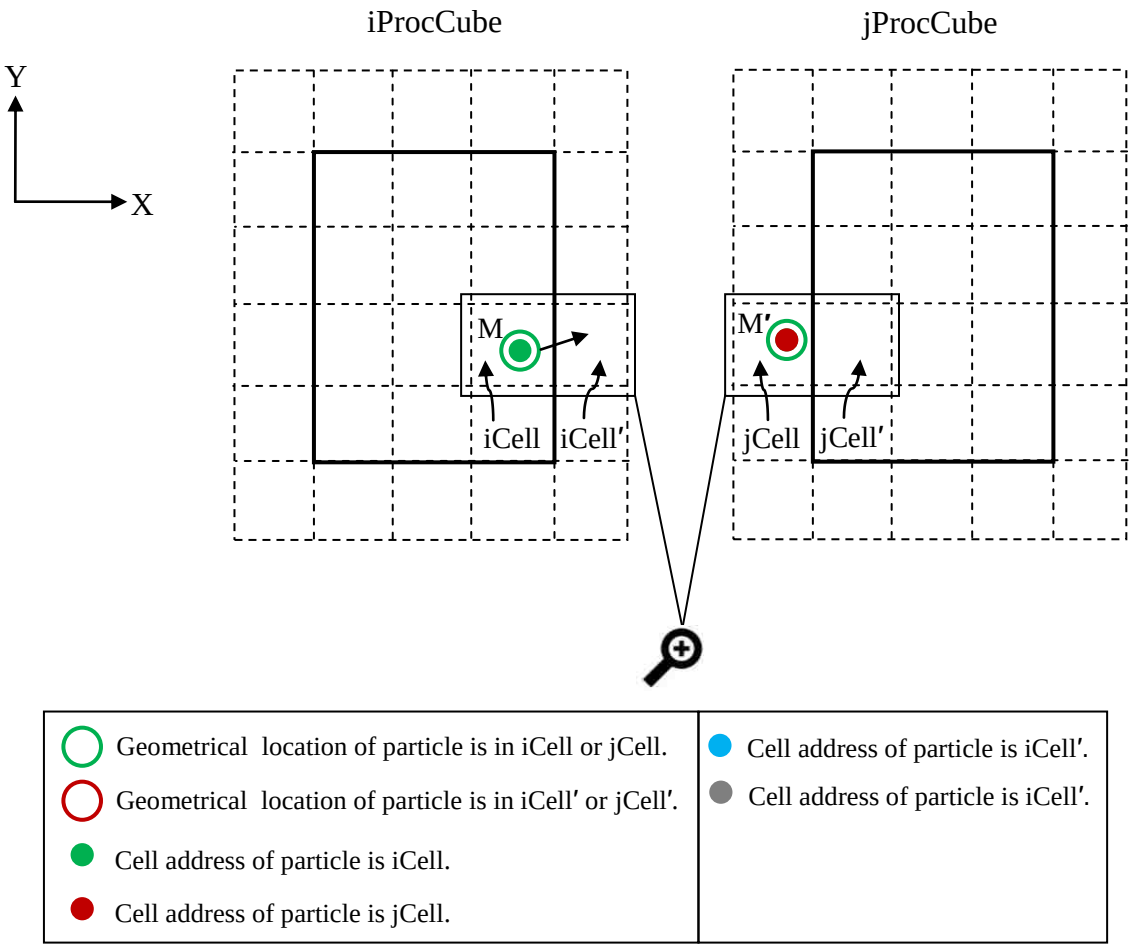


Figure 3.17 – Particle shuffling between iProcCube and jProcCube in one time step.

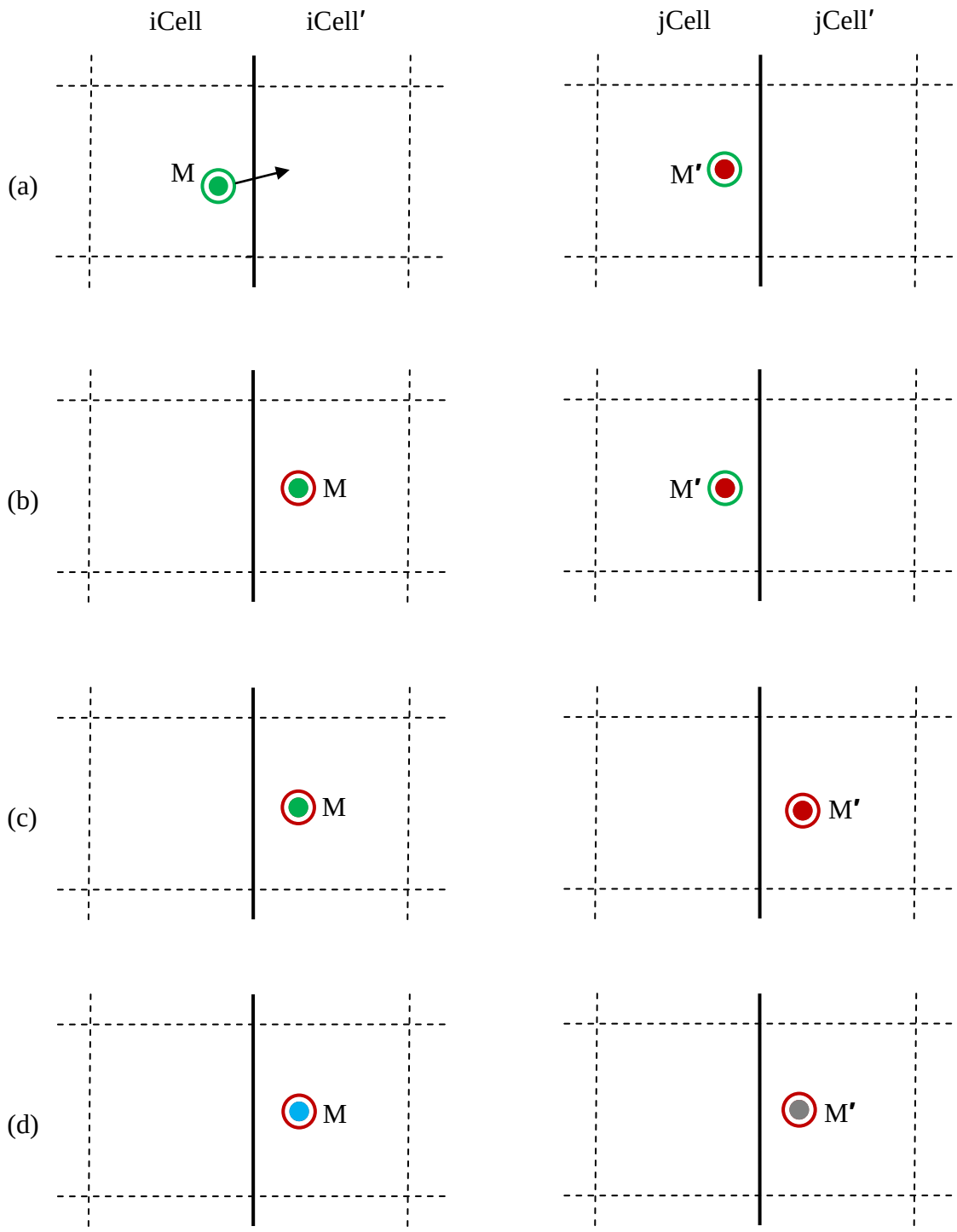


Figure 3.17 – Particle shuffling between iProcCube and jProcCube in one time step, continued. (a) Configuration of the particles before integration. Particles M and M′ have the same geometrical coordinates. (b) Configuration of the particles after integration. Particle M moves geometrically from iCell to iCell′ in iProcCube, but it still has wrong cell address (iCell). Particle M′ does not move from jCell to jCell′, however it must. (c) Configuration of the particles after message passing. Particle M′ has the same geometrical coordinates as particle M, but the wrong cell address (jCell). (d) Configuration of the particles after shuffling. Particles M and M′ have the same geometrical coordinates and correct cell addresses.

3.3.5. pdQ2 files

As discussed in previous sections, pdQ2 is designed to be a transparent, extensible, and user-friendly code for particle dynamics simulation. It is an engine that can be used for any type of particle dynamics simulation. To this end, the source code of pdQ2 is divided into two groups of files: non-user and user files, containing non-user and user subroutines respectively. All non-user files and user files are shown in Fig. 3.18.

Non-user files of pdQ2 are applicable to all particle dynamics simulations and are not changed by the user. These non-user files include the main pdQ2 driver, MPI initiation and termination routines, domain decomposition, processor communication, particle shuffling, and timing routines.

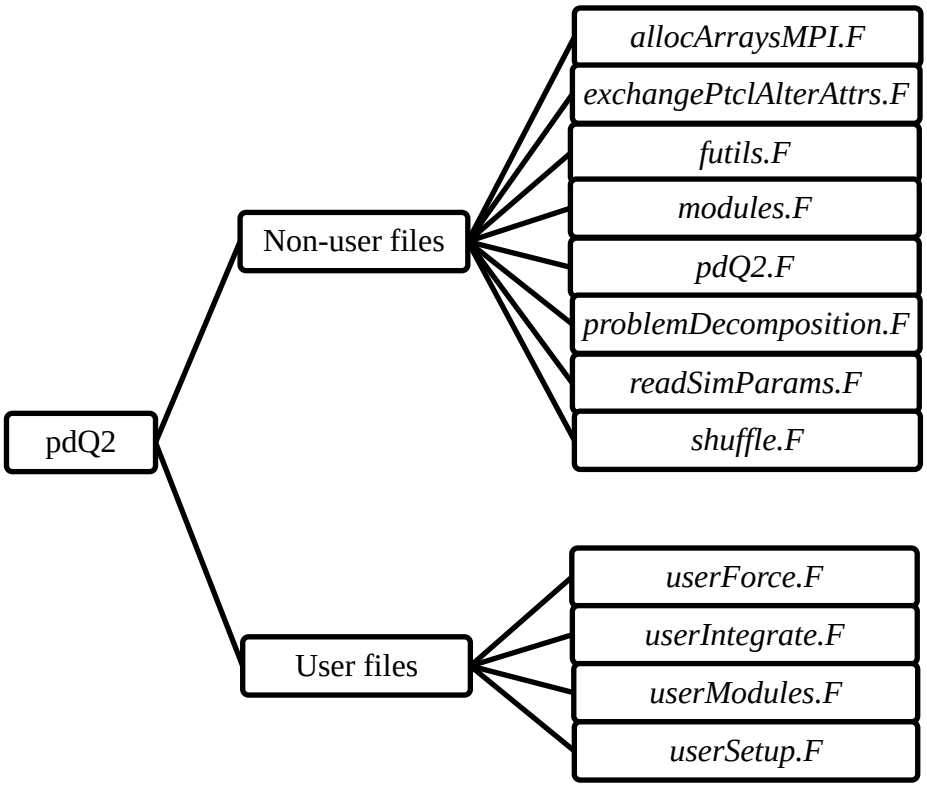


Figure 3.18 – Source code files in pdQ2.

The non-user files are briefly described next.

allocArraysMPI.F. Arrays related to message passing are allocated (subroutine *AllocArraysMPI*)

exchangePtclAlterAttrs.F. particle communications are handled using the MPI library as explained in Section 3.3.2 (subroutine *ExchangePtclAlterAttrs)*.

futils.F. MPI is initialized (subroutine *InitRTS*) and timing arrays are started (subroutine *InitTimers*). Also, code termination (subroutine *Terminate*) and array deallocations (subroutine *DeallocAll*) are accomplished by subroutines in this file.

problemDecomposition.F. The domain is decomposed and particles in procCubes and cells are initialized in the array *ptclAlterAttrs* and *ptclFixedAttrs* as described in section 3.3.1 (subroutines *ComputeDomainParams*, *ComputeProcParams*, *ProcLayout*, and *SetupProblem*).

readSimParams.F. Simulation parameters are read from the *pdQInput.dat* file on processor zero and broadcast to all other processors (subroutine *ReadSimParams*).

shuffle.F. Particle shuffling is accomplished as described in Section 3.3.3 (subroutine *Shuffle*).

The remaining four user files: *userForce.F*, *userIntegrate.F*, *userModules.F*, and *userSetup.F* are described in the Appendix. These files are used to tailor the physics and particle integration scheme to the particular problem (molecular dynamics, peridynamics, etc) of interest to the user.

3.4. Postprocessing

The pdQ2 engine outputs data files such as particle alterable attributes restart files, time history files, and timing. In addition to these files, users can generate their own output files at selected times. The user can then analyze and interpret the output files graphically using an interpreted language like MATLAB.

Particle alterable attributes restart files are stored in the *ParticleAlterAttrs.timestep.rst* file at selected time steps. In these files, particle alterable attributes of all particles are stored. Also, CPU time breakdowns for each subroutine are written to the *timing.out* file. Time history files show the history of particles in a specific time period, and it is up to the user to decide to output time history files for a specified number of particles at a specified time interval. It is stored in a file called *timehist.dat*.

3.5. Code validation and performance analysis

In this section, the runtime efficiency and numerical accuracy of pdQ2 is compared with those of pdQ. To achieve this purpose, we investigate a simple 2D benchmark peridynamic problem.

The problem considered, a 2D peridynamic linear elastic beam, is illustrated in Fig. 3.19. The cantilever beam is fixed at the left side and is loaded transversely at the right side. The size of the beam is m×s by n×s. The force radius, δ, is 3 particle spacings, s; thus each particle (for example the green particle) potentially interacts with 28 other particles (red particles).

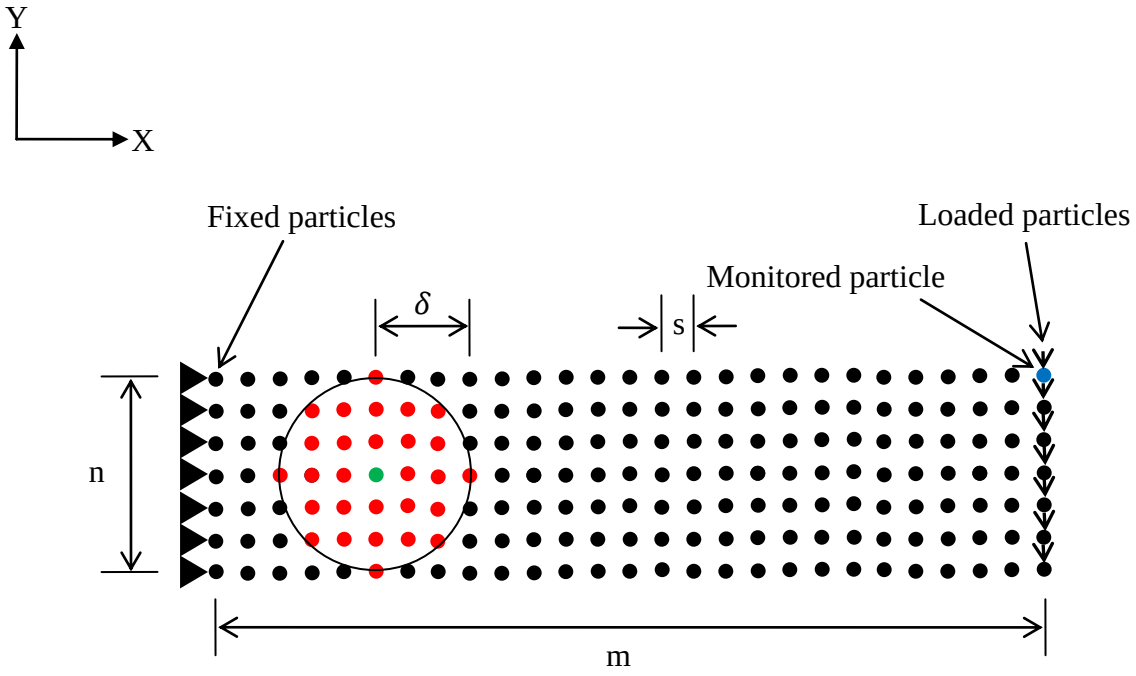


Figure 3.19 – 2D peridynamic linear elastic beam configuration.

Three performance studies were performed. First, the number of particles was held fixed and the number of processors varied. Second, the number of particles was varied and the number of processors held fixed. Third, both the number of processors and the number of particles were held fixed but the compiler flag was varied. For each case, the timings and results from pdQ2 are compared with those from pdQ as both validation (to check if the results remain unchanged) and to assess the comparative performance of the two codes. The simulation parameters are given in Table 3.1.

Simulation parameters
Spacing, s = 0.0254 m
Force radius, $\delta = 3 \times s$
Min cell size = $1.2 \times \delta$
Delta time = 3.36 E-6 sec
Load per particle = 1 N
Damping factor = 0.2
Young's modulus = 24.85 E9 N/m^2
Poisson's ratio = 0.22
Number of time steps = 200
Monitored particle ID = number of particles
OPT_PGF = -g (e.g. optimization flag)

Table 3.1 – Simulation parameters.

3.5.1. Fixed number of particles; varying the number of processors

A specific problem with m = 721, and n = 309 for a total of 223,820 particles was simulated using pdQ and pdQ2 using varying numbers of processors. The numbers of processors in the X and Y directions (denoted nPX and nPY) was varied. The monitored particle is indicated in blue in Fig. 3.19. The timing performance is shown in Fig. 3.20. We see in Fig. 3.20 that with one processor, pdQ is slightly faster than pdQ2. On the other hand, with 32 processors, pdQ2 is about four times as fast as pdQ.

The position of the monitored particle in the Y direction, at the end of the simulation, is shown in Table 3.2 to compare the accuracy of the two codes. We see some differences in the 13th significant digit; the reason for this difference is unknown and needs to be further investigated.

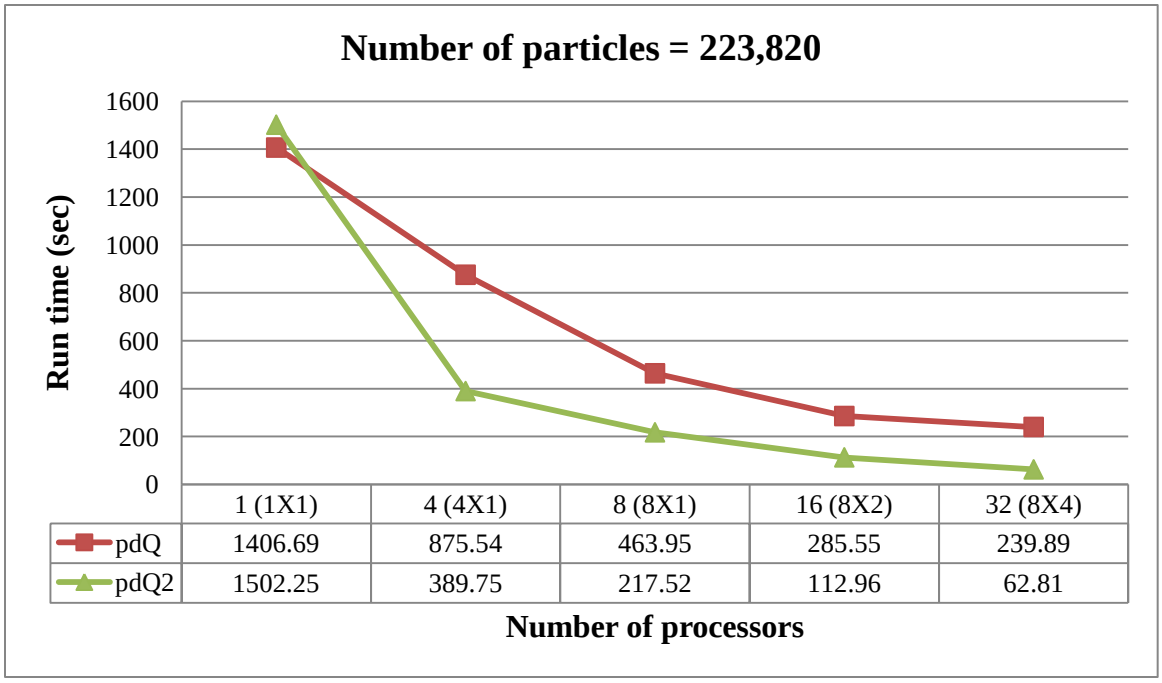


Figure 3.20 – Performance analysis of pdQ and pdQ2 using 223,820 particles.

	Y position of monitored particle at the last time step (m)	
Number of processors	**pdQ**	**pdQ2**
1 (nPX = 1, nPY = 1)	7.848600129112179	7.848600129111611
4 (nPX = 4, nPY = 1)	7.848600129112179	7.848600129111607
8 (nPX = 8, nPY = 1)	7.848600129112179	7.848600129112170
16 (nPX = 8, nPY = 2)	7.848600129112179	7.848600129112170
32 (nPX = 8, nPY = 4)	7.848600129112179	7.848600129112170

Table 3.2 – Numerical validation of pdQ2 vs. pdQ using 223,820 particles (differences shown in red).

3.5.2. Fixed number of processors; varying the number of particles

Fixing the number of processors at 32 (nPX = 8, nPY = 4), the number of particles is varied. The smallest simulation has 223,820 particles and largest has 1,122,892 particles. The performance analysis is illustrated in Fig. 3.21 and the Y position of the monitored particle at the last time step is given in Table 3.3. We mainly see some differences in the 10^{th} significant digit of the example with 659,680 particles that needs to be further studied. As seen in Fig. 3.21, pdQ2 is about four times as fast as pdQ using 32 processors.

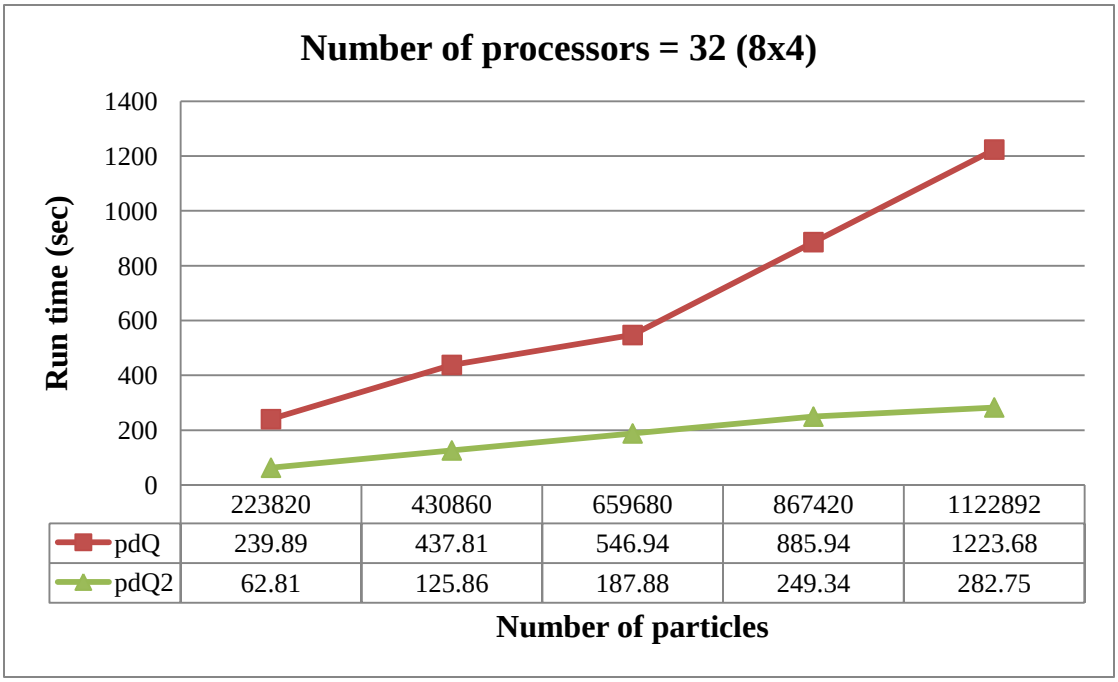


Figure 3.21 – Performance analysis of pdQ and pdQ2 using 32 processors. 8 processors in X direction and 4 processors in Y direction have been used.

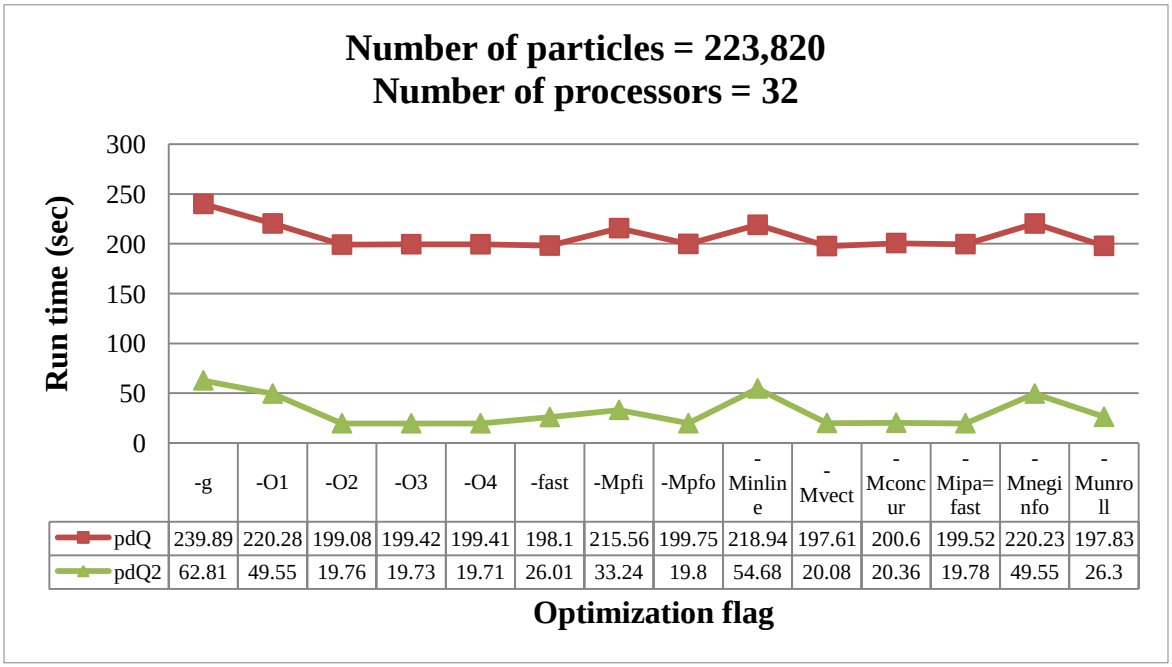
	Y position of monitored particle at the last time step (m)	
Number of particles	**pdQ**	**pdQ2**
223,820 (m = 721, n = 309)	7.848600129112179	7.848600129112170
430,860 (m = 1,001, n = 429)	10.89660012938630	10.89660012938630
659,680 (m = 1,239, n = 531)	13.487400196513 48	13.487400128802 17
867,420 (m = 1,421, n = 609)	15.46860012889683	15.46860012889683
1,122,892 (m = 1,617, n = 693)	17.602200129564 26	17.602200129564 04

Table 3.3 – Numerical comparison of pdQ and pdQ2 using 32 processors (differences shown in red).

3.5.3. Performance analysis using different optimization flags

As performance is a very important issue in parallel programs, it is important to optimize performance with respect to different compiler flags.

Both pdQ and pdQ2 were compiled with the PGI compiler [User's Guide 2012] and fourteen compiler optimization options were investigated. To do the performance study using different flags, a specific problem with 223,820 particles (m = 721, n = 309) and 32 processors (nPX = 8, nPY = 4) was investigated. The performance results using the fourteen different optimization flags are shown in Fig. 3.22. For validation purposes, the Y position of the monitored particle at the last time step is shown in Table 3.4.



	-g	-O1	-O2	-O3	-O4	-fast	-Mpfi	-Mpfo	-Minline	-Mvect	-Mconcur	-Mipa=fast	-Mnegiinfo	-Munroll
pdQ	239.89	220.28	199.08	199.42	199.41	198.1	215.56	199.75	218.94	197.61	200.6	199.52	220.23	197.83
pdQ2	62.81	49.55	19.76	19.73	19.71	26.01	33.24	19.8	54.68	20.08	20.36	19.78	49.55	26.3

Figure 3.22 – Performance analysis of pdQ and pdQ2 using different optimization flags.

As seen in Fig. 3.22, the optimization flag, "-Mvect", gives the most efficient timing for pdQ, and "-O4" leads to the highest efficiency for pdQ2. As shown in Table 3.5, the Y position of the monitored particle does not change when either pdQ or pdQ2is compiled with different optimization flags.

	Y position of monitored particle at the last time step (m)	
Optimization	**pdQ**	**pdQ2**
-g	7.848600129112179	7.848600129112170
-O1	7.848600129112179	7.848600129112170
-O2	7.848600129112179	7.848600129112170
-O3	7.848600129112179	7.848600129112170
-O4	7.848600129112179	7.848600129112170
-fast	7.848600129112179	7.848600129112170
-Mpfi	7.848600129112179	7.848600129112170
-Mpfo	7.848600129112179	7.848600129112170
-Minline	7.848600129112179	7.848600129112170
-Mvect	7.848600129112179	7.848600129112170
-Mconcur	7.848600129112179	7.848600129112170
-Mipa=fast	7.848600129112179	7.848600129112170
-Mneginfo	7.848600129112179	7.848600129112170
-Munroll	7.848600129112179	7.848600129112170

Table 3.4 – Numerical comparison of pdQ and pdQ2 using 223,820 particles and 32 processors (differences shown in red).

3.5.4. Understanding the performance differences between pdQ and pdQ2

To understand the reasons for the observed performance differences between pdQ and pdQ2, we need to study the timings of both codes in detail. In Fig. 3.23 and 3.24, the timings of pdQ and pdQ2 are broken down into “Communication”, “Force computation”, and “other”.

Comparing the performance details of pdQ and pdQ2, it is seen that pdQ2 spends about the same amount of time on force computation as pdQ, on both single and multiple processors. However, on multiple processors, pdQ spends much more time on communication than pdQ2.

The reason for the significant differences between the communication timings of pdQ and pdQ2 can be understood by counting the number and size of the messages passed in one time step on a specific processor.

Consider the problem with 223,280 particles that ran on 16 processors. The size and number of the messages sent from processor number 8 to other processors, for instance, are tabulated in Table 3.5.

Also, slight difference is observed between pdQ and pdQ2 timings on single processor. This can be due to the method that *ptclAlteAttrs*, *ptclFixedAttrs*, and *ptclIntegAtrs* arrays are dimensioned. In pdQ, they are 2 dimensional arrays; however, in pdQ2, they are 5 dimensional arrays.

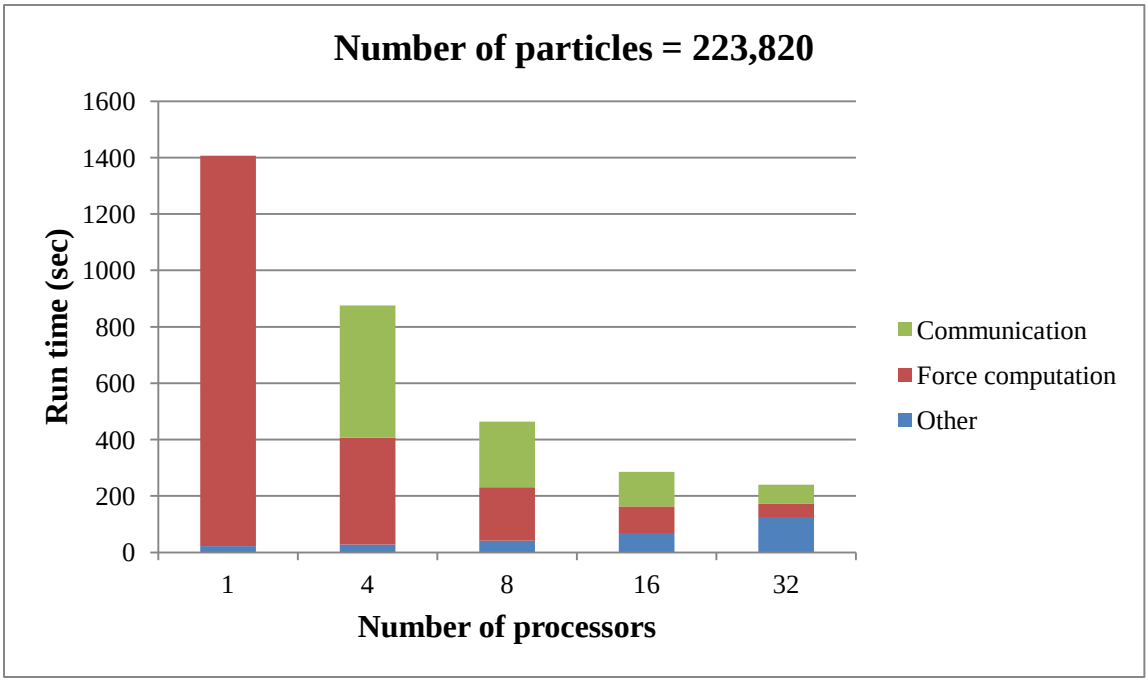


Figure 3.23 – Performance details for pdQ.

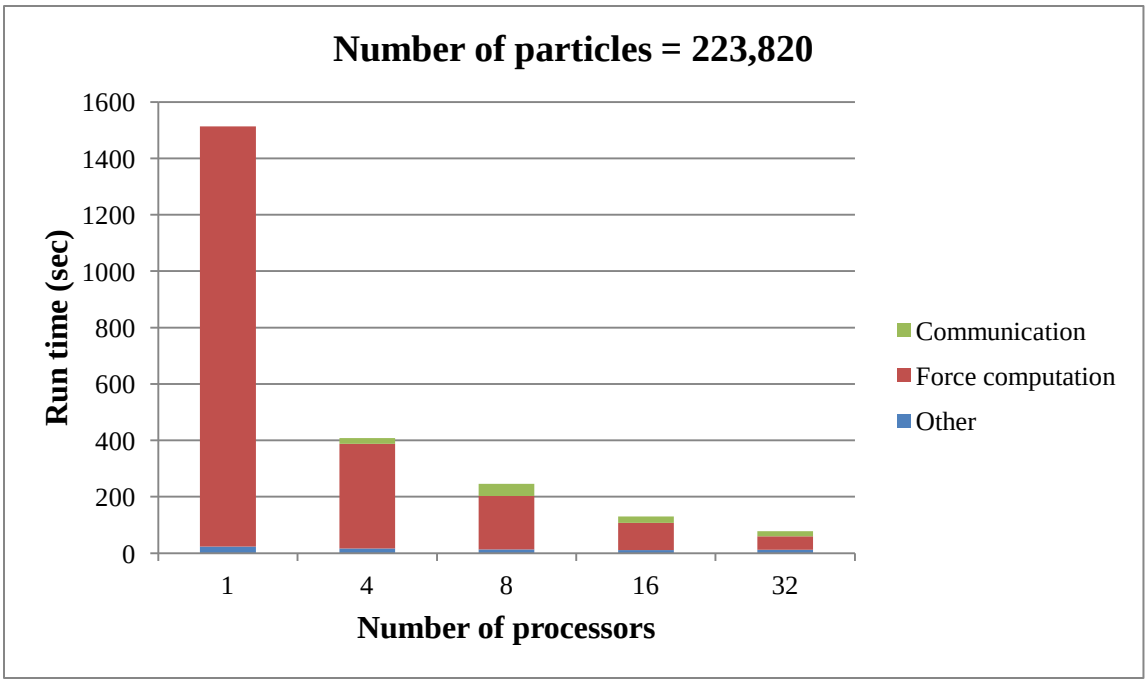


Figure 3.24 – Performance details for pdQ2.

Message passing details in one time step for processor number 8			
Number of sent messages		**Size of sent messages (MB)**	
pdQ	**pdQ2**	**pdQ**	**pdQ2**
19	4	37.0656	0.21788

Table 3.5 – Message passing details for processor number 8. For 2D peridynamic linear elastic beam using 223,820 particles.

As shown in Table 3.5, the number and size of messages sent by a particular processor are much less for pdQ2 than for pdQ.

3.6. Summary

In this chapter, the design and implementation of the pdQ2 parallel particle dynamics code has been described. The concepts of cell, wall, core, cell block, skin, and procCube have been defined for particle partitioning, communication, and shuffling.

The efficiency of pdQ2 with respect to pdQ has been investigated through several timing performance analyses. It is shown that pdQ2 is about four times as fast as pdQ, at least for the particular 2D problem studied. Also, the performance of pdQ and pdQ2 using different optimization flags has been studied and compared.

In the next chapter, a newly-developed particle dynamics model for quasi-brittle structures, the micropolar peridynamic lattice model is implemented using pdQ2.

4. MICROPOLAR PERIDYNAMIC LATTICE MODEL FOR QUASI-BRITTLE STRUCTURES

4.1. Introduction

Continuum mechanics is an awkward model for modeling concrete and other quasi-brittle structures, because concrete deformation, in most regimes of interest, is inherently discontinuous, and at scales smaller than the aggregate size, concrete is no longer a homogeneous material. The peridynamic model has not entirely discarded the continuum paradigm, in that the material space continues to be idealized as continuous and thus further *ad hoc* discretization choices need be made to implement peridynamics computationally [Gerstle *et al.* 2012].

In this chapter, we discard the continuum material concept completely, and regard the material space as a discrete lattice of particles. By using close-packed particle lattices, the "micropolar peridynamic lattice model (MPLM)" is able to capture the major features of quasi-brittle materials, including large-deformation elasticity, anisotropic damage, and fracture. With the MPLM, rather than viewing the model as a collection of beam or truss elements connected together at nodes (as with traditional truss- and beam-analogy lattice models), the model is viewed as a collection of interacting point masses (as with the peridynamic models). The material constitutive behavior is captured via inter-particle force vectors that are functions of relative particle positions and velocities and their histories. The MPLM uses a finite number of regularly-spaced interacting particles of finite mass, rather than an infinite number of infinitesimal particles. Thus the MPLM is more straightforward for computational implementation, because fewer arbitrary discretization decisions need be made. Additionally, the MPLM is conceptually simpler

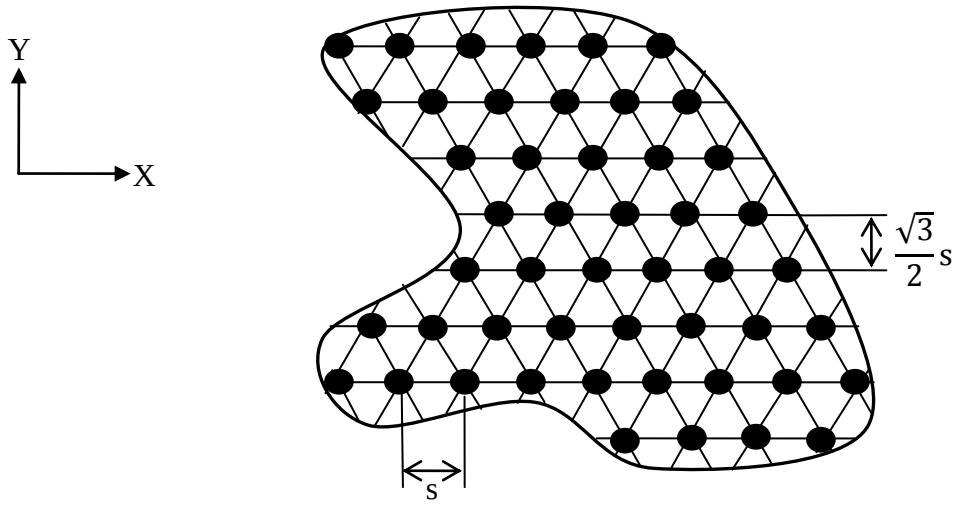
than both the truss-analogy lattice model and the original peridynamic model, not to mention classical finite element methods [Gerstle *et al*. 2012].

In Section 4.2, the MPLM is defined. A constitutive model for concrete is developed and calibrated in Section 4.3, and its use is demonstrated using several example problems in Section 4.4.

4.2. Micropolar peridynamic lattice model (MPLM)

In Fig. 4.1, a 2D close-packed particle lattice is shown. Each particle is spaced a distance, *s*, from its six nearest neighbors. To create a 3D face-centered cubic lattice, the layer shown in Fig. 1 is replicated repeatedly with a stride of $\frac{1}{2}s$ in the X direction, a stride of $\frac{\sqrt{3}}{2}s$ in the Y direction, and a stride of $\frac{\sqrt{2}}{3}s$ in the Z direction.

The material volume occupied by each particle is $\Delta V = sA$ for a 1D lattice representing an axial member with cross-sectional area, A, $\Delta V = \frac{\sqrt{3}}{2}s^2 t$ for a 2D hexagonal lattice representing a flat plate of thickness, t, and $\Delta V = \frac{s^3}{\sqrt{2}}$ for the 3D lattice representing a solid. To represent a material with a mass density of ρ, each particle is endowed with a mass of $\Delta m = \rho \Delta V$. Assuming that each particle is represented by a solid sphere with radius, *r*, its mass moment of inertia, ΔI, identical about all axes through the particle, is $\Delta I = \frac{2}{5}\Delta m r^2$ [Gerstle *et al*. 2012].



Lattice strides: $\frac{1}{2}s$ in X, $\frac{\sqrt{3}}{2}s$ in Y, and $\sqrt{\frac{2}{3}}s$ in Z direction

Figure 4.1 – Hexagonal lattice in 2D (after [Gerstle *et al.* 2012]).

The particle lattice spacing, *s*, can reasonably be chosen as the material grain characteristic size (such as the maximum aggregate size for concrete). Alternately, the spacing *s* may be chosen based upon the requirement that the number of particles used for a particular problem not exceed the capacity of the computational resource. For a material like concrete, it makes no sense to allow the particle lattice spacing to be less than the aggregate size; the mesoscale of the material sets a lower bound on appropriate lattice particle spacing. Indeed, it makes no sense to define geometric features that are smaller than the aggregate size, as even if a structure with such small features could be constructed, these tiny features could hardly be considered as consisting of a spatially homogenous material. Thus, within the MPLM, a "perfectly sharp crack" and a "perfectly sharp corner" are meaningless features, impossible to express [Gerstle *et al*. 2012].

With the material mass represented by particles in a lattice, Newton's second law of motion is applied to each particle *i*:

$$\sum_{j=1}^{N_i} \vec{F}_{i\,j} + \vec{F}_{Ext\,i} = \Delta m \ddot{\vec{u}}_i \ , \tag{4.1}$$

and

$$\sum_{j=1}^{N_i} \vec{M}_{i\,j} + \vec{M}_{Ext\,i} = \Delta I \ddot{\vec{\theta}}_i \ , \tag{4.2}$$

where $\vec{F}_{i\,j}$ and $\vec{M}_{i\,j}$ are the force and moment vectors, respectively, exerted by particle *j* on particle *i*, $\vec{F}_{Ext\,i}$ and $\vec{M}_{Ext\,i}$ are the externally applied force and moment vectors, respectively, applied to the centroid of particle *i*, and $\ddot{\vec{u}}$ and $\ddot{\vec{\theta}}$ are the linear and angular acceleration vectors, respectively, of the centroid of particle *i*. N_i is the number of particles, *j*, that are within the spherical neighborhood, whose radius is the material horizon, δ, of particle *i*. With a close-packed lattice, and $\delta = 1.5s$, N_i is two (or less) for a 1D problem, N_i is six (or less) for a 2D problem, and N_i is eighteen (or less) for a 3D problem. In the three-dimensional case, each particle, *i*, is surrounded by 12 nearest neighbors, at distance *s* from particle *i*, and 6 second-nearest neighbors, at a distance $\sqrt{2}s$ from particle *i*. For the 3D case, it has been shown that all 18 particles must be considered as interacting neighbors of particle *i* if isotropic classical elasticity is to be well-approximated by a hexagonal close-packed lattice model [Rahman 2012]. Of course, one could contemplate MPLM models with larger material horizons, which might be preferable from the point of view of isotropic damage behavior with respect to lattice orientation, but the number of neighboring particles, N_i, and thus the number of force computations would be larger, per particle [Gerstle *et al*. 2012].

In our implementation, Eqs. 1 and 2 are integrated explicitly in time using a simple Verlet integration method, with time step, $\Delta t = \frac{s}{nc_0}$, with s being the particle spacing, c_0 being the speed of sound in the material, and n being π or greater for stability. Because we are interested in modeling cementitious materials, with highly nonlinear material behavior, explicit time integration, with the required small time steps, is the method of choice.

The pairwise functions $\vec{F}_{ij}$ and $\vec{M}_{ij}$ describe the internal forces and moments between neighboring lattice particles, and from these functions, the material behavior emerges. For a bond-based micropolar peridynamic model, the pairwise functions are chosen to be functions of the reference position vectors, and current position vectors, and also as functions of the velocities of particles *i* and *j*. Note that all of these kinematic vectors include particle positions and velocities as well as particle rotations and angular velocities. In the bond-based damage model, the pairwise functions also depend upon evolving damage parameters, ω_{ij}, associated with the interaction between particles *i* and *j* [Gerstle *et al*. 2012].

For a state-based peridynamic model [Silling *et al*. 2007], the pairwise functions depends not only upon the states of particles *i* and *j*, but also upon the states of all other particles, *k*, and also upon the interaction damage states, ω_{ij}, within the peridynamic horizon of particle *i*.

With today's high performance parallel computers and using a code such as pdQ2 as described in the previous chapters, a million particles can easily be modeled. Thus, on a parallel computer, it is feasible to simulate large 3D concrete structures using the

MPLM – not just small laboratory specimens. 2D simulations as demonstrated in Section 4.4 are entirely practical on single-processor computers.

The MPLM, as presented here, is a suitable material model for solids, but not for liquids and gases. In solid models, the forces between particles are assumed to arise due to deviations from a reference state. As long as the deviation of particle positions from their reference locations is not too extreme, the MPLM is appropriate.

On the other hand, with gases and liquids the forces between particles depend upon particle current locations and velocities, but not upon a particle reference configuration. Molecular dynamics has been used to model liquids, but only at extremely small size and at short time scales [Gerstle *et al*. 2012].

In the next section, the material constitutive models are defined for a quasi-brittle material.

4.3. MPLM constitutive model for concrete

The MPLM constitutive model for concrete is described in the following three subsections. Although it is specialized for concrete, it is certainly possible to construct analogous constitutive model for other quasi-brittle materials such as ceramics, bone, and so on. In Section 4.3.1, the linear elastic model for MPLM is explained, in Section 4.3.2, the MPLM damage model is elaborated, and in Section 4.3.3, the damping model is described.

4.3.1. Linear elastic model

Assume a large material domain, with minimum characteristic size *D*. The domain is represented by a 1D, 2D, or 3D close-packed particle lattice with spacing, *s*, with $s \ll D$. Within the classical theory of elasticity, away from stress singularities and domain boundaries, the static strain field is approximately spatially homogeneous within a neighborhood of size *s*. By equating the strain energy stored within a particle volume, ΔV, of the MPLM to the strain energy stored within an equivalent volume of the classical elasticity model, we calculate the linear relationship between particle force components acting between a pair of particles, *i* and *j*, with the particle displacement components depicted in Fig. 4.2 [Rahman 2012]. The force vector acting between particles *i* and *j* is termed as "interaction *ij*".

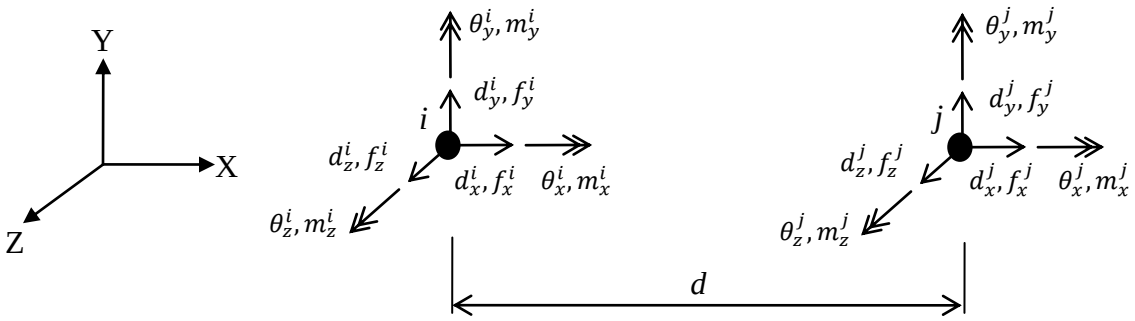


Figure 4.2 – Displacement and force components, in local coordinates, acting between particles *i* and *j*, separated by distance, *d* (from [Gerstle *et al.* 2012]).

For 1D and 2D plane-strain and plane-stress problems, the interaction *ij* has length, *d*, equal to the lattice spacing, *s*, and the stiffness relation of interaction *ij* is

$$\begin{Bmatrix} f_x^i \\ f_y^i \\ m_z^i \\ f_x^j \\ f_y^j \\ m_z^j \end{Bmatrix} = \begin{bmatrix} \frac{a}{s} & & & & & \\ & \frac{12b}{s^3} & \frac{6b}{s^2} & & \textit{Sym.} & \\ & \frac{6b}{s^3} & \frac{4b}{s} & & & \\ -\frac{a}{s} & & & \frac{a}{s} & & \\ & -\frac{12b}{s^3} & -\frac{6b}{s^2} & & \frac{12b}{s^3} & -\frac{6b}{s^2} \\ & \frac{6b}{s^2} & -\frac{2b}{s} & & -\frac{6b}{s^2} & \frac{4b}{s} \end{bmatrix} \begin{Bmatrix} d_x^i \\ d_y^i \\ \theta_z^i \\ d_x^j \\ d_y^j \\ \theta_z^j \end{Bmatrix}, \tag{4.3}$$

where s is the lattice spacing, and a and b are the micropolar peridynamic elastic constants. For 2D "frame-type" problems,

$$a = EA \text{ and } b = EI, \tag{4.4}$$

where A is the cross-sectional area of the frame, I is the moment of inertia of the cross sectional area about the centroidal out-of-plane axis, and E is Young's modulus [Rahman 2012]. For 2D plane stress problems,

$$a = \frac{Est}{\sqrt{3}(1-\nu)} \text{ and } b = \frac{Es^3(1-3\nu)t}{12\sqrt{3}(1-\nu^2)}, \tag{4.5}$$

and for 2D plane strain problems,

$$a = \frac{Est}{\sqrt{3}(1-\nu^2)} \text{ and } b = \frac{Es^3(1-4\nu)t}{12\sqrt{3}(1-\nu^2)}, \tag{4.6}$$

where t is the thickness of the geometric domain in the Z direction, E is Young's modulus, and ν is Poisson's ratio [Rahman 2012].

Thus it is seen that the isotropic MPLM, with three material parameters, *s*, *a*, and *b*, is approximately equivalent to the classical theory of elasticity, with material parameters *E* and *v*. The difference is that with MPLM, a length scale, *s*, has been introduced. This length scale is crucial for the model to be extended to model strain-softening behavior, and consequent fracture.

In the same procedure used for 2D solids, the stiffness relation of interaction *ij* for 3D solid problems can be derived [Rahman 2012]. Note that the stiffness relation in Eq. 4.3 is in local coordinates, and must be transformed into global coordinates for determining the forces acting upon the particles for subsequent time integration.

For many quasi-brittle structures, it is sufficient to assume small-deformation behavior, with small particle and interaction rotations. In this case the force interaction geometry between particles can be assumed constant over the course of the simulation time, and the elastic stiffness matrix of each interaction need only be calculated once for each type of interaction before the time integration loop is entered. In this case, computation of the elastic stiffness relations is trivial compared to time history damage computations.

However, for some analysis regimes, it is necessary to update the particle locations in each time step to account for finite interaction rotations. This geometrically nonlinear analysis is slightly more computationally expensive, while allowing for large deformation (but still small strain) behavior to be computed. In our computational formulation, elastic interaction deformations (interaction stretches and curvatures) are assumed to be reasonably small, but large translations and rotations are accounted for using a co-rotational stiffness formulation [Crisfield 1991 and Yaw 2008]. Thus,

damaged fragments can detach as rigid bodies and move correctly with large translations and rotations. However, collision behavior is not currently incorporated into the model, except between adjacent particles in the reference lattice. The co-rotational formulation, allowing changing elastic stiffness caused by changing geometry, can be employed to facilitate modeling of compression and shear-band failures [Gerstle *et al*. 2012].

4.3.2. Damage model

With reference to Fig. 4.2, the micropolar axial stretch of interaction *i j*,

$$\varepsilon_a = \frac{d_t - d}{d} \ , \tag{4.7}$$

where d_t is the current distance and d is the reference distance between two particles, is defined in a manner similar to axial strain.

Similarly, the maximum micropolar curvatures about the local Z, Y and X axes, respectively, of interaction *ij* are [Gerstle *et al*. 2013]

$$\psi_x \equiv \left| \frac{\theta_x^j - \theta_x^i}{d} \right| \ , \tag{4.8}$$

$$\psi_y \equiv \max\left[\left| \frac{2}{d}\left(2\theta_y^i + \theta_y^j - \frac{3}{d}\left(d_z^j - d_z^i \right) \right) \right|, \ \left| \frac{2}{d}\left(2\theta_y^j + \theta_y^i - \frac{3}{d}\left(d_z^i - d_z^j \right) \right) \right| \right] \ , \tag{4.9}$$

$$\psi_z \equiv \max\left[\left| \frac{2}{d}\left(2\theta_z^i + \theta_z^j - \frac{3}{d}\left(d_y^j - d_y^i \right) \right) \right|, \ \left| \frac{2}{d}\left(2\theta_z^j + \theta_z^i - \frac{3}{d}\left(d_y^i - d_y^j \right) \right) \right| \right] \ , \tag{4.10}$$

The measures of micropolar tensile and compressive interaction deformation are defined as

$$\varepsilon_{mp+} = \varepsilon_a + \beta d \sqrt{\psi_x^2 + \psi_y^2 + \psi_z^2} \ , \tag{4.11}$$

$$\varepsilon_{mp-} = \varepsilon_a - \beta d\sqrt{\psi_x^2 + \psi_y^2 + \psi_z^2}\,, \tag{4.12}$$

where β is a dimensionless parameter [Gerstle *et al*. 2013].

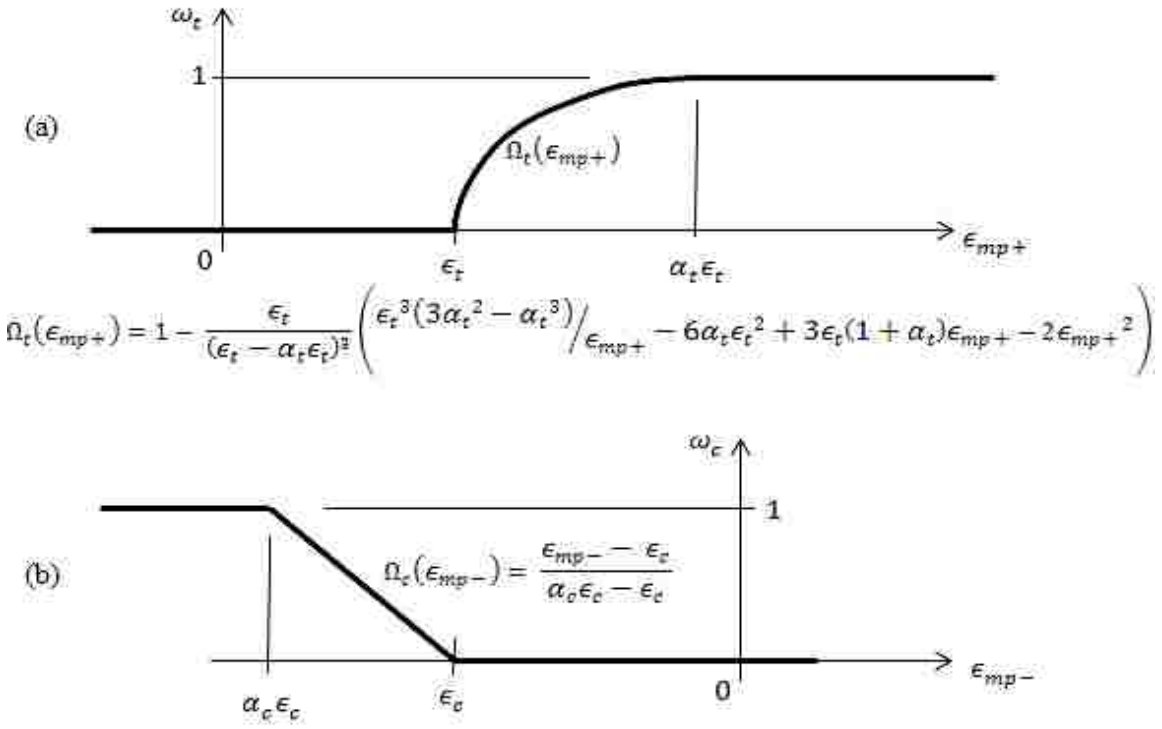


Figure 4.3 – (a) Damage, ω_t, versus the micropolar strain measure, ε_{mp+}. (b) Damage, ω_c, versus the micropolar strain measure, ε_{mp-}. (ω_t and ω_c never decrease with time.) (from [Gerstle *et al*. 2013]).

The tensile damage parameter, ω_t, is defined in terms of these deformation measures, with reference to Fig. 4.3, as follows.

For tension damage, ω_t:

$$0 \le \varepsilon_{mp+} < \varepsilon_t, \qquad \omega_t = \max(0, \omega_{t\,prev}), \tag{4.13}$$

$$\varepsilon_t \le \varepsilon_{mp+} < \alpha_t\varepsilon_t, \qquad \omega_t = \max(\Omega_t(\varepsilon_{mp+}), \omega_{t\,prev}), \tag{4.14}$$

$$\alpha_t\varepsilon_t \le \varepsilon_{mp+}, \qquad \omega_t = 1, \tag{4.15}$$

where $\omega_{t\,prev}$ is the value of the tensile damage parameter for interaction *ij* in the immediately preceding time step. The damage function $\Omega_t(\varepsilon_{mp+})$ is defined in Fig. 4.3(a), and it has been chosen in such a way that the cohesive tensile softening behavior is modeled approximately correctly.

For the evolution of compression damage, ω_t :

$$\varepsilon_{mp-} \le \alpha_c \varepsilon_c, \qquad \omega_c = 1 \ , \tag{4.16}$$

$$\alpha_c \varepsilon_c \le \varepsilon_{mp-} \le \varepsilon_c, \quad \omega_c = \max(\Omega_c(\varepsilon_{mp-}), \omega_{c\,prev}) \ , \tag{4.17}$$

$$\varepsilon_c \le \varepsilon_{mp-}, \qquad \omega_c = \max(0, \omega_{c\,prev}) \ , \tag{4.18}$$

where $\omega_{c\,prev}$ is the value of the compressive damage parameter for interaction *ij* in the immediately preceding time step. Function $\Omega_c(\varepsilon_{mp-})$ is defined in Fig. 4.3(b).

The damage parameter, ω, is computed as the maximum of ω_t and ω_c.

If $\varepsilon_a \ge 0$, then

$$\{f\}=(1-\omega)[K]\{d\} \ , \tag{4.19}$$

and if $\varepsilon_a < 0$, then

$$\{f\}=(1-\omega)[K^*]\{d\} \ , \tag{4.20}$$

where *{f}* is the force vector acting between particles *i* and *j*, *[K]* is the elastic stiffness matrix defined using Eq. 4.3, *{d}* is the vector of particle deformations, associated with interaction *ij*. Because there are many interactions per particle, this form allows damage to be anisotropic.

With the stiffness matrix, $[K^*]$, the axial components of force are the same as that

computed by *[K]*, but the shears and moments are reduced by the damage parameter $(1-\omega)$ Thus, compression failure is indirectly precipitated by loss of moment and shear capacity (and subsequent instability due to nonlinear geometric effects), but not by loss of axial stiffness. In this implementation, damage can be either tensile or compressive, but not both [Gerstle *et al*. 2013].

The constitutive model presented has eight parameters: peridynamic lattice spacing parameter *s*, micro-elastic stiffness parameters *a* and *b*, and the parameters governing tensile and compressive damage evolution: $\varepsilon_t, \alpha_t, \varepsilon_c, \alpha_c$, and β.

The lattice spacing parameter, *s*, is chosen to be as small as the available computational capacity allows, but no less than the largest material grain size.

The parameter ε_t is calibrated to reproduce the tensile strength, f_t, of the concrete: $\varepsilon_t \approx \frac{f_t}{E}$.

The parameter, $\beta \approx 0.1$, is chosen to replicate the ratio of uniaxial compressive load to uniaxial tensile load, usually around 10.0, as is observed empirically for normal-strength concrete.

The parameter $s_c \approx 0.001$ is chosen to replicate the strain at which uniaxial compressive failure commences, and $\alpha_c s_c \approx 0.003$ is chosen to represent the ultimate compressive strain [Gerstle *et al*. 2013].

The parameter α_t is chosen to replicate the tensile fracture energy, G_f, of the material, as described in [Gerstle *et al*. 2012].

In dynamic analysis, it is also necessary to represent material damping, as described in the next section.

4.3.3. Damping model

In dynamic MPLM simulations, damage events can release sudden bursts of acoustic energy. If no material damping is included in the model, this acoustic energy can cause spurious vibration and consequent damage. Thus a material damping model is incorporated. When computing the force in interaction *ij*, the relative translational velocity vector, $\vec{V}_{ij}$, between particles *i* and *j* is computed. The damping force, $\vec{f}_{Damp\ ij}$, between the two particles is given by

$$\vec{f}_{Damp\ ij} = 2\zeta m\omega_n \vec{V}_{ij}, \text{ and} \tag{4.21}$$

$$\vec{F}_{ij} = \vec{f}_{Elast\ ij} + \vec{f}_{Damp\ ij}, \tag{4.22}$$

where ζ is the ratio of critical damping, with value set between 0 and 1, *m* is the particle mass, ω_n is the highest natural frequency of vibration, $\vec{V}_{ij}$ is the relative velocity between particles *i* and *j*, $\vec{f}_{Elast\ ij}$ is the elastic inter-particle force calculated in the previous section, including the effect of damage, and $\vec{F}_{ij}$ is the internal force vector used in Eq. 4.1. The damping force, always opposing the direction of motion, removes energy from the system. It is found that choosing $\zeta \approx 0.5$ produces reasonable damping behavior, and sufficient accuracy is provided with critical time step

$\Delta t_{Crit} = \left(\frac{2}{\omega_n} \right) \left(\sqrt{1+\zeta^2} - \zeta \right)$. Without going into detail, a similar strategy can be used to damp shear and rotational degrees of freedom [Gerstle *et al*. 2013].

4.3.4. Modeling of reinforcing bars and bond

A reinforcing bar is represented as a 1D lattice of MPLM particles representing a bar with cross-sectional area A_s and cross-sectional moment of inertia I_s. The material parameters are Young's modulus E_s and yield stress F_y. Steel particles interact if they spaced less than *s* from each other. Steel particles from separate reinforcing bars do not interact [Gerstle *et al.* 2013].

As shown in Fig. 4.4, only every other steel particle of a given rebar is connected to concrete particles within a horizon *s* using the same elastic interaction model as for concrete-concrete particles (such interactions assume no damage). The reason that only every other steel particle is connected to concrete is to allow cracks in concrete to develop unhindered by the non-damaged steel-concrete interactions. If the distance between a steel particle and a concrete particle is zero, the interaction between these two particles is ignored [Gerstle *et al.* 2013].

Bond-slip is indirectly modeled and emerges from the elasticity and damage of the interactions between steel particles and the surrounding the concrete particles.

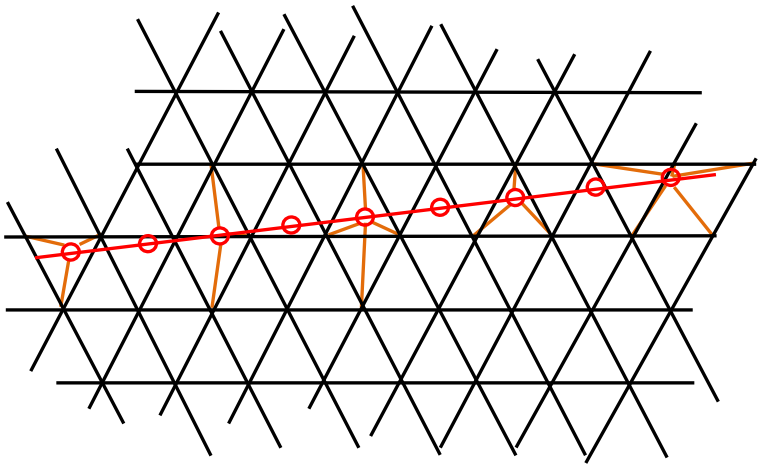


Figure 4.4 – Bond of reinforcement (red) to concrete (black) using peridynamic interactions (tan) (from [Gerstle *et al.* 2013]).

4.4. Examples

In this section, several examples are run using pdQ2 on both single- and multi-processors. In all of the examples, the target classical material parameters are shown in Table 4.1, and the corresponding selected MPLM parameters for concrete and steel are shown in Tables 4.2 and 4.3. The time step is chosen as $\Delta t = \frac{s}{24 c_{0(steel)}} = 1.808 \times 10^{-7} s$. In each example, the load was linearly ramped from zero to the peak load for duration of at least four fundamental periods of the structure, and was thus essentially quasi-static. The load is ramped from time zero up to 75% of the total simulation time and then held constant.

Strength is defined as the peak load at which static equilibrium can still be achieved. In all of the following examples, the particles in the deformed configuration are shown and the damaged interactions are color-coded as shown in Fig. 4.5. Undamaged

interactions are not shown in the figures.

The steel-concrete MPLM interactions are identical to concrete-concrete interactions, except that they were assumed to be linear elastic, with no damage. The steel was modeled as elastic-perfectly plastic [Gerstle *et al.* 2013].

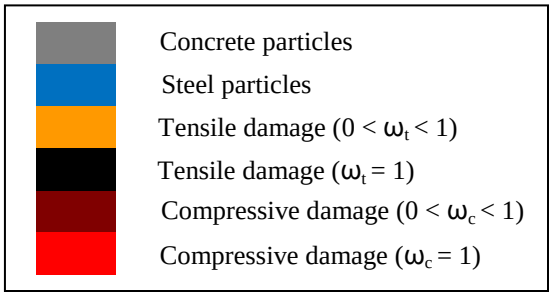


Figure 4.5 – Color-coding for example problems (from Gerstle *et al.* 2013]).

Parameter	**Value**	**Units**
Concrete Young's modulus, E_c	24.86	GPa
Concrete Poisson's ratio, ν_c	0.20	–
Concrete comp. strength, f'_c	27.58	MPa
Concrete tens. strength, f_t	2.758	MPa
Concrete Density, ρ_c	2323.0	kg/m^3
Concrete fracture toughness, G_F	175.0	N/m
Steel Young's modulus, E_s	200.0	GPa
Steel yield strength, F_y	414.0	MPa
Steel Poisson's ratio, ν_s	0.3	MPa
Steel Density, ρ_s	7850.0	kg/m^3

Table 4.1 – Classical material parameters.

Parameter	Value	Units
Lattice Spacing, s	0.020	m
Microelastic parameter, a	4.557×10^7	N
Microelastic parameter, b	506.3	$N\text{-}m^2$
Tensile stretch limit S_t	0.000126	–
Tensile stretch ratio α_t	10	–
S_c	-0.001	–
α_t	5.728	–
β	0.10	–
Damping ratio, c	0.05	–

Table 4.2 – MPLM parameters for concrete.

Parameter	Value	Units
Lattice Spacing, s	0.020	m
Microelastic parameter, a	E_sA_s	N
Microelastic parameter, b	E_sI_s	$N\text{-}m^2$
Yield limit, S_t	0.00207	–
Damping ratio, c	0.05	–

Table 4.3 – MPLM parameters for steel.

4.4.1. 2D uniaxial tension

Fig. 4.6 shows a rectangular plane-stress plate (24×14×12 cm) for two uniaxial tension examples, with the load direction in Fig. 4.6(b) rotated by 90^o with respect to the lattice in Fig. 4.6(a). Equal tensile loads are applied to each of the particles in the end two layers of particles; similarly, opposite forces are applied to the two layers of particles on the opposite end of the specimen. The total number of simulation time steps is 10000. The damage patterns in each specimen at the end of the simulation are shown in Fig. 4.6. For the lattice orientation in Fig. 4.6(a), the failure load was at a stress level of 2.784 MPa (within 1% of the target f_t). However, for the lattice orientation in Fig. 4.6(b), the failure load was at a stress level of 3.226 MPa (17% higher than the target f_t). Thus, although character of the damage patterns are reasonable in both cases, we conclude that the tensile strength is somewhat sensitive to lattice orientation. Significant tensile damage (yellow) is evident prior to crack formation (black) [Gerstle *et al.* 2013].

(a)

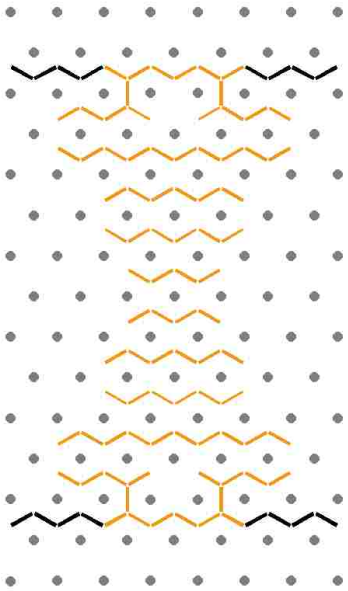


Figure 4.6 – Damage patterns for uniaxial tension. Deformations are magnified by factor of 100, at time step 10000. (a) Load applied in vertical direction.

(b)

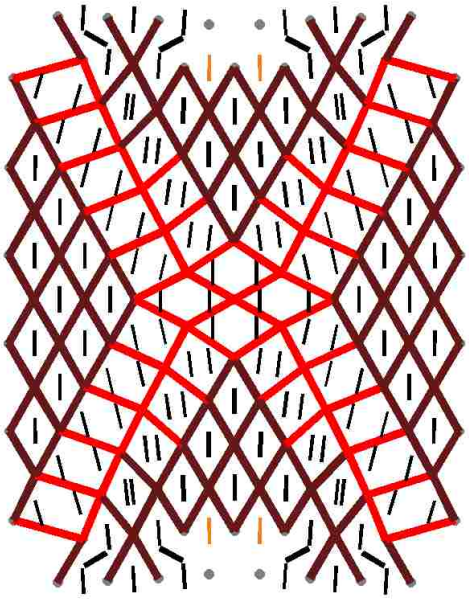
Figure 4.6 – Damage patterns for uniaxial tension. Deformations are magnified by factor of 100, at time step 10000, continued. (b) Load applied in horizontal direction (from [Gerstle *et al.* 2013]).

4.4.2. 2D uniaxial compression

To investigate uniaxial compressive behavior, the problem described in Section 4.4.1 is repeated, but now the loading directions are reversed. The resulting deformed configurations and damage patterns are shown in Figs. 4.7(a) and 4.7(b) for two loading directions with respect to the lattice orientation.

For the lattice orientation in Fig. 4.7(a), the failure load is at a stress level of 28.48 MPa (3.3% higher than the target f'_c). However, for the lattice orientation in Fig. 4.7(b), the failure load is at a stress level of 61.06 MPa (121% higher than the target f'_c). As shown in Figs. 4.7(a) and 4.7(b), the tensile damage patterns between the two loading orientations are similar, but the compressive damage patterns are different. We conclude that both the compressive strength and failure mode are sensitive to lattice orientation. Further study and investigation are needed [Gerstle *et al.* 2013].

(a)



(b)

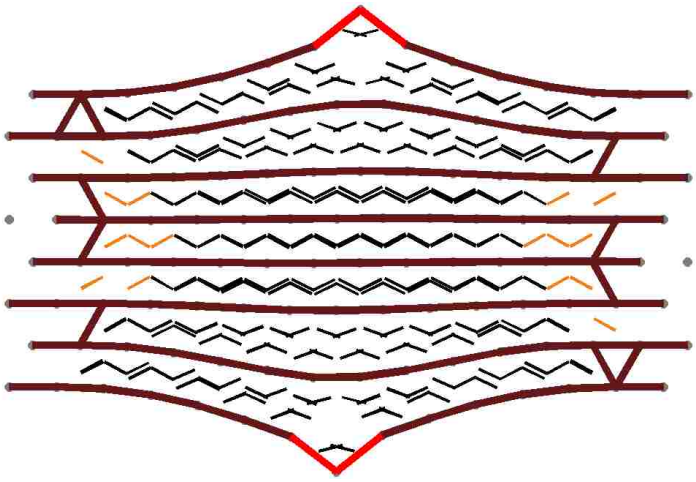


Figure 4.7 – Damage patterns for uniaxial compression. Deformations are magnified by factor of 10. (a) Load applied in vertical direction. (b) Load applied in horizontal direction (from [Gerstle *et al.* 2013]).

4.4.3. 2D plain concrete beam; simply-supported

We assume a plane-stress uniformly-loaded beam, with span of 1.16 m, depth of 0.26 m, and thickness of 0.12 m. With reference to Fig. 4.8, each particle in the top layer of particles is loaded downward to simulate uniform loading. To apply the load approximately statically, the load is ramped linearly from time zero up to 75% of total simulation time and then the load is held steady. The simulation is run for 40000 time steps. Assuming, classically, that the beam's strength is achieved when the bending stress reaches tensile strength, f_t, the failure load is predicted as 22.14 kN/m. The MPLM simulation predicts a failure load approximately 2.6 times higher than the classical failure load: 57 kN/m. This result is expected: the modulus of rupture is typically two or three times the tensile strength, especially for small beams. The deformed shape and the evolving damage in the beam at time step 40000 are shown in Fig. 4.8. Note that the crack branches which is probably a consequence of dynamic fracture [Gerstle *et al.* 2013].

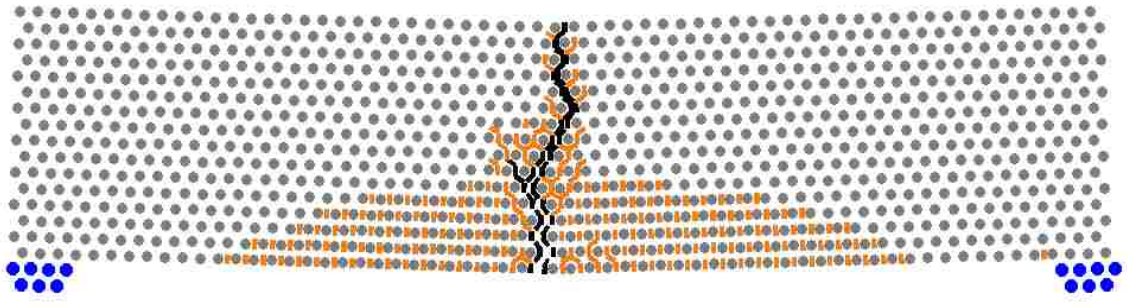
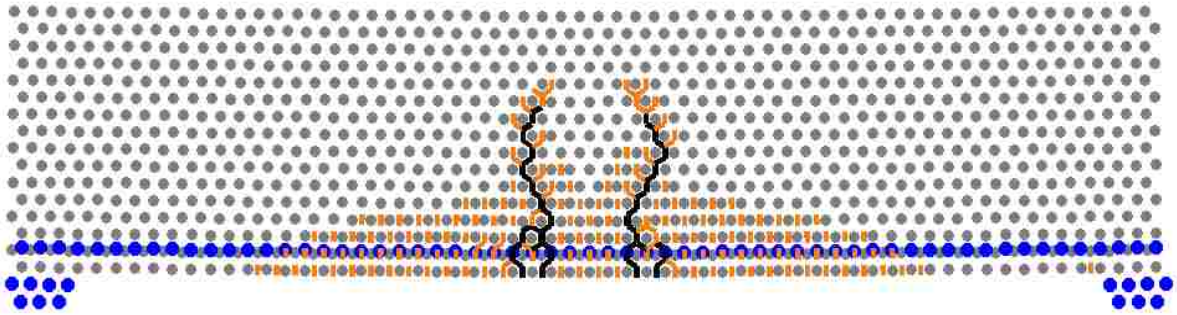


Figure 4.8 – Deformed shape and damage in plain concrete beam subject to uniform loading at time step 40000. Deformation is magnified by factor of 10 (from [Gerstle *et al.* 2013]).

4.4.4. 2D reinforced concrete beam with no stirrups; simply-supported

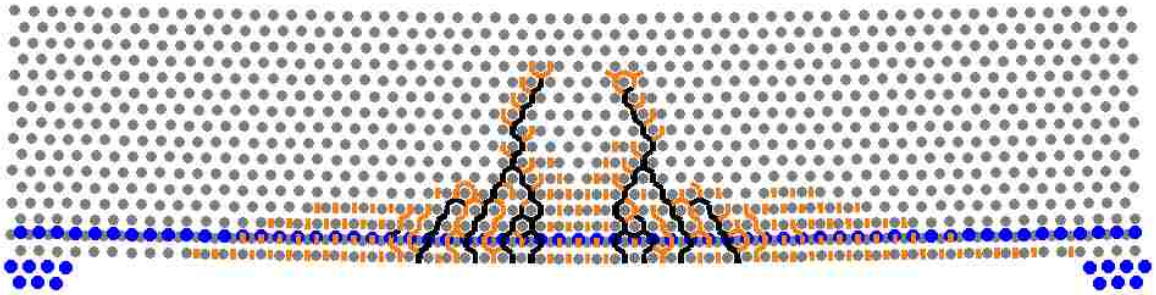
The plain beam in the previous section is now reinforced with a single horizontal steel reinforcing bar, of diameter 1.27 cm, whose centroid is located 2 cm above the bottom of the beam, as shown in Fig. 4.9. The beam is loaded as in the previous section. The simulation is run for 80000 time steps. According to the ACI code [ACI318 2011], the bending strength of the singly-reinforced beam is 72.24 kN/m. The MPLM simulation predicts a slightly higher strength of 78.74 N/m. The deformed shape and associated damage are shown in Fig. 4.9.

The damage patterns look reasonably realistic, including secondary cracking at the bottom of the beam and some compression damage at the top of the beam. At time step 65000, the distributed damage above the reinforcing bar extending to the ends of the beam and the compression damage above the supports is somewhat unexpected and requires further study.

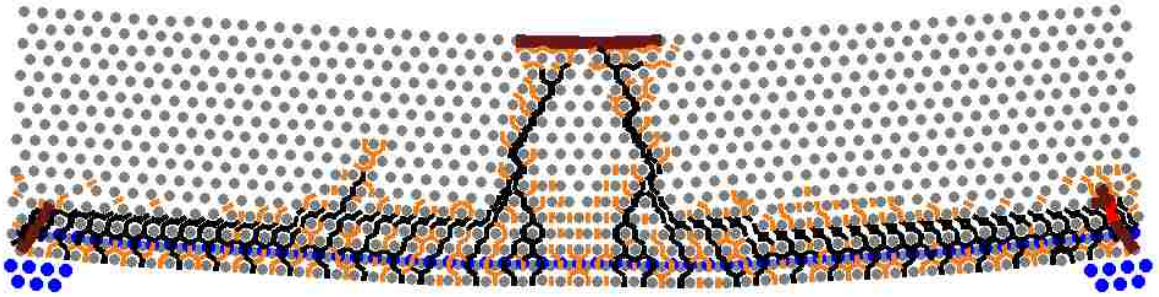


Time step 45000

Figure 4.9 – Deformed shape and damage in reinforced concrete beam subject to uniform loading. Deformations are magnified by factor of 10.



Time step 50000



Time step 65000

Figure 4.9 – Deformed shape and damage in reinforced concrete beam subject to uniform loading, continued (from [Gerstle *et al.* 2013]).

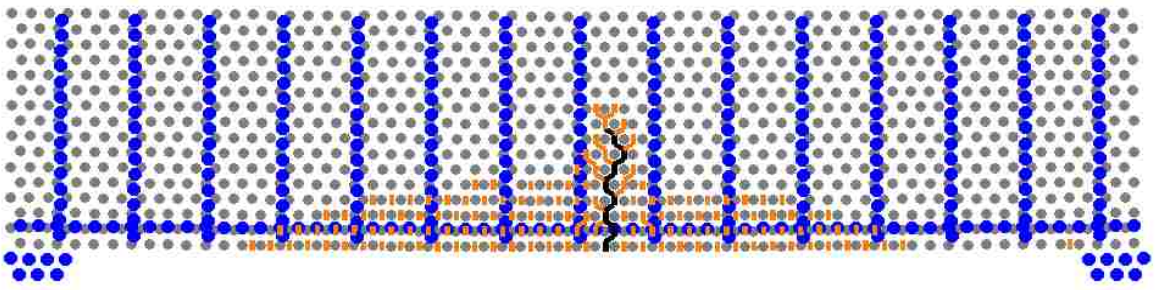
4.4.5. 2D reinforced concrete beam with stirrups-bending failure; simply-supported

The plain beam in Section 4.4.3 is now reinforced with both flexural and the shear rebars having diameters of 1.27 cm and 1.0 cm, respectively. The centroid of flexural steel is located 2.0 cm above the bottom of the beam. The centroid of left-most stirrup (shear reinforcement) is positioned at 4.0 cm from the left end of the beam and the stirrup

spacing is 15 cm. The ACI code [ACI318 2011] predicts a bending-type failure of 72.24 kN/m. The MPLM simulation is run for 80000 time steps. The MPLM simulation predicts strength of 68.63 kN/m. The deformed shape and damage in the beam are shown in Fig. 4.10 at three different stages of loading. It is quite similar to those of the ACI code [ACI318 2011] but is achieved in a more rational (less empirical) way than the ACI equations.

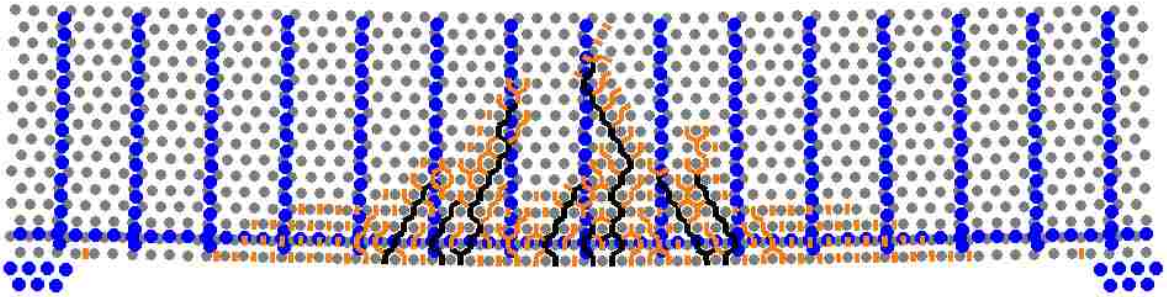
The advantage of the MPLM over continuum models is that it allows fracture to develop unhindered by assumptions of continuous deformation behavior.

Interestingly, by adding stirrups, the cracking behavior was altered significantly from the unreinforced beam, and the failure load was slightly reduced. More study is indicated [Gerstle *et al.* 2013].

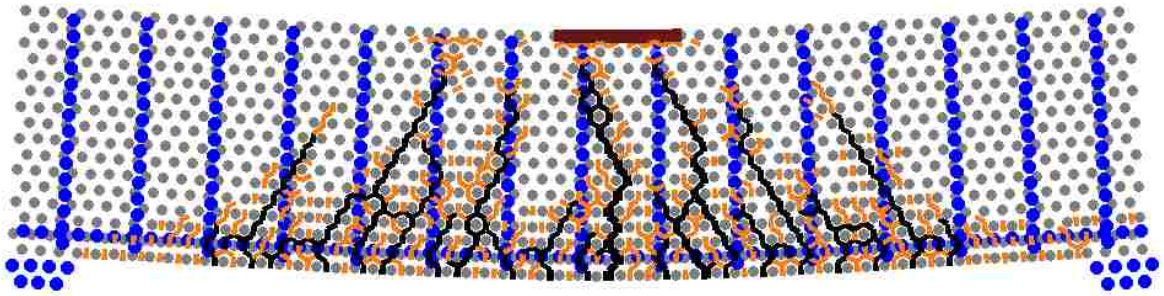


Time step 50000

Figure 4.10 – Deformed shape and damage in reinforced concrete beam with stirrups. Deformations are magnified by factor of 10. Bending failure is indicated .



Time step 60000

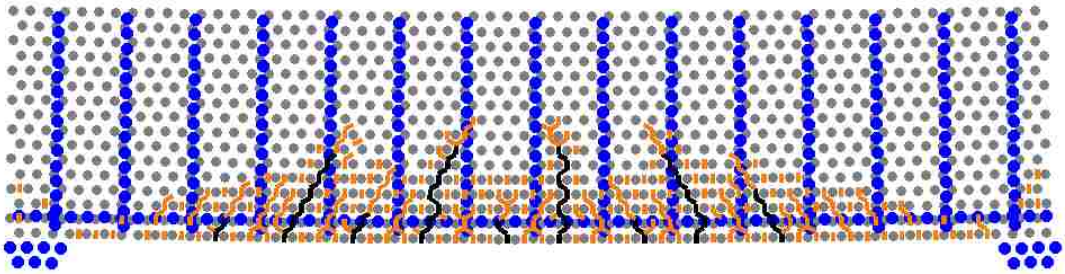


Time step 80000

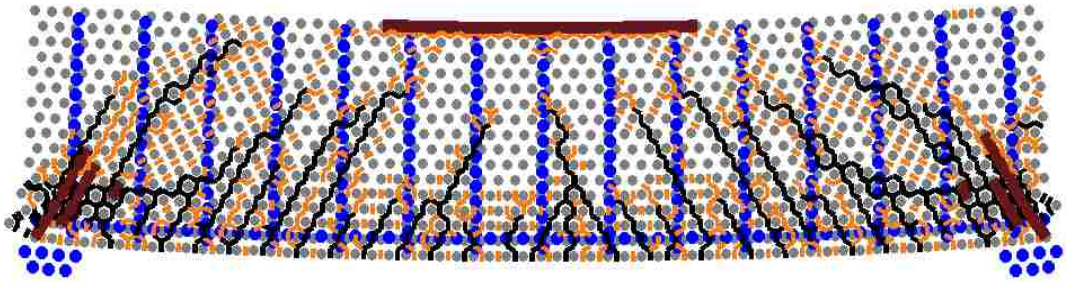
Figure 4.10 – Deformed shape and damage in reinforced concrete beam with stirrups, continued (from [Gerstle *et al.* 2013]).

4.4.6. 2D reinforced concrete beam with stirrups-shear failure; simply-supported

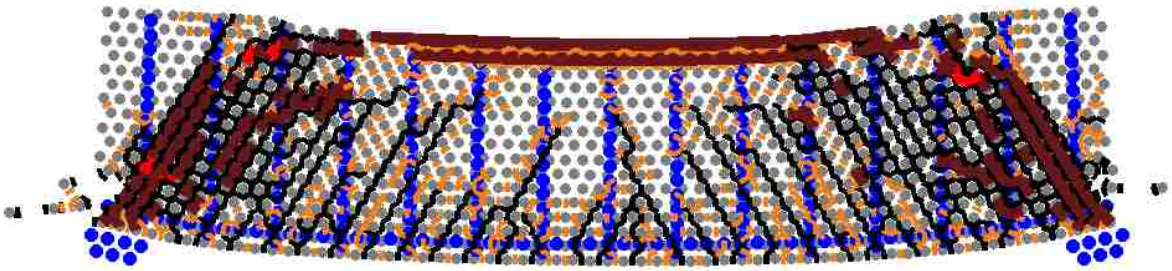
To increase the bending strength and thus induce a shear failure, the diameter of horizontal rebar in the previous section is now increased to 3.175 cm while keeping the stirrups unchanged. According to ACI code [ACI318 2011], the failure is now of the shear type, with a strength 210 kN/m. The deformed shape and damage in the beam are shown in Fig. 4.11. The MPLM predicts a failure load of 252 kN/m [Gerstle *et al.* 2013].



Time step 30000



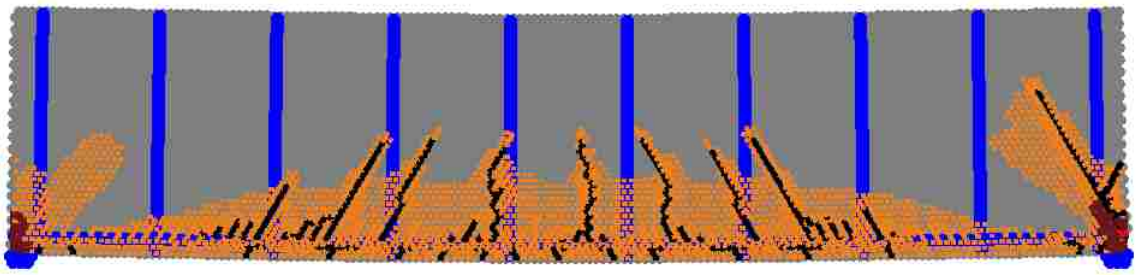
Time step 45000



Time step 80000

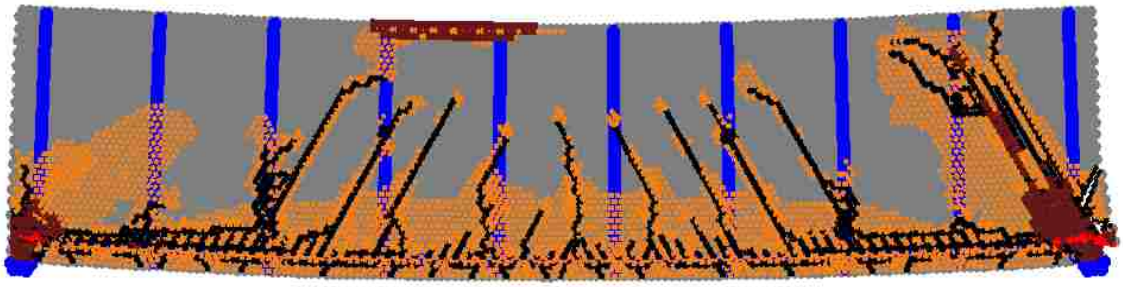
Figure 4.11 – Deformed shape and damage in reinforced concrete beam with stirrups. Deformations are magnified by factor of 10. Shear failure is indicated (from [Gerstle *et al.* 2013]).

In this section, another example of the reinforced concrete beam with stirrups was run using pdQ2 on a parallel supercomputer. 32 processors (8 in the X direction, 4 in the Y direction) were used to run this example and each run took about three minutes. This beam is a plane-stress uniformly-loaded beam, with the length of 4.06 m, depth of 1.02 m and thickness of 0.41 m. It is reinforced with flexural and shear rebar. The flexural and the shear rebar have diameters of 10.16 cm and 2.54 cm respectively. The centroid of flexural reinforcement is located 5.1 cm above the bottom of the beam. The centroid of shear rebar is positioned 8.9 cm from each side of the beam and the shear rebar spacing is 43.2 cm. With reference to Fig. 4.12, each particle in the top layer of particles is loaded downward to simulate uniform loading and the simulation is run for 45000 time steps. The ACI code [ACI318 2011] predicts a shear-type failure of 599 kN/m. The deformed shape and damages in the beam are shown in Fig. 4.12. Also, the MPLM predicts a failure load of 497 kN/m.

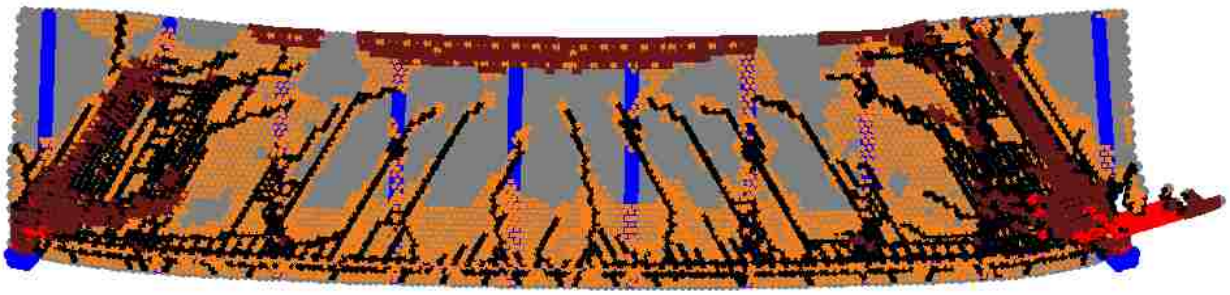


Time step = 25000

Figure 4.12 – Deformed shape and damages in reinforced concrete beam with stirrups. Deformations are magnified by factor of 10. Shear failure is indicated.



Time step = 35000



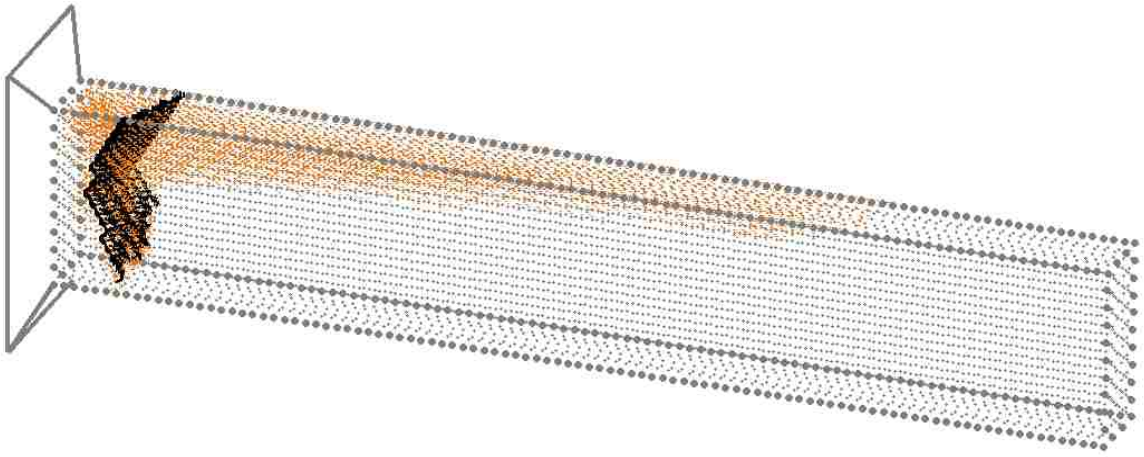
Time step = 45000

Figure 4.12 – Deformed shape and damages in reinforced concrete beam with stirrups, continued.

4.4.7. 3D plain concrete beam; cantilever

With reference to Fig. 4.13, a plain concrete beam is clamped at the left end and is point-loaded at the right tip. Two layers of particles are fixed at the left end and one layer of particles is loaded downwardly at the right end. The beam has a length of 2.0 m, a depth of 0.35 m, and a width of 0.16 m. This beam is modeled in 3D and simulated using 16 processors (8 in the X direction, 2 in the Y direction and 1 in the Z direction). The simulation was run for 100000 time steps and took about 30 minutes. The predicted failure load is 14784 N. Using the ACI code [ACI318 2011]; the failure load is 4505 N.

The deformed shape and damages in the beam are shown in Fig. 4.13.

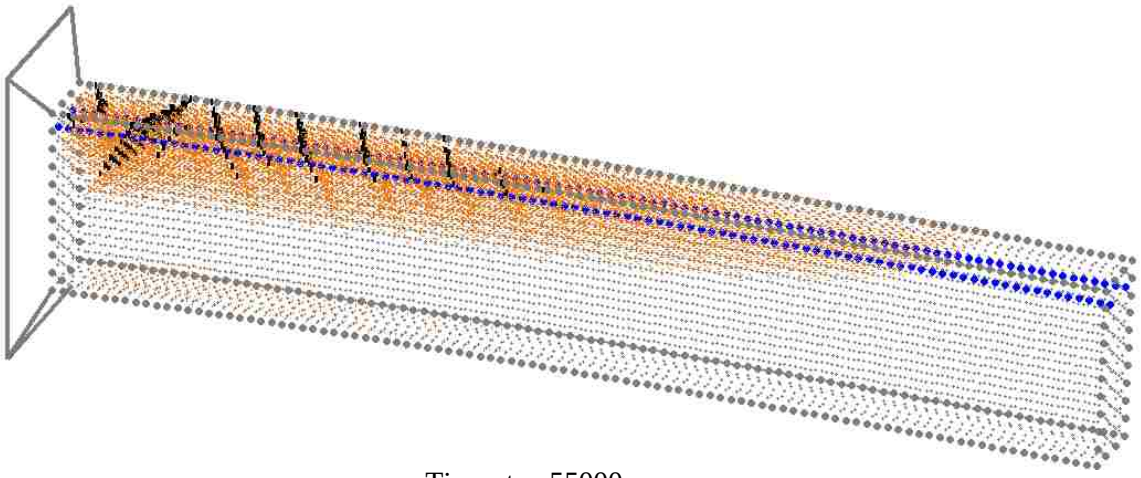


Time step = 100000

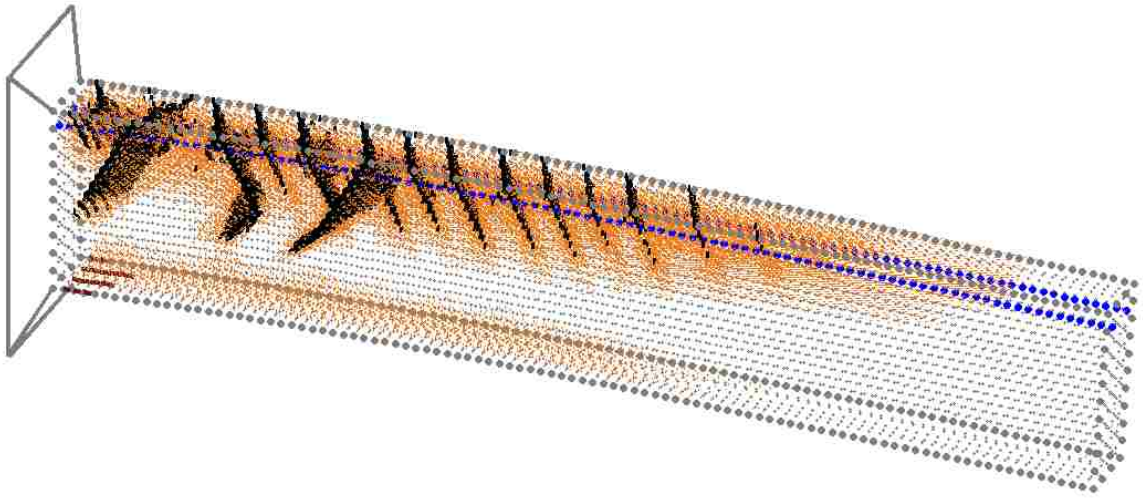
Figure 4.13 – Deformed shape and damages in cantilever plain concrete beam. Deformations are magnified by a factor of 10.

4.4.8. 3D reinforced concrete beam; cantilever

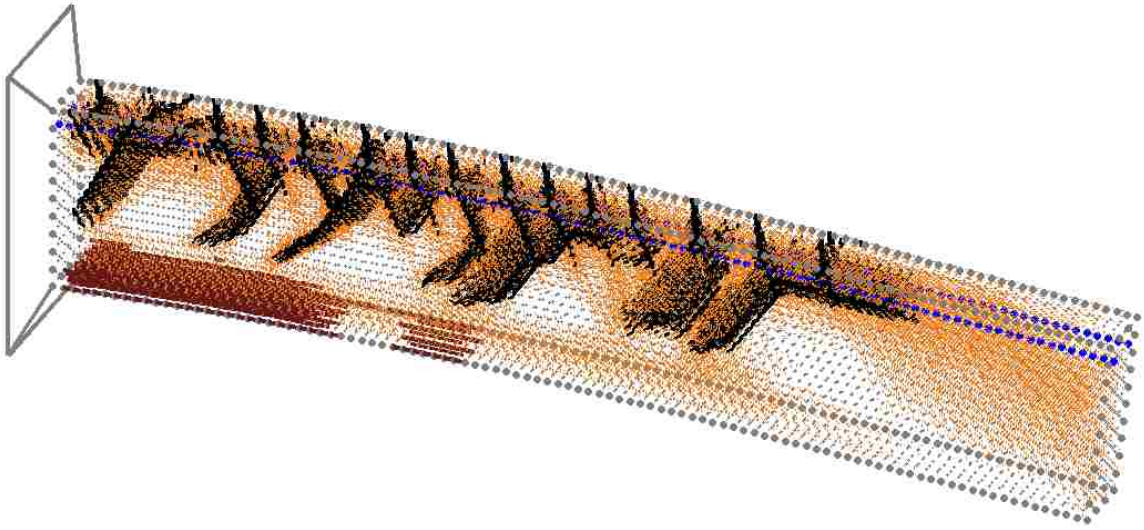
With reference to Fig. 4.14, the previous plain concrete beam is reinforced with 2 #19 rebars with the cover of 4.0 cm. Two layers of particles are fixed at the left end and one layer of particles is loaded downwardly at the right end. The simulation was run for 100000 time steps and took about 45 minutes. The predicted failure load using MPLM is 58400 N. Using the ACI code [ACI318 2011]; the failure load is 37000 N. The deformed shape and damages in the beam are shown in Fig. 4.14.



Time step 55000



Time step 70000

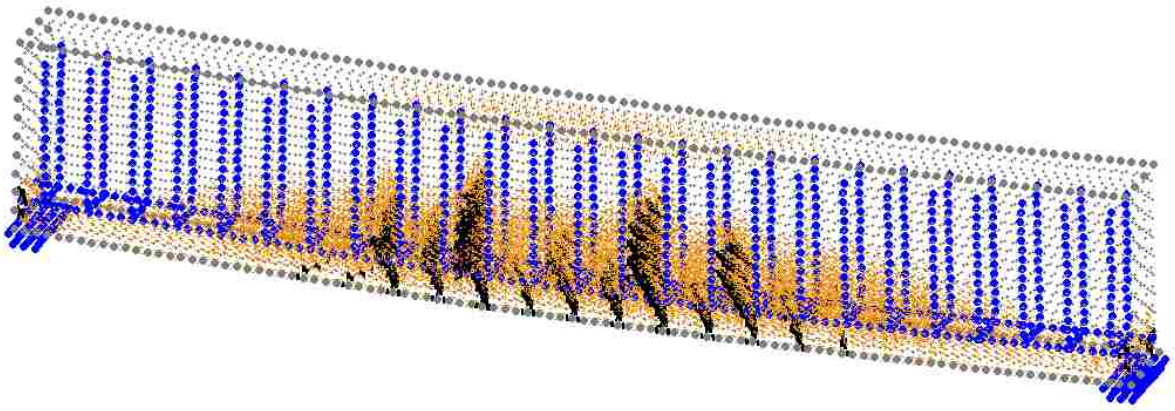


Time step 85000

Figure 4.14 – Deformed shape and damages in cantilever reinforced concrete beam. Deformations are magnified by a factor of 10.

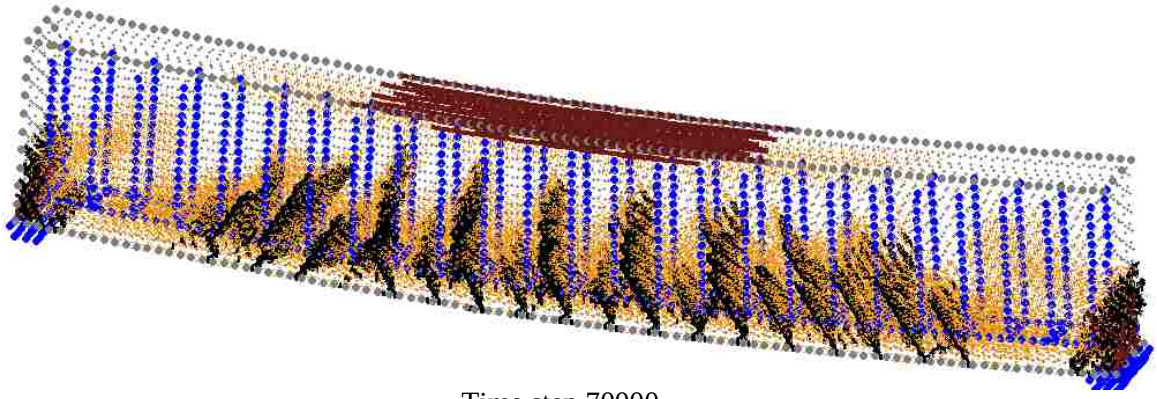
4.4.9. 3D reinforced concrete beam with stirrups-bending failure; simply-supported

The previous example is modeled, but this time, the beam has simple supports and uniform loading on the top of the beam. For shear reinforcement, 2 #13 stirrups are used along the beam with the spacing of 8 cm. The simulation was run for 100000 time steps and took about 50 minutes. The predicted failure load using MPLM is 242 KN/m. Using the ACI code [ACI318 2011]; the failure load is 148 KN/m. The deformed shape and damages in the beam are shown in Fig. 4.15. At the end of the simulation, many damages are seen on the supports. It can be due to the anchorage of longitudinal rebars.

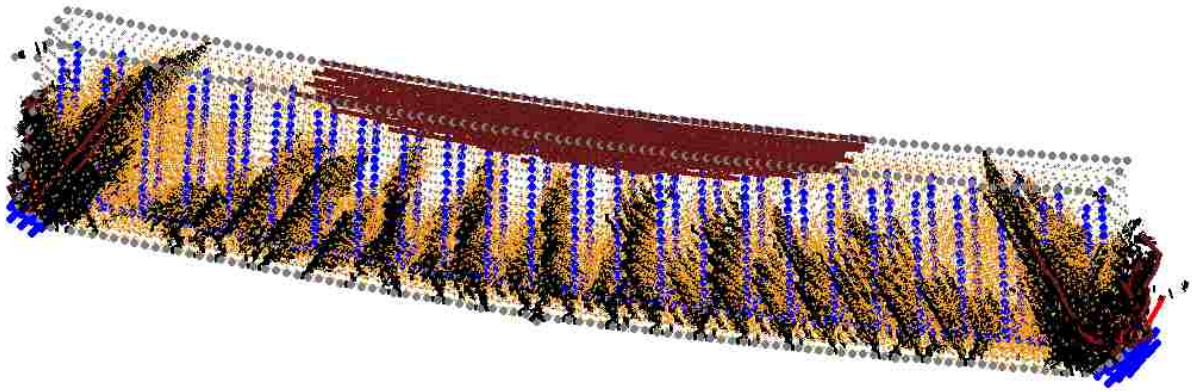


Time step 45000

Figure 4.15 – Deformed shape and damages in reinforced concrete beam with stirrups. Deformations are magnified by factor of 10. Bending failure is indicated.



Time step 70000



Time step 100000

Figure 4.15 – Deformed shape and damages in reinforced concrete beam with stirrups, continued.

4.5. Summary

In this chapter, a microploar peridynamic lattice model (MPLM) for quasi-brittle structures was described, and the MPLM constitutive model for concrete was presented. Through several examples using pdQ2, the ability of MPLM to model major features of reinforced concrete such as stiffness, strength, and failure mechanisms were illustrated. It is important to note that the MPLM is not entirely objective with respect to lattice rotation. This non-objectivity can be addressed in the future by increasing the material horizon, albeit at higher computational cost.

In the next chapter, conclusions and suggestions for future research are presented.

5. CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

The parallel particle dynamics code pdQ2 that has been developed and explained in Chapter 3 has been shown to have the capability to perform large particle dynamics simulations. pdQ2 can be used for different particle dynamics method such as molecular dynamics, peridynamics, the discrete element method. It is a simple, efficient, extensible, and user-friendly code.

pdQ2 benefits from the concepts of procCubes, cells, cores, skins, cell blocks, and walls so that it can be efficiently run on parallel computers. By introducing these concepts, problem partitioning and message passing have resulted in a higher efficiency and flexibility than in the previous version of the code, pdQ. For instance, for peridynamic users, it is now possible to simply implement the state-based peridynamic model which was not previously possible in pdQ due to the complex data structures used. pdQ2 also enabled the straightforward implementation of particle shuffling which was not possible in pdQ. Thus, pdQ2 can be used for simulations in which particle neighborhoods can change. Also, pdQ2 provides user files with clear interfaces for defining domain-specific modules, hiding the complexities of domain decomposition, message passing, and shuffling from the user.

The performance analysis of pdQ2 shows that it is about four times faster than pdQ for the studied benchmark problem. It was found that the reason for this significant difference is the increased efficiency of the message passing in pdQ2. The size and number of the messages passed in pdQ2 is significantly decreased compared to pdQ.

However, pdQ2 can still be improved. More simplifications are possible in the non-user files. Periodic boundary conditions should be implemented, essential for molecular dynamics simulations. Because of the simplifications accomplished, this should not be difficult.

The micropolar peridynamic lattice model (MPLM) introduced in Chapter 4 has been demonstrated to accurately model the behavior of the quasi-brittle structures such as the reinforced concrete. It allows the cracks to grow spontaneously, unlike with classical continuum mechanics. Although the smeared crack model can do similar modeling, it is more complex than the MPLM, and therefore is not presented.

The MPLM is conceptually simple, and therefore can be used with confidence by practicing engineers to produce more rational designs. It has been shown that MPLM with eight parameters is able to capture the tensile and compressive mechanisms of a quasi-brittle material like concrete quite reasonably.

As shown in the examples, the hexagonal lattice model that is presented is not entirely objective with respect to lattice rotations. However, it can model, with reasonable accuracy, the major features of concrete such as stiffness, strength, and failure mechanisms. The non-objectivity with respect to lattice rotations can be reduced by increasing the material horizon to include more particles but at higher computational cost. Furthermore, the geometric description provided by the MPLM is more fundamental than conventional modeling approaches, avoiding implicit assumptions about spatial topology and allowing distributed damage, discrete cracks, and fragmentation to develop as emergent properties [Gerstle *et al.* 2012].

Therefore, using pdQ2 and the MPLM in the digital age seems to provide a reasonable basis for modeling of reinforced concrete structures.

However, further research is required in the future to improve the MPLM model to make it more applicable for engineering applications. The hexagonal lattice is not the only choice and perhaps can be changed to reduce the non-objectivity with respect to lattice rotation. Also, material behaviors such as plasticity and rate dependency like creep and relaxation should be implemented to model concrete more realistically.

APPENDIX – USER MANUAL FOR pdQ2

A.1. Introduction

Users of pdQ2 develop their simulation models in FORTRAN 90. The four user files (*userForce.F*, *userIntegrate.F*, *userSetup.F*, and *userModules.F*) are programmed according to the user's wishes; however, the user need not alter the rest of the pdQ2 files, which handle the complicated aspects of domain decomposition and parallelization.

Unlike with many commercial and over academic simulation programs, the approach of allowing the user access to the source code allows maximal modeling flexibility. In addition, the modeler is thus able to completely review the simulation models of others. This approach also eases the design burden on the pdQ2 developer, as it is unnecessary to foresee all possible future model types that a user might wish to implement.

Users do not need to be concerned with the parallelization that takes place when running on multiple processors. Instead, the user need only program the physics and the response output behavior. The development of the model can take place on a single-processor machine. When the model is completely developed, it can then be readily run on a multi-processor computer.

A.2. Input files

The required input files are free-format ASCII text files, with the following names:

pdQInput.dat

pdQUserInput.dat

ParticleFixedAttrs.dat

ParticleAlterAttrs.dat

pdQ.pbs (batch submission file, required only for MPI simulations)

Each of these files is described next.

pdQInput.dat has general inputs required by any simulation such as decomposition, force field, and integration parameters. Decomposition parameters are the number if processors in X, Y, and Z directions. The force field parameter is the minimum cell size which should be slightly larger than the material horizon. The Integration parameter is the number of time steps.

In addition, the user can specify other input files, LinkFixedAttrs and LinkAlterAttrs, for example.

In *pdQUserInput.dat,* users can define their user-specific parameters.

ParticleFixedAttrs.dat contains, in each record, the particle global ID (a unique integer), the reference x, y, and z particle positions, and any other particle attributes (such as boundary conditions and material properties) that do not change during the course of the simulation.

ParticleAlterAttrs.dat contains, in each record, the particle global ID (a unique integer), the current x, y, and z particle positions, and any other particle attributes that may change during the course of the simulation (such as rotations, velocities, damage parameters).

In *pdQ.pbs*, the batch submission script for running pdQ2 in a shared queue on a parallel machine, users specify the number of computational nodes, processors, and required time for MPI based simulation.

A.3. Processing

The user can customize the four user files: *userForce.F*, *userIntegrate.F*, *userModules.F*, and *userSetup.F*. Note that any file, subroutine or variable beginning with the word "user" refers to a user specification.

***userForce.F*:**

In *userForce.F*, users define the force interactions between particles. The force calculations depend upon access to particle alterable and fixed attributes that are stored in the *ptclAlterAttrs* and *ptclFixedAttrs* arrays (five-dimensional arrays) that are defined in *modules.F* file in the module *procCubeArrays*.

For example, the i^{th} alterable attribute of particle "j" in cell "k" on the home processor is addressed as *ptclAlterAttrs(i, jPtclCell, kCellX, kCellY, kCellZ)*.

***userIntegrate.F*:**

In *userIntegrate.F*, the user specifies the time integration algorithm using the *ptclIntegAttrs* array to update *ptclAlterAttrs*.

***userModule.F*:**

In *userModules.F*, the user may declare user-specified variables for modeling purposes.

For example, user scalars can be defined them in the module *userScalars* in *userModules.F* file, and included in a user subroutine with the FORTRAN statement "use *userScalars*".

***userSetup.F*:**

In *userSetup.F*, the user may want to set up extra data arrays. For example, user-defined links and even finite elements can be constructed in *userSetup.F*.

Also, users can output any file they wish, but files that may be output for all simulations are *ParticleAlterAttrs.timestep.rst*, *timing.out*, and *debug.txt.*

After writing the user files and input files, the user compiles the program to create the executable file. Then, the user runs the executable file.

A.4. Output files

The standard output files are free-format ASCII text files, with the following names:

ParticleAlterAttrs.nnnn.rst, where nnnn is the timestep.

timing.out

debug.txt

All of the .rst files have the same formats as the corresponding .dat files. The integer *nnnn* refers to the simulation time step.

In *timing.out* file, the amount of CPU time it takes to run the whole program is reported, along with breakdowns for each component subroutine.

In *debug.txt,* user can write the information that is required to debug the code.

A.5. Steps for running a simulation on multi-processor machine

Step 1. Go to the center website which provides you supercomputing capability and request an account (for example, http://www.carc.unm.edu/ for University of New Mexico).

Step 2. If your local computer has any operating system other than Linux, download a secure shell (ssh) to be able to connect to machines running the Linux OS.

Step 3. Use your account and host information for connecting to clusters through ssh (for example, the host name for the nano supercomputer at UNM is "nano.alliance.unm.edu").

Step 4. Copy the pdQ2 folder from the repository to your computer. The pdQ2 folder contains source files (explained in Section 3.3.4) and a run directory. Note that if you have changed any user files, you need to update them in your directory via ssh.

Step 5. Compile the processor files (FORTRAN files) using the "makeMPI" command.

Step 6. Go to run the directory and update the input files using "SSH file transfer".

Step 7. Edit the "pdQ2.pbs" script. You should specify number of nodes, processors per node, and wall time (amount of time you need to run your simulation to completion).

Step 8. Run the simulation using the "runMPI" command.

Step 9. You can do whatever you want with the output files that are written in the run directory. MATLAB is a good programming environment for postprocessing the results.

REFERENCES

ACI318, 2011. “Building Code Requirements for Structural Concrete”. American Concrete Institute.

Allen, M.P., Tildesley, D.J., 1987. “Computer Simulation of Liquids”. Oxford Science Publications, Oxford.

Atlas, S.R., Cummings, J.C., Reynders, J.V.W., 1996. “Parallel Molecular Dynamics Simulation in the POOMA FrameWork”. paper presented at the 1996 Conference on Parallel Object-Oriented Methods and Applications, Santa Fe, NM.

Atlas, S.R., 1999. “PDQ, a Portable, Parallel and Extensible Molecular Dynamics Simulation Program”. Technical Report, University of New Mexico.

Atlas, S.R., Valone, S.M., 2012. “Density Functional Theory of the Embedded-Atom Method: Multiscale Dynamical Potentials with Charge Transfer”. preprint, Physical Review B, to be submitted.

Atlas, S.R., Wright, A.F., Ramprasad, R., Cano, L., 1998-2012. “Vernet: A Parallel Planewave Pseudopotential Code for Density Functional Electronic Structure Calculations in Solids”. University of New Mexico.

Beazley, D.M., Lomdahl, P.S., 1994. “Message-Passing Multi-Cell Molecular Dynamics on the Connection Machine 5”. Parallel Computing, 20 (2): 173-195.

Bhatele, A., Kumar, S., Mei, C., Phillips, J.C., Zheng, G., Kalé, L.V., 2008. “Overcoming scaling challenges in biomolecular simulations across multiple platforms”.

22nd IEEE International Symposium on Parallel and Distributed Processing, Pages: 1-12 , DOI: 10.1109/IPDPS.2008.4536317.

Bowers, K.J., Chow, E., Xu, H., Dror, R.O., Eastwood, M.P., Gregersen, B.A., Brown, D., Maigret, B., 1999. "Large Scale Molecular Dynamics Simulations using the Domain Decomposition Approach". Proceedings of the 24th SPEEDUP Workshop, Berne, Sept. 24-25, 12(2): 33-41.

Bowers, K., Dror, R., Shaw, D.E., 2006. "The Midpoint Method for Parallelization of Particle Simulations". Journal of Chemical Physics, 124: 184109.

Bruck, J., Ho, C., Kipnis, S., Weathersby, D., 1994. "Efficient Algorithms for All-To-All Communications in Multi-Port Message-Passing Systems". Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures, Cape May, NJ, ACM Press, 6: 298-309.

Crisfield, M.A., 1991. Nonlinear Finite Element Analysis of Solids and Structures. Vol. 1: Essentials, Wiley, NY.

Cusatis, G., Bazant, Z.P., Cedolin, L., 2006. "Confinement-Shear Lattice CSL Model for Fracture Propagation in Concrete". Computer Methods for Applied Mechanics and Engineering, 195: 7154–7171.

Foster, I., 1995. Designing and Building Parallel Programs, Addison – Wesley, Reading, MA.

Fincham, D., Ralston, B.J., 1981. “Molecular Dynamics Simulation Using the Cray-1 Vector Processing Computer”. Computer Physics Communications, 23(2): 127-134.

Germann, T.C., Kadau, K., Swaminarayan, S., 2009. “369 Tflop/s molecular dynamics simulations on the petaflop hybrid supercomputer ‘Roadrunner’”. Concurrency and Computation: Practice and Experience 21(17): 2143–2159.

Gerstle, W.H., Honarvar Gheitanbaf, H., Asadollahi, A., 2013. “Computational Simulation of Reinforced Concrete using the Micropolar State-Based Peridynamic Hexagonal Lattice Model”. Proceedings of FRAMCOS 8, Toledo, Spain, accepted.

Gerstle, W., Honarvar Gheitanbaf, H., Asadollahi, A., Tuniki, B.-K., and Rahman, A., 2012. “Simulation of concrete using micropolar peridynamic lattice model”, Computers and Structures. under review.

Gerstle, W., Sakhavand, N., Chapman, S., 2010. “Peridynamic and continuum models of reinforced concrete lap splice compared”. Proceedings of the 7th International Conference on Fracture Mechanics of Concrete and Concrete Structures (FraMCoS-7), pp. 306-312.

Gerstle, W., Sau, N., Aguilera, E., 2007a. “Micropolar Peridynamic Constitutive Model for Concrete”. 19th Intl. Conf. on Structural Mechanics in Reactor Technology (SMiRT 19), Toronto, Canada, August 12-17, pp. B02/1-2.

Gerstle, W., Sau, N., Silling, S., 2007b. “Peridynamic Modeling of Concrete Structures”. Nuclear Engineering and Design, 237(12-13): 1250-1258.

Gerstle, W., Silling, S., Read, D., Tewary, V., Lehoucq, R. 2008. “Peridynamic Simulation of Electromigration”. Computers, Materials and Continua, Tech Science Press, 8(2): 75-92.

Hendrickson, B., Plimpton, S., 1992. “Parallel Many-Body Simulations Without All-To-All Communication”. Technical Report SAND92-0792, Sandia National Laboratories, Albuquerque, NM.

Hess, B., Kutzner, C., van der Spoel, D., Lindahl, E., 2008. “GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation”. Journal of Chemical Theory and Computation, 4: 435–447.

Jennings, P.C., 1971. “Engineering features of the San Fernando earthquake of February 9”. California Institute of Technology 1971 EERL-71-02.

Kadau, K., Germann, T. C., Lomdahl, P. S., 2006. “Molecular Dynamics Comes of Ages: 320 Bilion Atom Simulation on BlueGene/L”, Modern Physics, 17(12): 1755-1761.

Klepeis, J.L., Kolossváry, I., Moraes, M.A., Sacerdoti, F.D., Salmon, J.K., Shan, Y., Shaw, D.E., 2006. “Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters”. Proceedings of the ACM/IEEE Conference on Supercomputing (SC06), Tampa, Florida, November 11–17, 200.

Lindahl, E., Hess, B., van der Spoel, D., 2001. “GROMACS 3.0: A Package for Molecular Simulation and Trajectory Analysis”. Journal of Molecular Modeling, 7(8): 306-317.

Lindorff-Larsen, K., Piana, S., Palmo, K., Maragakis, P., Klepeis, J.L., Dror, R.O., Shaw, D.E., 2010. “Improved Side-Chain Torsion Potentials for the Amber ff99SB Protein Force Field”. Proteins: Structure, Function, and Bioinformatics 78 (8): 1950–1958.

Moore, G.E., 1965. “Cramming more components onto integrated circuits". Electronics. Volume 38, Number 8.

Nelson, M., Humphrey, W., Gursoy, A., Dalke, A., Kalé, L., Skeel, R.D., Schulten, K., 1996. “NAMD - A parallel, object-oriented molecular dynamics program”. International Journal of Supercomputer Applications and High Performance Computing, 10:251-268.

Nguyen, V.B., Chan A.H.C., Crouch, R.S., 2005. “Comparisons of Smeared Crack Models For RC Bridge Pier Under Cyclic Loading”. 13th ACME conference, University of Sheffield, pp. 111-114.

O'Mara, W.C., Herring, R.B., Hunt, L.P., 1990. Handbook of Semiconductor Silicon Technology. William Andrew Inc.

Pacheco, P.S., 2011. “An Introduction to Parallel Programming”.

Parks, M.L., Lehoucq, R.B., Plimpton, S.J., Silling, S.A., 2008. “Implementing Peridynamics within a Molecular Dynamics Code”. Computer Physics Communications, 179: 777-783.

Plimpton, S.J., 1995. “Fast Parallel Algorithms for Short-Range Molecular Dynamics”. Computational Physics, 117: 1-19.

PGI Compiler User's Guide, 2012. Parallel FORTRAN, C and C++ for Scientists and Engineers.

Rahman, ASM.A., 2012. "Lattice-Based Peridynamic Modeling of Linear Elastic Solids". Master's Thesis, University of New Mexico.

Sakhavand, N., 2011. "Parallel Simulation of Reinforced Concrete Structures Using Peridynamics". Master's Thesis, University of New Mexico.

Silling, S., 1998. "Reformulation of Elasticity Theory for Discontinuities and Long-Range Forces". Technical Report SAND98-2176, Sandia National Laboratories, Albuquerque, NM.

Silling, S.A., Epton, M., Weckner, O., Xu, J., Askari, E., 2007. "Peridynamic States and Constitutive Modeling". Journal of Elasticity, Vol. 88, Issue 2, pp. 151-184.

Tuniki, B.K., 2012, "Peridynamic Constitutive Model of Concrete". Master's Thesis, University of New Mexico.

Yaw, L.L., 2008. "Co-rotational Meshfree Formulation For Large Deformation Inelastic Analysis Of Two-Dimensional Structural Systems". PhD thesis, Dept. of Civil and Environmental Engineering, UC Davis.

Zhou, S.J., Preston, D.L., Lomdahl, P.S., Beazley, D.M., 1998. "Large-Scale Molecular Dynamics Simulations of Dislocation Intersection in Copper". *Science* 279, *1525.152.*