Multi-Particle Motion in Lattice Computers
(Preliminary Draft)

John Case
Computer Science Department
103 Smith Hall
University of Delaware
Newark, DE 19716

Srikrishnan R. Chitoor
820 Seward Ave, #GE
Evanston, IL 60202

Dayanand S. Rajan
DCS
Yardley, PA 19067

Anil M. Shende
Department of Computer Science
Bucknell University
Lewisburg, PA 17837

Abstract

Motion of a set of particles along their respective curves at constant speed can be simulated analogically, approximately in real time, by communication between processors in lattice computers representing bounded regions of euclidean space.An algorithm to simulate motion of a single particle along a curve already existed. In this paper, we extend this algorithm to produce two algorithms that simulate motion of a *set* of particles. Each extension exploits parallel processing, but models non-interacting particles, a bounded number of which can fit into a finite volume of space. This latter, physically natural constraint, ensures efficiency and makes the approximately real time simulations even possible. Allowing for some tolerable gaps and lags in simulations, our second extension runs even more efficiently than the first, at least under some reasonable conditions.

Contents

Chapter 1

Introduction

Scientific computing involves simulation of physical events. Many physical events can be modelled as motion of a set of objects in euclidean space. A typical problem involves simulating a physical event and finding the values of certain set of attributes associated with the simulated objects (e.g., velocity). The set of attributes associated with the simulated objects depend on the problem under consideration. The problem will invariably involve getting the values of this set of attributes of a set of objects at some points in time. For example, to determine whether two objects thrown from two different points on earth collide, we can try to simulate the motion of these objects. An attribute associated with these objects is the location of the object with respect to a 3-dimensional coordinate system. We might collect values of this attribute during the course of a relevant simulation and check to see if the values are the same at any instant.

We are interested in simulating physical events like the one described above for applications in scientific computing and in using parallel computing. We want to *analogically* simulate the motion of objects in euclidean space approximately in real time. We want the analogical simulation of the physical event to resemble that event as closely as possible. If we witness a physical event, we will be able to make claims (even if approximate) about distances between objects involved in the event. One could judge, just by "looking" at the event, the relative distance between two objects with respect to distance between another pair of objects. We note that this is true irrespective of the actual attributes we are interested in for a physical event. We want this to be true in our simulations. One should be able to make similar judgements by "looking" at the simulations. This paper comes out of a research project working on the development of a *literal/analogical* approach to simulating physical events in a computer providing an analog representation of the corresponding euclidean space. These computers are described briefly in Section 2.1.

The algorithms presented in this paper take us a step towards achieving the goal of analogically simulating any physical event. These algorithms simulate the motion of a *set* of particles along their respective curves (at constant speed). These algorithms are extensions of the algorithm for simulating the motion along a curve of a single particle in [She91, CRS91a]. We also prove correctness of our new algorithms.

In Section 1.1, we look at the approach we have taken in the project and its advantages. In Section 1.2, we present an overview of this paper.

1.1 Motivation

[AG89, Qui87] indicates that simulation of physical events has applications to a number of problems including fluid flow, computing terrain map based on reflected radar signals, weather prediction and N-body problems. Previous approaches using parallel computing to handle these simulations used analytical models. Most of these approaches require solving a set of differential equations. A major disadvantage of this approach is that there is no uniform procedure for simulations. Each problem brings about its own set of equations. Worse, there might be physical events for which no reasonable equations exist for a simulation. Secondly, a simulation algorithm might have to be changed to find values of a different set of attributes in the *same* physical event.

We divide space into disjoint regions and, in parallel, simulate the particles in different regions. Other such approaches may also divide space (e.g., [AG89]), but their representation of objects is not "direct." One cannot "look" at the simulations and make claims that one could make witnessing a physical event. In the previous approaches, location of a object gets represented as part of the attributes of the object. Therefore, when the object moves, it is one of the attributes that gets changed and not the *spatial location* of the object in the simulation. In the approaches we follow, where movement of an object in euclidean space translates into movement of the object in a representation of space, the spatial distance between objects is maintained. Space is represented by space. Basically, we move data and algorithms, representing objects, around in a spatial configuration of processors, in much the same way as real objects move around in the configuration of real space.

There are several advantages to our method. The approach is general enough to simulate most kind of physical events that are governed by a simple behavior pattern. Our simulations occur approximately in real time. In the previous approaches, the simulation time is problem dependent and may increase with the number of particles to be simulated in a non-linear way. Also, in these conventional approaches to simulation, there is a wide *semantic gap* [She91] between the simulation and the physical event. The simulation acts on analytical representations of objects which have to be translated to the real world[1] every time one wants to interpret the results. Our approach reduces this semantic

[1]Translated by either the user or the computer.

gap. What one sees in the simulation is very close to what is happening in the physical event.

1.2 Overview of the Paper

The previous two sections talk about the project as a whole. This paper is a small step in the project. We present two algorithms for simulating a set of particles along their respective curves at constant speed. The parameters of these algorithms are properties of a set of particles and curves traced by them. The result is a simulation of the motion of the set of particles in a representation of euclidean space.

In Chapter 2, we will look at some previous work that has been done in this project. Our algorithms use a representation of euclidean space from [CRS90a, She91, CRS91c, RS91] to simulate the motion of a set of particles, and we summarize this representation of euclidean space in Section 2.1 and describe briefly the corresponding computers for simulations. We summarize the single particle algorithm from [She91, CRS91a] in Section 2.2.

In Chapter 3 we present some notation used in the description and analysis of the two new algorithms and take a look at the step by step development of these algorithms. Section 3.1 specifies certain timing conditions, which, when satisfied by these algorithms, guarantee that they simulate motion approximately in real time. In Section 3.2, we present certain computations that can be eliminated, from the single particle motion simulation algorithm in [She91, CRS91a], without affecting its correctness. This savings in computations are extended to our new algorithms.

One might expect that an algorithm which simulates the motion of a set of, say, N particles would take N times more execution time than an algorithm which simulates the motion of a single particle. Using inherent properties of euclidean space and physical particles, the first algorithm, called the FE algorithm,[2] significantly reduces the execution time for simulating N particles, compared to the execution time required by N single particle algorithms. In Section 3.3, we will outline how this reduction was made possible. Allowing for some reasonable gaps and lags in the simulations, we developed another algorithm, called the SE algorithm,[3] which, in certain cases, simulates the motion of a set of particles in less time than the FE algorithm. We will outline the development of the SE algorithm in Section 3.4.

In Chapter 4 we present and analyze the FE algorithm. In Section 4.1, we present definitions that are made use of in the description of the FE algorithm. In Section 4.2, we present the this algorithm in pseudo code. In Section 4.3, we analyze the FE algorithm for correctness and optimality.

[2]It is the **F**irst **E**xtension of the single particle motion simulation algorithm.

[3]It is the **S**econd **E**xtension of the single particle motion simulation algorithm.

In Chapter 5 we present and analyze the SE algorithm. In Section 5.1, we present a concept used by this algorithm to ensure that the gaps and lags in its simulations are reasonable. The basic idea for this algorithm is presented in Section 5.2. In Section 5.3, we look at some definitions and description of the SE algorithm. In Section 5.4, we move on to analyze the SE algorithm for correctness and its performance vis-a-vis the FE algorithm.

In Chapter 6, we summarize the algorithms presented in this paper and present some of the future extensions that need to be considered.

In Chapter 7, we provide two tables that illustrate the execution of the SE algorithm.

Chapter 2

Previous Work

Two important issues to be considered in the literal/analogical approach from [CRS90b, CRS90a, She91, CRS91c, CRS91b, CRS91a, CRS92] for simulation of motion are:

- on which architectures are the simulations is going to be done, and
- how are the simulations going to be done.

[CRS90b, CRS90a, She91, CRS91c] deal with proposed architectures on which to do the simulations. This is briefly overviewed in Section 2.1 below. [She91, CRS91b, CRS91a] deal with algorithms for constant speed motion of, single, non-dissipating wave-fronts and particles. [CRS92] provides extensive motivation for this previous work of relevance to the present paper.

As mentioned in Chapter 1, this paper extends the single particle algorithms from [She91, CRS91a] to (non-interacting) *sets* of particles. In Section 2.2 below, we summarize the top level of the algorithms from [She91, CRS91a] for simulating the motion of a single particle.

2.1 Architecture

As promised just above, we briefly describe the computer architecture component of the approach to simulation of motion on which the present paper is based.

In this regard, in Section 2.1.1, we first discuss a way to nicely, discretely represent *all* of euclidean n-space. Then, also in Section 2.1.1, we show how to define relevant computer architectures (or at least their processor interconnection schemes) naturally based on the discrete representations for *bounded* regions of euclidean n-space. Lastly, we indicate some refinements to the discrete representations and associated parallel processing computers, first in Section 2.1.2, to improve the representation of space and second, in Section 2.1.3, to make sure we have algorithms for approximately linear in real time simulation of basic motions.

2.1.1 Discrete Representation of Space and Processor Interconnection

In the approach on which this paper is based, a *discrete representation of euclidean n-space* is a set of points $\mathcal{D}$ with the following properties.

- There is a positive constant ϵ such that for any point Q in the euclidean space, there is a point R in $\mathcal{D}$ such that the euclidean distance between the points Q and R is $< \epsilon$,
- the points in $\mathcal{D}$ do not cluster arbitrarily close together, and
- $\mathcal{D}$ is invariant under translations.

It turns out that a set of points $\mathcal{D}$ satisfies the above three conditions iff $\mathcal{D}$ is a n-dimensional lattice, where an *n-dimensional lattice* [CN93, GL87] is a set of points, from some copy of euclidean n-space, providing a discrete variant of an n-dimensional vector space [Her64] in which the scalar multipliers for magnifying vectors are restricted to being integers. For example, Z^n, the set of n-tuples of integers, and A_n, the set of $(n+1)$-tuples of integers that add up to 0, are both n-dimensional lattices [CN93]. For A_n, the associated copy of euclidean n-space is the set of $(n+1)$-tuples of *reals* that add to 0. Clearly, for example, the set of 3-tuples of *reals* that add to 0 is just a plane, a copy of 2-dimensional space.

Here is how $\mathcal{D}$ is associated with processor interconnection and associated computers. A *bounded* region of euclidean n-space is selected to be represented by a computer. Let $\mathcal{D}'$ be the set of points from $\mathcal{D}$ in this region. Place identical, synchronized processors centered on each point in $\mathcal{D}'$, and directly connect a pair of these processors (by identical, bi-directional communication channels) iff they are the *lattice minimal distance* apart. Any such computer for representing a corresponding bounded region of euclidean n-space is called a *lattice computer*. Figure 2.1 shows a lattice computer (with opaque processors) based on A_3 and representing a small parallelepiped region of euclidean 3-space.

2.1.2 Improving Representation of Space

In the interest of finding n-dimensional lattices that represent euclidean n-space especially well, more conditions can be imposed on the selection of the set $\mathcal{D}$. For example, it is interesting to be able to represent locally as many directions as possible in a lattice since in euclidean n-space there are a continuum of such directions.

$\mathcal{D}$ is said to be *regular* [CRS90a, CRS91c] iff there is a set of n minimal length vectors (points) which generate $\mathcal{D}$ and have a constant angle between any two

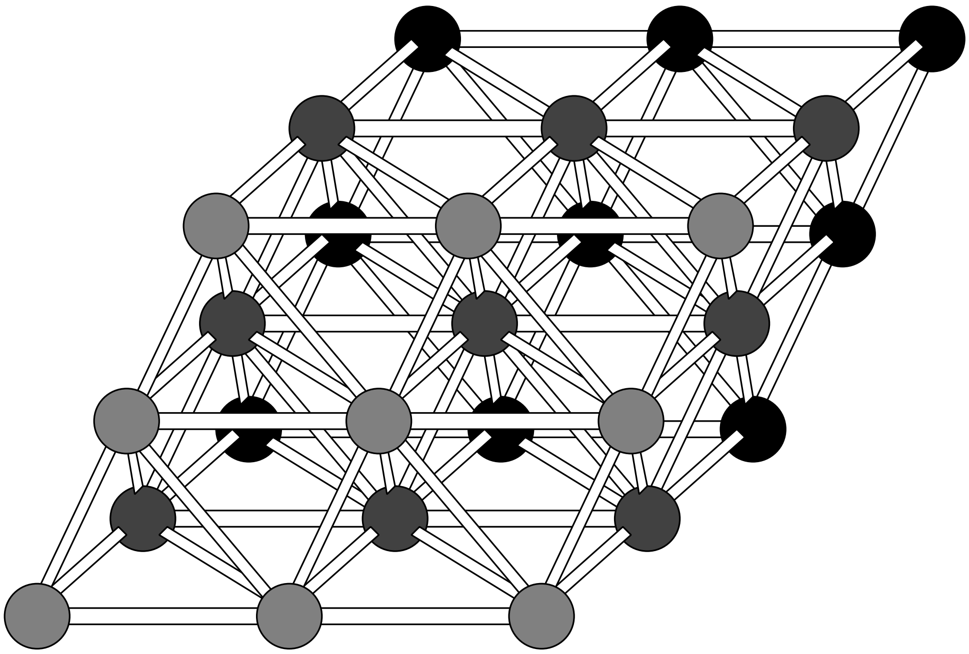

Figure 2.1: A small lattice computer based on A_3.

of them. Regularity is a strong symmetry condition. Z^n and A_n are examples of regular n-dimensional lattices. It is shown in [CRS90a, CRS91c] that A_n has the maximal number of nearest neighbors ($n(n+1)$) among the regular n-dimensional lattices! Hence, there is special interest in lattice computers based on A_n.

2.1.3 Guaranteeing Appropriate Algorithms for Motion

Besides having architectures for representing space well, it is crucial to have approximately linear in real time algorithms for the simulation of basic motions. For certain lattice computers, including those based on A_n, [She91, CRS91b, CRS91a] present algorithms for constant speed motion of, single, non-dissipating wave-fronts and particles for certain lattice computers. It is now known [CRS92] that these algorithms extend to a very wide and important class of lattices, known as the *root lattices* [CN93, RS91]. We will describe below a characterization of such lattices completed by [RS91] (see also [CN93]).

In a lattice computer we think of each processor, sitting on a lattice point P, as responsible for the points of euclidean n-space as least as close to P as to the lattice point of any other processor. These regions of processor responsibility are called *Voronoi cells* [GL87, Aur90, RS91, CN93]. Voronoi cells are crucial for most of the algorithms developed so far (see, for example, Section 2.2 just below). The n-dimensional lattices whose Voronoi cells have inscribed n-spheres characterize the well-known root lattices (possibly rescaled). It is now known by our research group that the only n-dimensional lattices that are both root and regular are Z^n and A_n, again making A_n particularly interesting.

2.2 Single Particle Motion Algorithm

Definition 2.2.1 *We define* MinLgth($\mathcal{L}$) *to be the length of a minimal length vector in lattice $\mathcal{L}$.*

The basic computational model envisages synchronous processor execution. A basic computation step executed by each processor is the *pass*. A pass essentially involves receiving messages, local processing of received messages and then sending out messages. The messages passed between processors contain information about the particle. The code executed in a single pass is the same for any processor in the system. The starting and ending of the passes are synchronized. Each pass takes the same constant amount of time to execute (we use this to synchronize the execution of the passes and to simulate the particle motion in approximate real time). A pass can roughly be described with the following Pseudo code.

- Receive and store all the incoming messages.
- Perform local computations to determine whether the particle should be represented in the current pass by the processor performing the computations.
- Perform local computations to find the nearest-neighbor processors to which messages should be sent at the end of the current pass for the simulation to proceed.
- Send out messages in the appropriate directions.

A processor repeatedly executes the above code (with possible waits for synchronization at the beginning and the end of the above code). The lattice computer algorithm is specified by the code to be executed in a single pass.

As specified before, a processor P represents all the particles that are present in the Voronoi cell of P. A data structure is used to implement the concept of representation. Simplistically, one can assume that any particle present in the data structure is being represented by the processor. In the algorithms presented later, we will see variations in this definition of representation. We note that a processor may receive messages about a particle that it does not need to represent. Messages are used to pass information between processors about the *possible* presence of particles; whereas, representation is used to signify actual presence of particles.

In this section we will look at the important issues in the simulation of motion of a particle at constant speed, in the architecture described above. This will give the background required by the user to understand the algorithms to be presented in the next few chapters.

Suppose the curve traced by the particle is presented as a function ρ of time. We select points from these curves such that

- the first point, denoted by $\rho_s(0)$, is $\rho(0)$, and
- the i^{th} point, denoted by $\rho_s(i)$, is a point on the curve such that the euclidean distance between $\rho_s(i)$ and $\rho_s(i-1)$ is equal to a constant $s <$ MinLgth($\mathcal{L}$).

The adjacent points in the set of selected points are separated by a constant distance, s. To simulate the particle at a constant speed, we choose to make sure that the simulation of motion in the line segment between adjacent points is done within a constant amount of time. A basic computation step carried out by any processor is the pass. Therefore, we allocate a constant (say C) number of consecutive passes to simulate the motion along the line segment between adjacent points. The motion along the line segment between the first point, $\rho_s(0)$, and the next point, $\rho_s(1)$, is simulated in passes $0, \ldots, C-1$, the motion along the line segment between the $\rho_s(1)$ and $\rho_s(2)$ is simulated in the passes $C, \ldots, 2*C-1$, and so on.

The next important issue that arises is the selection of the constant C. We note that the constant C should be selected in such a way that

- the number of passes selected thus is enough to simulate the motion of the particle across the entire line segment, and
- C is the least such required number.

The communication of messages can only be between adjacent processors. Therefore, if a single line segment l of a particle p crosses the Voronoi cells of n processors, we have that, information about the particle p must reach all these n processors within C passes (as after these C passes, we simulate the next line segment). In the worst case, we can have these n processors form a chain such that no more than two of the processors are adjacent to each other. Then, a message about the particle p in the first processor in the chain will take n passes to reach the final processor in the chain. Therefore, we require

$$n \leq C. \tag{2.1}$$

[She91] shows that there is a constant, $PathLgthDil$, such that the line segment of length s can cross the Voronoi cells of a maximum of $PathLgthDil$ processors. The value of $PathLgthDil$ depends on the type and the dimension of the lattice on which the architecture is based. Therefore, from (2.1), we have that

$$PathLgthDil \leq C. \tag{2.2}$$

Since we want to select the minimum possible value for C, we assign $PathLgthDil$ to C; hence, we simulate the motion along each single line segment within *exactly* $PathLgthDil$ passes. We simulate the k^{th} line segment of the particle from passes $PathLgthDil * (k-1)$ through $PathLgthDil * k - 1$.

The messages passed between processors contain information about the particle being simulated. A variable (say, k) is used to maintain the number of the line segment to be simulated in the current pass. This is obtained by the integer division of the current pass number by $PathLgthDil$. When a message about the particle is received by a processor P, P checks to find whether the line segment, of the particle currently being simulated, intersects the Voronoi cell of P. If the line segment intersects the Voronoi cell of P, P starts representing the particle.

There are two cases to be considered at this point. The first case is when processor P contains the right end point of the line segment currently being simulated. In this case, P needs to send a message to the appropriate processor only in pass $PathLgthDil * k - 1$, since we are sure that there is no processor *after* P that needs to simulate the line segment k. So, processor P waits till the pass number reaches $PathLgthDil * k - 1$ and sends the message about the particle to the appropriate nearest-neighbor processor.

The second case to be considered is when the processor P does not contain the right point of the line segment currently being simulated. In this case, P is in the middle of a chain of processors that have to represent the current line segment. Therefore, P needs to send a message in the current pass to its appropriate nearest-neighbor processors. The appropriate nearest-neighbor processors to which messages have to be sent cannot be obtained accurately. [She91, CRS91a] present a method by which one can get a set of nearest-neighbor processors S, from the location of the end points of the current line segment, such that S is a superset of the set of all processors that should receive the message. The message about the particle is sent to all the processors in S. The processors in the set S, to which messages should not have been sent by processor P, ignore the message after finding that they do not have the particle in their Voronoi cell.

The code and associated proofs are presented in detail in [She91, CRS91a].

Chapter 3

Preliminaries

In this chapter we will present some notation and the intuition behind the development of our new algorithms. In the previous chapter, we briefly explained the algorithm to simulate the motion of a single particle from [She91, CRS91a]. The present paper extends this algorithm to simulate the motion of a set of particles. As in the single particle algorithm, an approximate form of the curve traced by each particle is used in the simulation. The approximation is obtained by dividing the curve traced by the particle into line segments of length equal to a constant s. We require $s < \mathrm{MinLgth}(\mathcal{L})$. We denote the maximum number of nearest-neighbor processors any processor can have by N_{nnp}. We note that N_{nnp} is a constant dependent on the lattice $\mathcal{L}$.

We assume that the total number of particles being simulated is N (> 1). For all particles p, and any positive integer k, the starting point of the k^{th} line segment in the approximation to the curve traced by the particle p is $\rho_s(p, k-1)$. The k^{th} line segment of the approximation to the curve traced by a particle p is denoted by $\mathrm{ls}(p, k)$. Formally, $\mathrm{ls}(p, k)$ is the line segment from the point $\rho_s(p, k-1)$ to the point $\rho_s(p, k)$ including $\rho_s(p, k-1)$, but excluding $\rho_s(p, k)$.

In the course of the simulation it is also assumed that the maximum number of particles that can be present in the Voronoi cell of any processor at any point of time is a constant N_{crowd}. Physically this assumption represents that there is a limit on how many particles will fit into a bounded region of space, for example, a Voronoi cell. Furthermore, we cannot guarantee correctness of our algorithms if the simulation lets more than N_{vor} particles into a single Voronoi cell. Physically natural extensions of our algorithms which resolve *crowding* problems are left to future work. Importantly, the algorithms developed thus far should provide a sound and natural basis for these and other extensions. Our algorithms thus far developed also easily admit extensions which handle particle *collisions* in any of various physically natural ways. For this paper, though, it is convenient to imagine particles which share a common region of space during some time as merely ectoplasmically passing through one another.

We require our simulations to satisfy certain reasonable timing constraints. In Section 3.1, we will look at these conditions and the rationale behind them. As an aid to exposition, in Sections 3.2, refd-f-ext, and 3.4, we will outline the development of the algorithms.

3.1 Timing Conditions

Definition 3.1.1 *[She91] Consider any processors P and Q. We define* PathLength(P,Q) *to be the minimum number of immediate connections that has to be traversed by a message passing from P to Q.*

Assuming that processors receive constant size messages at the beginning of a pass and send the same constant size messages out at the end of a pass, $PathLgth(P,Q)$ is also numerically equal to the minimum number of passes such a message sent from P takes to reach Q.

Definition 3.1.2 *Suppose p is a particle in the simulation and k is a positive integer.* Head(p,k) *is defined to be the set of processors P such that* Voronoi$(P,\mathcal{L})$ *contains $\rho_s(p,k-1)$.*

Definition 3.1.3 *Suppose p is a particle in the simulation and k is a positive integer. We define* Furthermost(p,k) *to be the set of processors, P, such that*

1. $\mathrm{Voronoi}(P,\mathcal{L}) \cap \mathrm{ls}(p,k) \neq \emptyset$, *and*
2. $\min_{h \in \mathrm{Head}(p,k)} \mathrm{PathLength}(P,h) = PathLgthDil$.

Intuitively, $\mathrm{Furthermost}(p,k)$ contains exactly the processors that intersect $\mathrm{ls}(p,k)$ and are located such that they receive a message about the particle p, $PathLgthDil$ passes after a processor in $\mathrm{Head}(p,k)$ has received a message about p.

As in the the single particle case, there is a constant c such that the interval of passes $[c*(k-1), c*k)$ is the interval in which we simulate the k^{th} line segment of every particle. In the lattice computer, during these passes, a processor P represents the particles present in its Voronoi cell. Consider the k^{th} line segment of a particle p, $\mathrm{ls}(p,k)$. In our simulation,

- if the k^{th} line segment of the curve traced by the particle p intersects the Voronoi cell of the processor P, and
- $P \notin \mathrm{Furthermost}(p,k)$,

we ensure that P represents the k^{th} line segment of particle p during some nonempty subinterval of consecutive passes in the interval of passes $[c*(k-1), c*k)$. In fact, P will represent $\mathrm{ls}(p,k)$ exactly during every pass in the interval of passes $[pc_{start}, pc_{end}]$, where pc_{start} and pc_{end} are pass numbers dependent on p, P and k and satisfy all the following five conditions. These conditions will ensure that our simulation of uniform motion runs approximately in real time.

1. $[c*(k-1), c*k) \cap [pc_{start}, pc_{end}] \neq \emptyset$.

2. $0 \leq pc_{start} - c*(k-1) \leq \epsilon$, where ϵ is a constant independent of p, P and k. For the FE algorithm, $\epsilon = PathLgthDil$ and for the SE algorithm, $\epsilon = PathLgthDil + 1$. This means that p is not represented by P before p reaches the Voronoi cell of the processor and that P starts representing p within some constant number of passes after p *actually* enters the Voronoi cell of P. Ideally we would want ϵ to be zero, i.e., we would want P to start representing p as soon as p enters the Voronoi cell of P.

3. $c*k-1 \leq pc_{end}$. This means that P does not stop representing the particle p until p has moved onto $\mathrm{ls}(p, k+1)$.

4. if the starting point of the line segment $k+1$ does not lie in the Voronoi cell of P, then $pc_{end} - (c*k-1) < \delta$, where δ is a positive constant independent of P, p and k. For the FE algorithm, $\delta = 1$ and for the SE algorithm, $\delta = 2$. This means that P stops representing p within a constant number of passes after the p has left the Voronoi cell of P.

5. if the Voronoi cell of processors P_1 and P_2 intersects the k_1^{th} and k_2^{th} line segments of the particle p, respectively, and the processors start simulating p from passes pc_{start1} and pc_{start2}, respectively, we have that the following condition is satisfied.

$$(pc_1 < pc_2) \Longrightarrow (pc_{start1} < pc_{start2}).$$

   This condition means that processors start representing p in the temporal sequence in which it entered the Voronoi cell of the processors.

We would like the Conditions 1 through 5 to be true for *all* processors P such that the Voronoi cell of processor P intersects the k^{th} line segment of a particle p. In the interest of minimizing the execution time of a pass as much as possible, we do not enforce Conditions 1 through 5, in the case when the Voronoi cell of processor P intersects $\mathrm{ls}(p, k)$ *but* $P \in \mathrm{Furthermost}(p, k)$. We note that the Voronoi cell of every such processor $P \in \mathrm{Furthermost}(p, k)$ contains the left end point of $\mathrm{ls}(p, k+1)$ and thus simulates in pass $c*k$, an initial portion of $\mathrm{ls}(p, k+1)$. So, there is no measurable loss of continuity in the simulated motion of the particle.

3.2 Elimination of Computations

In the algorithms to be presented, each message passed between processors contains information about only one of the particle. We say it is about that particle p. A processor can send messages to any of its nearest neighbor processors and to itself. A message sent by a processor to itself is called a *self-message*. We use

$\vec{0}$ to represent the direction in which self-messages are sent. When a processor P sends a self-message in pass k, P receives that message in pass $k+1$.

In the single particle algorithm discussed in the previous chapter, we said that each pass proceeds as follows

1. receive messages,
2. find whether the particle is to be represented,
3. if the particle is to be represented, compute the appropriate nearest-neighbors to which messages about the particle needs to be sent, and
4. send messages to these nearest-neighbor processors.

We observed that in Step 3 of the above process, only an approximation (a superset of the actual set of processors to which messages need to be sent) to the set of nearest-neighbor processors to which messages need to be sent is computed. Any processor P, in the set of processors to which the messages need not be sent, ignores the message in the next pass after finding that the message is not about a particle P needs to represent.

This suggests that we might eliminate the computation of Step 3 completely and send the messages to *all* the nearest-neighbor processors. Any processor that does not need to receive the message, ignores the message as in the single particle algorithm. We remind the reader that though more work needs to be done, the overall time to do the computations does not increase as the computations are distributed on different processors. This idea will be carried over to the algorithms to be presented in this paper.

3.3 Developing the First Extension

The first and the most naive idea that comes up when trying to extend the single particle algorithm to multi-particles is to run the version of single particle algorithm on *all* the messages received, in *every* pass. It may be necessary for a single processor to receive N messages in a single pass. Therefore, we will have to effectively run the single particle algorithm on each of the N messages received in this naive algorithm (as we want the passes to be synchronized). This method works, but is awfully slow.

We note that physically it is typically impossible for more than a certain number of particles to be present in a bounded region (for our purposes, say the Voronoi cell of a processor). Therefore, it is unlikely in simulations of such physical events that a processor receives messages about every particle being simulated, in a single pass. We recall that there can be a maximum of N_{crowd} particles that can be present in the Voronoi cell of a processor at any point of time. Let $N_{\text{vor}} = min(N, N_{\text{crowd}})$. Therefore, at the end of any pass pc, each processor sends messages about at most N_{vor} particles to all its nearest-neighbor processors.

Consider a processor P in pass $pc+1$. P can receive messages from

- all N_{nnp} nearest-neighbor processors of P, and
- itself.

Therefore, the maximum number of messages the processor P can receive in pass $pc+1$ is $\min(N, N_{\mathrm{vor}} * (N_{nnp}+1))$. Therefore, in a single pass in the FE algorithm we need to execute a single pass in the single particle algorithm, at least
$\min(N, N_{\mathrm{vor}} * (N_{nnp}+1))$ times.

Any processor P executing the FE algorithm selects a maximum of $\min(N, N_{\mathrm{vor}} * (N_{nnp}+1))$ distinct messages from the set of incoming messages in the beginning of any pass. Then, P checks each of the distinct messages to see if its associated particle is present in the Voronoi cell of P. Messages about the particles which are actually present are sent out by P, at the end of the pass, to all its nearest-neighbor processors and to itself. The FE algorithm in pseudo code and the proofs for the correctness of the algorithm are presented in detail in Chapter 4.

3.4 Developing the Second Extension

Having established that one cannot expect to perform better using the above model (in which every processor checks for representation of *every* message received), we tried a different approach, where we attempt to reduce the execution time of the simulation by checking only a subset of messages (to be specific, we just check a maximum of the ceiling of half the number of total particles, N) received in a single pass, for representation.

The first observation we make is that when $\lceil N/2 \rceil > N_{\mathrm{vor}} * (N_{nnp}+1)$, this approach could require us to check a maximum number of messages greater than $N_{\mathrm{vor}} * (N_{nnp}+1)$, for representation in each pass. The FE algorithm, which checks a maximum of $N_{\mathrm{vor}} * (N_{nnp}+1)$ messages, then performs better in this situation. When $N = 2 * N_{\mathrm{vor}} * (N_{nnp}+1)$, the number of messages that are checked for representation is the same for both the algorithms. Therefore, the SE algorithm developed from this approach, is an alternative to the FE algorithm and is to be applied only when $\lceil N/2 \rceil < N_{\mathrm{vor}} * (N_{nnp}+1)$. When $\lceil N/2 \rceil < N_{\mathrm{vor}} * (N_{nnp}+1)$, the number of messages that are checked for representation by the FE algorithm is $\min(N, N_{\mathrm{vor}} * (N_{nnp}+1))$ and the number of messages that are checked for representation by the SE algorithm is $\lceil N/2 \rceil$. In this case,

$$\lceil N/2 \rceil < \min(N, N_{\mathrm{vor}} * (N_{nnp}+1)),$$

and therefore, the SE algorithm checks for fewer messages than the FE algorithm.

Following the idea of checking only a proper subset of the received messages for representation, we encounter several different issues. The development of the SE algorithm essentially involved noting some of the issues and resolving some others in a reasonable way. We will look at the different issues in the next section.

3.4.1 Issues

The main issues that were encountered in the development of the SE algorithm are presented in this section.

- **Selection of the subset.** The most important issue that came up was the criteria to be used for the selection of the subset of messages, from the set of distinct received messages, to be checked for representation. The selection of the subset of messages should be fair, i.e., there should be a constant $n > 0$, such that no message about a particle p is ignored[1] for more than n consecutive passes. We incur some inaccuracy when we choose to ignore a subset of messages. This inaccuracy manifests itself in two forms. In the first form we have consecutive passes during which a particle p that may not be present in the Voronoi cell of a processor P is nonetheless represented by P. The second form manifests in terms of consecutive passes in which a particle that may be present in the Voronoi cell of P is not represented by P.

  If a message about a particle p is ignored in pass pc by a processor P, all the processors receiving a message about particle p in pass pc ignore the message. We divide the particles into two roughly equal size partitions S_0 and S_1 (the exact division is specified in Chapter 5). Messages about particles in the set S_0 are ignored in any odd pass and messages about particles in the set S_1 are ignored in any even pass.

- **Processing ignored messages.** Since we ignore messages about particles in partitions S_0 and S_1 in the odd and even passes respectively, we can be sure that any message about a particle is not ignored for more than one pass. We note that a message that is ignored in a pass by a processor P might be about a particle p that is *actually* present in the Voronoi cell of the processor. Then, at the end of that pass, a message about p might need to be sent to its nearest-neighbor processors. In this case P sends a message about all the particles that were not checked for representation to all its nearest-neighbor processors and to itself.

- **Making a correction.** The basic correctness of the SE algorithm lies in the fact that any processor P whose Voronoi cell intersects the k^{th} line segment of a particle p must represent the particle p and vice-versa.

[1] not checked for representation.

Since we do not check all the particles about which messages were received for representation, it is possible that a processor P (whose Voronoi cell intersects the k^{th} line segment of the particle p) receives a message about particle p and ignores the message. Therefore, it is possible that the processor P never represents the particle. To avoid this possibility, we increase by one, the number of passes during which a line segment is simulated.

In this case, all the processors whose Voronoi cells intersect the line segment of a particle p receive a message about p at least one pass *before* they start simulating the particle p in its next line segment. Since messages about particles are checked for representation at least once in any two consecutive passes, any such processor receiving a message about particle p represents the particle p for at least one pass.

This summarizes the development of the SE algorithm. The details of the code and the associated proofs are presented in Chapter 5.

Chapter 4

The FE algorithm

In this chapter, we will present the FE algorithm and the analysis for proving its correctness. First, we will present the definitions of some variables and constants that will be made use of in the FE algorithm.

4.1 Definitions

Let us define the following variables and constants before proceeding to look at the FE algorithm in Section 4.2.

- k is a variable whose value is the index of the line segment of the particles currently being simulated by any processor. The first line segment has an index of zero.
- *SimClock* is the variable which contains the current clock time of the system. The system clock is the finest resolution, discrete level clock available with the system. This variable is assumed to be updated automatically by the system.
- *CompLen* is an algorithm dependent constant used in synchronizing the passes. The value of *CompLen* is equal to the maximum time (as measured by the system clock) any processor might take to execute a pass as specified by the FE algorithm.
- p is a variable ranging over the particles in the simulation.
- *Info(p)* contains all the relevant information about particle p, that a processor executing the FE algorithm needs. Info(p) may be updated by the processors in the course of a simulation. Given a particle p, there is an unique Info(p) associated with the particle.

- *passcount* is a variable whose value is the number of passes executed by any processor. The variable has a value of 0 at the beginning of the simulation.
- *Repr* is a data structure local to each processor that stores the information about particles. This data structure is used in defining the concept of the processor *representing* a particle. *Repr* can be accessed only by the following operations:
  - AddToRepr(p) : Modify the data structure to add particle p, starting from the current pass.
  - DeleteElements($passcount$) : Modify the data structure to delete all elements that were present in the data structure in the previous pass, but were found not to be present in the Voronoi cell of the processor in the current pass.
  - IsRepresenting(p) : Returns 'true' if the particle p has been added to *Repr* using the AddToRepr(p) operation and has not been deleted since then by a DeleteElements($passcount$) operation.
- M_{max} is the constant which bounds the number of particles whose messages are processed by any processor in a single pass. We will show that M_{max} can be assigned any value greater than or equal to $\min(N, N_{\mathrm{VOR}} * (N_{nnp} + 1))$.
- n_{recvd} is a variable whose value is the number of particles about which messages were received by any processor in the current pass.
- *storage* is a data structure representing a set. The components of the set are Info(p). The data structure is used to store information from messages received in the current pass. The data structure can store a maximum of M_{max} components.
- *outbuffer(D)* is the buffer for storing outgoing messages along direction D. The messages essentially consist of Info(p). A maximum of M_{max} messages can be held in this output buffer.

4.2 Pseudo Code

The FE algorithm is started up by an external agent sending messages about each particle p to all the processors P whose Voronoi cell contains $\rho_s(p, 0)$.

1. Let $pd = \{\ p \mid p$ is a particle such that a message about particle p was received in the current pass$\}$;
2. For each particle p in the set pd do
   (a) Add Info(p) to *storage* and increment n_{recvd}.

3. For each particle p such that Info(p) is in *storage* do
   (a) If $[\rho_s(p,k), \rho_s(p,k+1)) \ \cap$ Voronoi$(P, \mathcal{L}) = \emptyset$,
      i. Delete Info(p) from *storage*.
      ii. Decrement n_{recvd}.
4. For all particles p such that Info(p) is in *storage* do
   (a) AddToRepr(p).
   (b) Delete Info(p) from *storage*.
   (c) Decrement n_{recvd}.
   (d) For each direction D in $\mathcal{D} \cup \{\vec{0}\}$, add Info($p$) to outbuffer($D$).
5. Increment *passcount*.
6. Assign (*passcount* DIV *PathLgthDil*) to k.
7. DeleteElements(*passcount*).
8. Idle till *SimClock* has the value $(CompLen * passcount) - 1$.
9. Simultaneously send all messages in outbuffer.

4.3 Analysis

The correctness of the FE algorithm is proved in this section. Before going into the details of the correctness of the algorithm regarding timing and functionality, we will show that the above code always halts (in each pass).

We can see that the only place where the code might go into an infinite loop is in the for-loops. Since all these for-loops execute for a maximum of M_{max} times, we have a constant bounding the number of times the for-loops execute. Therefore, there is a bound on the maximum amount of time a pass might take to complete (which is *CompLen*). Therefore, the code always halts within finite time in each pass.

Definition 4.3.1 *The messages corresponding to the particles that fail the test in Step 3 are referred to as* false-messages*.*

Definition 4.3.2 *There is an* overflow of messages *in pass $pc+1$ iff some processor P receives more than M_{max} messages in pass pc.*

The correctness of the FE algorithm is proved in two steps. First we prove that a value of $\min(N, N_{\mathrm{VOR}} * (N_{nnp}+1))$ assigned to M_{max} makes sure that the number of messages received by any processor in any pass never exceeds M_{max}, and, therefore, for this value of M_{max}, messages from one pass of the processors do not spill over to the next pass.

Finally, we want to prove that the FE algorithm does simulate the particles *approximately* linear in real time. A processor P is said to *represent* a particle p in a pass interval $[a, b]$ iff the IsRepresenting(p) operation returns the value 'true' in processor P at the end of every pass in the pass interval $[a, b]$.

4.3.1 Proof of Non-Overflow of Messages

We will prove that when M_{max} is assigned the value $\min(N, N_{\text{VOr}} * (N_{nnp} + 1))$, there is no overflow of messages.

Theorem 4.3.1 *If* M_{max} *is assigned the value of* $\min(N, N_{vor} * (N_{nnp} + 1))$, *every processor executing the code in Section 4.2 receives less than or equal to* $\min(N, N_{vor} * (N_{nnp} + 1))$ *messages in any pass.*

Proof: Suppose M_{max} is assigned the value $\min(N, N_{\text{VOr}} * (N_{nnp} + 1))$. We will use induction over passes to prove the theorem.

Base case: The FE algorithm initializes the processors with messages about the particles present in the Voronoi cell of the processors in pass 0. As we assume that there cannot be more than N_{VOr} particles in the Voronoi cell of any processor at any point of time, we have that no processor receives more than N_{VOr} ($< M_{max}$) messages in pass 0. Therefore no processor receives more than M_{max} messages in pass 0.

Inductive step: Suppose no processor received more than M_{max} messages in pass $pc - 1$. There are two cases that will be considered in the simulation.

- **Case 1** : $\mathbf{N} \geq \mathbf{N_{vor}} * (\mathbf{N_{nnp}} + \mathbf{1})$. We have $M_{max} = N_{\text{VOr}} * (N_{nnp} + 1)$. Consider any processor P in pass $pc - 1$. By the inductive hypothesis, we have that processor P received no more than M_{max} messages in pass $pc - 1$. Consider the maximum number of messages that could be sent out by the processor P in pass $pc - 1$ in any direction D.

  Processor P eliminates all the false-messages in the set of incoming messages and determines the particles that are *actually* to be simulated. Messages are sent out about the remaining particles in the end of the current pass to all the nearest-neighbor processors and to itself. We assumed that a maximum of N_{VOr} particles can be present in the Voronoi cell of any processor at any point of time. Therefore we have that processor P can send a maximum of N_{VOr} messages to any nearest-neighbor processor at the end of pass $pc - 1$.

  In pass pc, a processor P can receive only the self-messages sent out by processor P in pass $pc - 1$ and the messages sent out by the nearest-neighbor processors of P in pass $pc - 1$. Thus the maximum of total number of messages that processor P can receive in pass pc is equal to $N_{\text{vor}} * (N_{nnp} + 1)$. This value is equal to M_{max} and therefore the total number of messages received by any processor in pass pc is less than or equal to M_{max}.

- **Case 2** : $\mathbf{N < N_{vor} * (N_{nnp} + 1)}$. We have $M_{max} = N$. It is clear that the maximum number of messages received by any processor in any pass can never exceed the total number of particles present in the system. This proves that the maximum number of messages received by any processor in any pass is less than or equal to M_{max} $(= N)$. $\square$

4.3.2 Timing Constraints

From Definition 3.1.1, we have

- PathLength(P,P) $= 0$, and
- (PathLength(P',P) $= d + 1$) $\Longrightarrow \exists Q$ such that PathLength(P',Q) $= d$ and P is a nearest-neighbor of Q.

If there exists a particle p and a k such that $\text{ls}(p, k)$ intersects both Voronoi$(P, \mathcal{L})$
and Voronoi$(Q, \mathcal{L})$, by definition, PathLength(P,Q) $\leq PathLgthDil$.

We want to simulate the motion of the particles, along the respective curves, at constant speed. [She91] shows that a single line segment in the approximation of the curve traced by the particle can cross a maximum of $PathLgthDil$ Voronoi cells.

We note that, while the maximum number of Voronoi cells that can be crossed a line segment, $\text{ls}(p, k)$, is a constant, the number of Voronoi cells crossed by a line segment $\text{ls}(p, k)$ is not a constant. We delay the number of passes needed to simulate any single line segment by the maximum number to ensure that the simulation proceeds at constant speed. Therefore, we simulate the k^{th} line segment of the curve traced by a particle in the pass interval

$$[PathLgthDil * (k - 1), PathLgthDil * k). \tag{4.1}$$

Part of proving the correctness of the FE algorithm involves proving that a processor receives a message about a particle p when it is to start representing particle p. Consider $\text{ls}(p, k)$ for some particle p and some index k. The following two theorems prove that any processor P such that $P \notin \text{Furthermost}(p, k)$ and $\text{Voronoi}(P, \mathcal{L}) \cap \text{ls}(p, k) \neq \emptyset$ receives a message in the pass interval given by (4.1).

These theorems will be used in proving that the FE algorithm satisfies all the conditions listed in Section 3.1 with $c = PathLgthDil$.

Theorem 4.3.2 *Consider a processor P such that for a particle p and some k,* *1.* $\text{Voronoi}(P, \mathcal{L}) \cap \text{ls}(p, k) \neq \emptyset$*,*

2. $P \notin \text{Head}(p, k)$*, and*

3. $P \notin \text{Furthermost}(p, k)$*.*

If all the processors in $\text{Head}(p, k)$ *receive a message about the particle* p *in pass* $PathLgthDil * (k-1)$*, then this processor* P *receives a message about the particle* p *in some pass in the pass interval*

$$(PathLgthDil * (k-1), PathLgthDil * k).$$

Proof: Let $d = \min_{h \in \text{Head}(p,k)}(\text{PathLength}(h, P))$. Let $\text{PathLength}(H, P) = d$, and $H \in \text{Head}(p, k)$. The Voronoi cell of both processors P and H intersect $\text{ls}(p, k)$. Also, as $P \notin \text{Furthermost}(p, k)$, $d \neq PathLgthDil$. Therefore,

$$0 < d < PathLgthDil. \tag{4.2}$$

Consider the pass $pc = PathLgthDil * (k-1) + d$. We will show later in the proof that processor P receives a message about particle p in pass pc.

Adding $PathLgthDil * (k-1)$ to each term in (4.2) says

$$PathLgthDil * (k-1) < pc < PathLgthDil * k,$$

which proves that processor P receives a message in some pass in the pass interval (PathLgthDil*(k-1), PathLgthDil*k).

All that remains to prove is that the processor P receives a message about the particle p in pass $pc = PathLgthDil * (k-1) + d$. We will prove this by induction.

Base case: d = 0. This says that the processor P we are considering belongs to $\text{Head}(p, k)$. The above statement is vacuously true any processor $\in \text{Head}(p, k)$.

Inductive step: Let the above statement be true for all d such that

$$0 \leq d < m < PathLgthDil.$$

We will prove that any processor P, such that

$$\text{ls}(p, k) \cap \text{Voronoi}(P, \mathcal{L}) \neq \emptyset \textit{ and}$$
$$\min_{h \in \text{Head}(p,k)} (\text{PathLength}(h, P)) = m,$$

receives a message about particle p in pass $PathLgthDil * k + m$.

Let $H \in \text{Head}(p, k)$ and $\text{PathLength}(H, P) = m$.

$(\text{PathLength}(H, P) = m) \Longrightarrow (\exists Q$ whose $\text{PathLength}(H, Q) = m - 1$ and Q is a nearest-neighbor of P and $\text{ls}(p, k)$ intersects $\text{Voronoi}(Q, \mathcal{L}))$.

By the inductive assumption, we have that the processor Q receives a message in pass $m-1+PathLgthDil*(k-1)$ about the particle p. The processor Q checks to see if $\mathrm{ls}(p,k)$ intersects $\mathrm{Voronoi}(Q,\mathcal{L})$ and finds that it does. At the end of the pass, processor Q sends a message about the particle p to all the nearest-neighbor processors of Q. Processor P, being a nearest-neighbor of Q, receives a message about the particle p in the next pass, m. $\square$

Theorem 4.3.3 *Consider any processor P, some particle p and some k such that*

1. $\mathrm{Voronoi}(P,\mathcal{L}) \cap \mathrm{ls}(p,k) \neq \emptyset$.
2. $P \in \mathrm{Head}(p,k)$.

All such processors P receives a message about the particle p in pass

$$PathLgthDil*(k-1).$$

Proof: We will use induction on k to prove this theorem.

Base case: k=1. The FE algorithm is initialized by an external agent sending messages to all the processors whose Voronoi cells contain the starting point of $\mathrm{ls}(p,1)$. Therefore, all such processors receive a message about the particle p in pass 0 $(= PathLgthDil*(k-1))$.

Inductive step: Let us assume that the given is true for all k such that $1 \leq k < m$. We will prove the given statement for $k=m$.

Consider any processor P such that $P \in \mathrm{Head}(p,m)$. Consider the processor Q' such that

- $\mathrm{Voronoi}(Q',\mathcal{L}) \cap \mathrm{ls}(p,m-1) \neq \emptyset$, and
- $\mathrm{MinMgsTim}(Q',P) = 1$.

There are two cases to be considered.

- **Case 1:** $Q' \in \mathrm{Head}(p,m-1)$**.** By the inductive assumption we have that Q' receives a message about particle p in the pass $pc = PathLgthDil*(m-2)$. Therefore we have

$$PathLgthDil*(m-2) \leq pc < PathLgthDil*(m-1). \tag{4.3}$$

- **Case 2:** $Q' \notin \mathrm{Head}(p,m-1)$**.** From Theorem 4.3.2, we know that Q' receives a message about particle p in some pass pc such that

$$PathLgthDil*(m-2) < pc < PathLgthDil*(m-1). \tag{4.4}$$

From (4.3) and 4.4), we have that the processor Q' receives a message about the particle p in some pass pc such that

$$PathLgthDil * (m-2) \leq pc < PathLgthDil * (m-1).$$

The processor Q' executes the code in Section 4.2 and sends messages to itself and to its nearest neighbor processors till the pass number (being executed) reaches $PathLgthDil*(m-1)-1$. This shows that processor Q' sends a message about particle p to all its nearest-neighbor processors in pass $PathLgthDil * (m-1)-1$. P, being a nearest-neighbor processor of Q', receives a message about particle p in the next pass, $PathLgthDil * (m-1)$, which is what is to be proved. □

.

Proof of Timing Constraints

The following two theorems prove that the FE algorithm satisfies the timing conditions given in Section 3.1 with $c = PathLgthDil$. Hence, the FE algorithm satisfies the timing requirements for the simulation. The rationale for these conditions have been presented in Section 3.1.

Theorem 4.3.4 *Consider a processor P, a particle p and some k such that*

- $\mathrm{Voronoi}(P, \mathcal{L}) \cap \mathrm{ls}(p,k) \neq \emptyset$, *and*
- $P \notin \mathrm{Furthermost}(p,k)$.

Let $[pc_{start}, pc_{end}]$ be the maximal pass interval in which processor P represents p in line segment $\mathrm{ls}(p,k)$. Then

$$[PathLgthDil * (k-1), PathLgthDil * k) \cap [pc_{start}, pc_{end}] \neq \emptyset, \tag{4.5}$$

$$\exists \epsilon \text{ (an integer } > 0) \text{ such that } 0 \leq pc_{start} - PathLgthDil * (k-1) \leq \epsilon, \tag{4.6}$$

$$pc_{end} \geq PathLgthDil * k - 1 \text{ and} \tag{4.7}$$

$$\text{if } \mathrm{ls}(p, k+1) \cap \mathrm{Voronoi}(P, \mathcal{L}) \neq \emptyset,$$

$$\exists \delta > 0 \text{ such that } pc_{end} - (PathLgthDil * k - 1) < \delta. \tag{4.8}$$

Furthermore, we may choose $\epsilon = PathLgthDil$ and $\delta = 1$ in (4.6) and (4.8) respectively.

Proof: In this theorem we will prove stronger versions of (4.6) and (4.8). Specifically, we will show later in the proof the following two inequalities (which automatically prove (4.6), with $\epsilon = PathLgthDil$ and (4.8), with $\delta = 1$).

$$0 \leq pc_{start} - PathLgthDil * k \leq PathLgthDil \text{ and} \tag{4.9}$$

$$\text{if } \mathrm{ls}(p, k+1) \cap \mathrm{Voronoi}(P, \mathcal{L}) \neq \emptyset,$$

$$pc_{end} - (PathLgthDil * k - 1) < 1. \tag{4.10}$$

From Theorem 4.3.2 and Theorem 4.3.3, we know that processor P receives a message about particle p in some pass pc such that

$$PathLgthDil * (k-1) \leq pc < PathLgthDil * k. \tag{4.11}$$

The processor P checks to find that the particle needs to be represented and represents the particle at least from pass pc.

Therefore, we have

$$pc_{start} \leq pc. \tag{4.12}$$

We know that a processor cannot represent a particle p in line segment k, before the pass $PathLgthDil * (k-1)$. Therefore, we have,

$$PathLgthDil * (k-1) \leq pc_{start}. \tag{4.13}$$

From (4.13) and (4.11), we have that

$$[PathLgthDil * (k-1), PathLgthDil * k) \cap [pc_{start}, pc_{end}] \neq \emptyset.$$

Thus (4.5) is proved. From (4.11), we know that $pc_{start} < PathLgthDil * k$. Therefore,

$$pc_{start} - PathLgthDil * (k-1) < PathLgthDil.$$

Thus, (4.9) is proved.

Once the processor P receives a message about particle p, the processor sends self-messages about particle p till $\text{ls}(p,k)$, the current line segment being simulated no longer intersects Voronoi$(P,\mathcal{L})$. Let $\text{ls}(p,k_1)$ be the first line segment that does not intersect Voronoi$(P,\mathcal{L})$. Clearly

$$\begin{aligned} k_1 &> k \qquad (4.14)\\ &\geq k+1. \qquad (4.15)\end{aligned}$$

In pass $PathLgthDil * (k_1 - 1)$, the processor finds that it does not represent the particle p any more and hence stops representing the particle p at the end of the pass. Therefore, we have that

$$pc_{end} = PathLgthDil * (k_1 - 1) - 1. \tag{4.16}$$

From (4.15) and(4.16), we can deduce that

$$\begin{aligned} pc_{end} &= PathLgthDil * (k_1 - 1) - 1 \\ &\geq PathLgthDil * k - 1, \end{aligned}$$

thus proving (4.7).

We have defined k_1 to be the number of the first line segment that does not intersect Voronoi$(P, \mathcal{L})$. Suppose $\mathrm{ls}(p, k+1) \cap \mathrm{Voronoi}(P, \mathcal{L}) \neq \emptyset$. Then we have that $k_1 = k+1$. From (4.16), we have that

$$\begin{aligned} pc_{end} - (PathLgthDil * k - 1) \\ = (PathLgthDil * (k_1 - 1) - 1) - (PathLgthDil * k - 1) \\ = PathLgthDil * (k + 1 - 1) - PathLgthDil * k \\ = 0 < 1. \end{aligned}$$

Thus (4.10) is proved. □

Theorem 4.3.5 *Consider a particle* p*, processors* P_1 *and* P_2*, such that for some* k_1 *and* k_2*,*

- $\mathrm{Voronoi}(P_1, \mathcal{L}) \cap \mathrm{ls}(p, k_1) \neq \emptyset$*,*
- $\mathrm{Voronoi}(P_2, \mathcal{L}) \cap \mathrm{ls}(p, k_2) \neq \emptyset$*,*
- $P_1 \notin \mathrm{Furthermost}(p, k_1)$*, and*
- $P_2 \notin \mathrm{Furthermost}(p, k_2)$.

Let $[pc_{start1}, pc_{end1}]$ *and* $[pc_{start2}, pc_{end2}]$ *be the pass intervals in which processors* P_1 *and* P_2 *represent* p *in line segments* $\mathrm{ls}(p, k_1)$ *and* $\mathrm{ls}(p, k_2)$*, respectively.*

Then, we have that

$$(k_1 < k_2) \Longrightarrow (pc_{start1} < pc_{start2}). \tag{4.17}$$

Proof: From Theorem 4.3.2 and Theorem 4.3.3, we know that processor P_1 receives a message about particle p in some pass in the pass interval

$$[PathLgthDil * (k_1 - 1), PathLgthDil * k_1).$$

If the processor P_1 receives a message anytime before this (about particle p), the message is ignored as the processor checks and finds that it is not representing the particle.

Similarly, the processor P_2 starts simulating the particle in some pass in the pass interval

$$[PathLgthDil * (k_2 - 1), PathLgthDil * k_2).$$

We have that

$$pc_{start2} \geq PathLgthDil * (k_2 - 1) and \tag{4.18}$$

$$pc_{start1} < PathLgthDil * k_1. \tag{4.19}$$

Suppose $k_1 < k_2$. We have from (4.18) that

$$\begin{aligned} pc_{start2} &\geq PathLgthDil * (k_2 - 1) && (4.20) \\ &> PathLgthDil * (k_1 - 1) && (4.21) \\ &\geq PathLgthDil * k_1. && (4.22) \end{aligned}$$

From (4.19) and (4.22), we have that

$$pc_{start1} < pc_{start2},$$

which is what is to be proved. □

4.3.3 Optimal Value of M_{max}.

In this section, we will show that, for the algorithm given in Section 4.2 to execute without any overflow of messages, M_{max} should be assigned *at least* the value of $\min(N, N_{\text{VOR}} * (N_{nnp} + 1))$. We will do that by giving an example where overflow of messages occurs with $M_{max} < \min(N, N_{\text{VOR}} * (N_{nnp} + 1))$.

We will look at an example in an architecture based on the A_2 lattice. The example is shown in Figure 4.1.

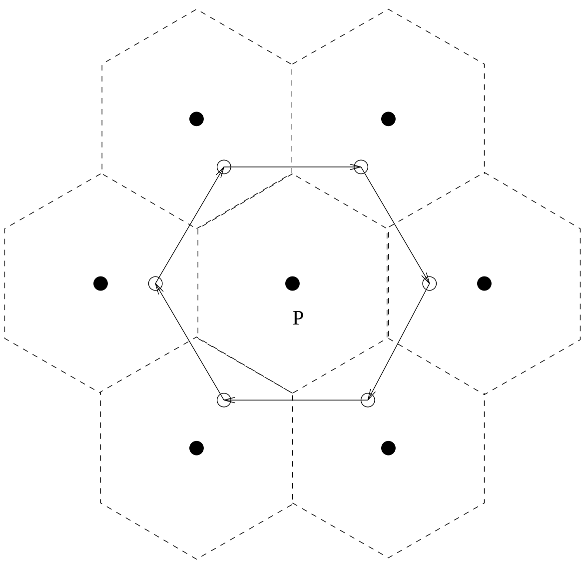

Figure 4.1: Example based on A_2.

Consider a processor P and the set, S_{nn}, of its nearest neighbor processors. We know that the cardinality of $S_{nn} = N_{nnp} = 6$.

For our example, initially, each of the processors in the set $S_{nn} \cup \{P\}$ simulates N_{VOR} particles. For each processor Q in S_{nn}, all of the N_{VOR} particles simulated by Q are at a single point in our example (indicated in Figure 4.1 by plain circles). In the next pass,

- the particles in processor P stay in P, and
- the particles in each Q in S_{nn} move to the processor adjacent to Q in the direction of the arrow indicated in Figure 4.1.

Now we can see that in the next pass, processor P receives N_{VOR} messages from each of its neighboring processors. The total number of messages received by processor P in the next pass is equal to $\min(N, N_{\text{VOR}} * (N_{nnp}+1))$. The maximum message storage capacity for processor P is just M_{max} ($< \min(N, N_{\text{VOR}} * (N_{nnp}+1))$). Therefore, overflow of messages occurs.

Chapter 5

The Next Extension

In this chapter we will present another algorithm, the *SE Algorithm*, which also simulates the motion of a set of particles along their respective curves. This algorithm can be executed correctly on a lattice computer only for simulations in which

$$\lceil N/2 \rceil < N_{\mathrm{vor}} * (N_{nnp} + 1). \tag{5.1}$$

Then, under certain reasonable assumptions (see (5.25) and (5.28)), the time taken by the SE algorithm to simulate a line segment of length s is less than the time taken by the FE algorithm to simulate the same line segment. The SE algorithm performs better using a tradeoff between executing faster and simulating the motion of particles accurately. The inaccuracy is that a processor executing the SE algorithm may simulate the motion of a particle which is not present in its Voronoi cell and may not simulate the motion of a particle which is present in its Voronoi cell. In Section 5.4.3 this inaccuracy is nicely limited. In this Chapter, we will present and analyze the code to be executed when (5.1) is satisfied.

The assumptions made regarding ρ_s and N_{crowd} in beginning of Chapter 3 also apply to this algorithm. We will prove the timing conditions given in Section 3.1 for this algorithm. The processors in the SE algorithm use what we call the *S-U tagged messages*, to exchange information about the particles that are being simulated. S stands for sure and U stands for unsure. Further explanation follows.

5.1 S-U Tagged Messages

We assume that the particles that are simulated by this algorithm are partitioned into two sets S_0 and S_1, such that

- $|S_0| = |S_1| = N/2$, if N is even, and

- $|S_0| = |S_1| + 1$ and $|S_0| = \lceil N/2 \rceil$, if N is odd.

using some method. We do not assume any specific partition of the particles. The SE algorithm will execute correctly irrespective of which set the individual particles belong to. This partition can even be assigned at execution time, but once assigned it cannot be changed during the simulation.

The messages passed between processors contain a tag associated with them in addition to containing information about a particle. A tag can be any value from the set $\{S, U\}$.

Consider a processor P that receives a message about a particle p with the tag value S in pass k. This means that, in pass $k-1$, the particle p was actually present in the Voronoi cell of the processor which sent the message about particle p.

Consider a processor P that receives a message about a particle p with the tag value U in pass k. This means that, in pass $k-1$, the particle p *may not* have been present in the Voronoi cell of the processor which sent the message about particle p.

5.2 The Basic Idea for the SE Algorithm

We will refer to the test done in Step 3 by a processor P in the FE algorithm as the *line segment test*. This test is applied to determine if a line segment to be simulated in the current pass intersects the Voronoi cell of the processor P.

Definition 5.2.1 *A processor P is said to* fully process *a message about a particle p in pass k, iff*

- *P receives a message about the particle p at the beginning of pass k, and*
- *P performs the line segment test on particle p in pass k.*

In FE algorithm, each processor P fully processes *all* messages received in the beginning of the pass. In the SE algorithm, each processor P fully processes only a subset of messages to see if they are about particles to be represented, thus saving the time to test the remaining messages. P blindly sends these (untested) messages to all the neighboring processors and to itself, but *after* marking them with the tag U. Therefore, any processor receiving a message with the tag U knows that the message corresponding to this particle was not fully processed in the previous pass and thus *should* be fully processed in the current pass. All messages about particles that pass the line segment check are given the tag S when these are sent out at the end of the pass.

More computations need to be done to ensure that in checking only a subset of messages, we do not lose information about the remaining particles. We need to make sure that no message is going to be ignored forever. In fact, in this algorithm we make sure that, if a message is not fully processed in the current pass, it will be fully processed in the very next pass. It is for this reason that we use the partitioning of the set of particles into two sets S_0 and S_1. We make sure that in the even passes, any message that is not fully processed is about a particle in the set S_1 and that in the odd passes, any message that is not fully processed is about a particle in the set S_0. This will ensure that a message about any particle is fully processed at least once in every two passes. This method will bring down the value of the maximum number of messages that need to be fully processed in a single pass (compared to the FE algorithm). It is enough if *only* messages about particles in set S_0 are fully processed in even passes and *only* messages about particles in set S_1 are fully processed in odd passes. This is exactly how the SE algorithm executes in each processor.

The salient points we have made so far are

- A message about a particle was not fully processed in the previous pass if the tag associated with the message is U. The message was fully processed in the previous pass if the tag associated with the message is S.
- We want to make sure that a message about a particle is fully processed at least once in any two consecutive passes, in other words a message about a particle cannot have the tag U in any two consecutive passes.
- We reduce the maximum number of messages a processor needs to check for representation in a single pass, by fully processing a subset of messages about particles depending on whether the pass is even or odd.
- We fully process messages about particles in the sequence S_0 in the even pass, and fully process messages about particles in the sequence S_1 in the odd pass.

Now, we will proceed to take a look at the SE algorithm in pseudo-code.

5.3 The SE Algorithm

5.3.1 Definitions

Let us define the following variables and constants before proceeding to look at this algorithm in pseudo-code in Section 5.3.2.

- *TimeDil* is a constant whose value is equal to $PathLgthDil + 1$. This is the number of passes for which the simulation of a single line segment of the curve is done. We increase the number of consecutive passes within which a single line segment is simulated over the FE algorithm. This increase is to make sure that every processor whose Voronoi cell intersects the line segment, of a particle p, simulates the particle p for at least one pass.

- k is a variable whose value is the index of the line segment of the particles currently being simulated by any processor. The first line segment has an index of zero.
- *SimClock* is the variable which contains the current clock time of the system. The system clock is the finest resolution, discrete level clock available with the system. This variable is assumed to be updated automatically by the system.
- *CompLen* is an algorithm dependent constant used in synchronizing the passes. The value of *CompLen* is equal to the maximum time (as measured by the system clock) any processor might take to execute a pass as specified by the SE algorithm.
- p is a variable ranging over the particles in the simulation.
- *Tag(p)* contains the tag associated with the message about the particle p, received in the current pass. If two messages with different tags about the particle p were to be received in the current pass, the value of Tag(p) would be S. However, it is easy to see from the pseudo code in Section 5.3.2 that this situation never occurs.
- *Info(p)* contains all the relevant information about particle p, that a processor executing the SE algorithm needs. Info(p) may be updated by the processors in the course of a simulation. Given a particle p, there is an unique Info(p) associated with the particle.
- *passcount* is a variable whose value is the number of passes executed by any processor. The variable has a value of 0 at the beginning of the simulation.
- *Repr* is a data structure local to each processor that contains information about the particles. This data structure is used to implement the concept of the processor *representing* a particle. *Repr* can be accessed only by the following operations:
  - AddToRepr(p, t) : Modify the data structure to add a particle p with the tag t, starting from the current pass.
  - DeleteElements($passcount$) : Modify the data structure to delete all elements that were present in the data structure in the previous pass, but were found not to be present in the Voronoi cell of the processor in the current pass.
  - IsRepresenting(p) : Returns 'true' if the particle p has been added to *Repr* using the AddToRepr(p, S) operation and has not been deleted since then by a DeleteElements($passcount$) operation.

- M_{max} is the constant which bounds the number of particles about which messages are received by a processor in any pass. We will show that M_{max} can be assigned any value greater than or equal to N.
- n_{recvd} is a variable whose value is the number of particles about which messages were received by any processor in the current pass.
- *storage* is a data structure representing a set. The components of the set are Info(p) and an associated tag. A maximum of M_{max} messages can be held in this output buffer.
- *outbuffer(D)* is the buffer for storing outgoing messages along direction D. The messages essentially consists of Info(p) and an associated tag. A maximum of M_{max} messages can be held in this output buffer.
- S_0 and S_1 are partitions of the set of particles such that
  - if N is even, $|S_0| = |S_1|$, and
  - if N is odd, $|S_0| = |S_1| + 1$.
- *Ignore-set* is a variable whose value can either be S_0 or S_1 depending on whether the current pass is odd or even, respectively.

5.3.2 Pseudo Code

The SE algorithm is started up by an external agent sending messages with the tag S, about each particle p to all the processors P whose Voronoi cell contains $\rho_s(p, 0)$.

1. Let $pd = \{ p \mid p$ is a particle such that a message about the particle p was received in the current pass$\}$;
2. Assign zero to n_{recvd}.
3. For each particle p in set pd do
   (a) Add Info(p) and Tag(p) to storage and increment n_{recvd}.
4. If *passcount* is even, assign S_1 to *Ignore-set.*
5. If *passcount* is odd, assign S_0 to *Ignore-set.*
6. For each particle p such that Info(p) is in *storage* and p is in *Ignore-set* do
   (a) AddToRepr(p,Tag(p)).
   (b) Delete Info(p) and Tag(p) from *storage.*
   (c) Decrement n_{recvd}.

(d) For each direction in D in $\mathcal{D} \cup \{\vec{0}\}$, add Info($p$) and tag U to outbuffer(D).

7. For each particle p such that Info(p) is in *storage* do
   (a) If $[\rho_s(p,k), \rho_s(p,k+1)) \cap \text{Voronoi}(P, \mathcal{L}) = \emptyset$,
      i. Delete Info(p) and Tag(p) from *storage*.
      ii. Decrement n_{recvd}.
8. For all particles p such that Info(p) is in *storage* do
   (a) AddToRepr(p,S).
   (b) Delete Info(p) and Tag(p) from *storage*.
   (c) Decrement n_{recvd}.
   (d) For each direction D in $\mathcal{D} \cup \{\vec{0}\}$, add Info($p$) and tag S to outbuffer(D).
9. Increment $passcount$.
10. Assign ($passcount$ DIV $TimeDil$) to k.
11. DeleteElements($passcount$).
12. Idle till $SimClock$ has the value $(CompLen * passcount) - 1$.
13. Simultaneously send all messages in outbuffer.

5.4 Analysis

In this section we will prove the correctness of the SE algorithm presented in Section 5.3.2. The layout of the correctness proofs is similar to that of the proofs presented in Chapter 4. Additionally, we will also prove a theorem which quantifies the inaccuracy in the tradeoff between the FE and the SE algorithm. In the beginning of every pass, the processors select a set of messages M from all the messages received in the beginning of the pass, such that no two messages in M are about the same particle. All references to messages in this analysis will be to members of the set M.

Definition 5.4.1 Overflow of messages *occurs in pass pc* ($\neq 0$) *iff some processor P receives more than M_{max} messages in the pass $pc - 1$.*

Definition 5.4.2 *A processor P is said to* represent *particle p in the pass interval $[a, b]$ iff, for every pass in the pass interval $[a, b]$, IsRepresenting(p) returns 'true' at the end of the pass.*

We will justify the increase in the number of passes required to simulate the motion along a line segment of length s. Suppose we simulate a single line segment in $PathLgthDil$ passes as in the FE algorithm. Consider a line segment $\mathrm{ls}(p, k)$ of some particle p which intersects the Voronoi cells of exactly *PathLgthDil* processors. Consider a processor P which intersects $\mathrm{ls}(p, k)$, such that the position of P in the sequence of processors that intersect $\mathrm{ls}(p, k)$ is $PathLgthDil - 1$[1] immediate connections away from any processor in $\mathrm{Head}(p, k)$. According to the SE algorithm, it is possible that the processor P receives a message about particle p, with the tag S, in pass $PathLgthDil * (k - 1) + PathLgthDil - 1$. According to the SE algorithm, the processor P does not check the message for representation, but sends a message about this particle (with the tag U), to all the nearest-neighbor processors and to itself. In the next pass, the processor P checks the particle for representation and finds that it is *not* representing the particle (as the pass number then is $PathLgthDil * k$). Therefore, the processor P never represents particle p (in line segment k) even though the Voronoi cell of the processor P intersects $\mathrm{ls}(p, k)$. Increasing the number of passes used to simulate a line segment by one makes sure that all the processors represent the particle p for at least one pass in this pass interval.

We will next prove that overflow of messages does not occur when the processors execute the code given in Section 5.3.2. Then, we will prove that the timing constraints (the rationale for which has been given in Section 3.1) of Section 3.1 hold. Finally we will quantify the inaccuracy in the tradeoff between the FE and the SE algorithm.

[1] Note that the behavior of any processor $PathLgthDil$ immediate connections away from any processor in $\mathrm{Head}(p, k)$ has been dealt with in Section 3.1.

5.4.1 Non-Overflow of Messages

Theorem 5.4.1 *If M_{max} is assigned the value of N, every processor executing the code in Section 5.3.2 receives less than or equal to M_{max} messages.*

Proof: The total number of particles being simulated is equal to N. Therefore, we have that no processor can receive more than $N = M_{max}$ distinct messages in any single pass. $\square$

Recall that for the SE algorithm, (5.1) is satisfied. Therefore, the value of M_{max} in the SE algorithm can be no worse than twice the value of M_{max} in the FE algorithm. This is because of the presence of 'unsure' messages. From Theorem 5.4.1, we have that there cannot be any overflow of messages in the course of the simulation in the SE algorithm.

5.4.2 Timing Constraints

The following two theorems will be made use of in proving the timing constraint conditions given in Section 3.1 (with $c = TimeDil$).

Theorem 5.4.2 *Consider a processor P such that for a particle p and some k,*

1. $\text{Voronoi}(P, \mathcal{L}) \cap \text{ls}(p, k) \neq \emptyset$,
2. $P \notin \text{Head}(p, k)$, *and*
3. $P \notin \text{Furthermost}(p, k)$.

If the processor $\text{Head}(p, k)$ *receives a message about the particle p in pass*

$$TimeDil * (k - 1),$$

then this processor P receives a message about the particle p in some pass in the pass interval

$$(TimeDil * (k - 1), TimeDil * k - 1).$$

Proof: Let $d = \min_{h \in \text{Head}(p,k)}(\text{PathLength}(h, P))$. Let $\text{PathLength}(H, P) = d$ and $H \in \text{Head}(p, k)$. As $P \notin \text{Furthermost}(p, k)$, we have $d \neq PathLgthDil$. The Voronoi cell of both processors P and H intersect $\text{ls}(p, k)$. Therefore,

$$0 < d < PathLgthDil. \tag{5.2}$$

Consider the pass $pc = TimeDil * (k - 1) + d$. We will show later in the proof that processor P receives a message about particle p in pass pc.

Adding $TimeDil * (k - 1)$ to each term in (5.2), we have

$$TimeDil * (k - 1) < pc < TimeDil * k - 1,$$

which proves that P receives a message about p in the pass interval

$$(TimeDil * (k-1), TimeDil * k - 1).$$

All that remains to prove is to show that the processor P receives a message about the particle p in pass $pc = TimeDil * (k-1) + d$. We will prove it by induction.

Base case: d = 0. This says that the processor P we are considering belongs to $\text{Head}(p,k)$. The above statement is vacuously true for any processor $\in \text{Head}(p,k)$.

Inductive step: Let the above statement be true for all d such that

$$0 \leq d < m < PathLgthDil.$$

We will prove that any processor P, such that

$$\text{ls}(p,k) \cap \text{Voronoi}(P, \mathcal{L}) \neq \emptyset \text{ } and$$
$$\min_{h \in \text{Head}(p,k)} (\text{PathLength}(h, P)) = m,$$

receives a message about particle p in pass $TimeDil * k + m$.

Let H be the processor such that

- $H \in \text{Head}(p,k)$, and
- $\text{PathLength}(H, P) = m$.

$(\text{PathLength}(H,P) = m) \Longrightarrow (\exists Q$ whose $\text{PathLength}(H,Q) = m-1$ and Q is a nearest-neighbor of P and $\text{ls}(p,k)$ intersects $\text{Voronoi}(Q, \mathcal{L}))$.

By the inductive assumption, we have that the processor Q receives a message in pass $m - 1 + TimeDil * (k-1)$ about the particle p. The processor checks to see if $\text{ls}(p,k)$ intersects $\text{Voronoi}(Q, \mathcal{L})$ and finds that it does At the end of the pass, processor Q sends a message about the particle p to all the nearest-neighbors of Q. Processor P, being a nearest-neighbor of Q, receives a message about the particle p in the next pass, m. □

Theorem 5.4.3 *Consider any processor P, some particle p and some k such that*

1. $\text{Voronoi}(P, \mathcal{L}) \cap \text{ls}(p,k) \neq \emptyset$.

2. $P \in \text{Head}(p,k)$.

Any such processor P receives a message about the particle p in pass

$$TimeDil * (k-1).$$

Proof: We will use induction on k to prove this theorem.

Base case: k=1. The SE algorithm is initialized by an external agent sending messages to all the processors whose Voronoi cells contain the starting point of $\mathrm{ls}(p,1)$. Therefore, all such processors receive a message about the particle p in pass 0 $(= TimeDil * (k-1))$.

Inductive step: Let us assume that the given is true for all k such that $1 \leq k < m$. We will prove the given statement for $k = m$.

Consider any processor P such that $P \in \mathrm{Head}(p,m)$. Consider the processor Q' such that

- $\mathrm{Voronoi}(Q', \mathcal{L}) \cap \mathrm{ls}(p, m-1) \neq \emptyset$.
- $\mathrm{MinMgsTim}(Q', P) = 1$.

There are two cases to be considered.

- **Case 1:** $Q \in \mathrm{Head}(p, m-1)$**.** By the inductive assumption we have that Q' receives a message about particle p in the pass $pc = TimeDil * (m-2)$. Therefore we have

$$TimeDil * (m-2) \leq pc < TimeDil * (m-1). \tag{5.3}$$

- **Case 2:** $Q' \notin \mathrm{Head}(p, m-1)$**.** We know that $Q' \notin \mathrm{Furthermost}(p,k)$. Therefore, From Theorem 5.4.2, we know that Q' receives a message about particle p in some pass pc such that

$$TimeDil * (m-2) < pc < TimeDil * (m-1) - 1. \tag{5.4}$$

From (5.3) and 5.4), we have that the processor Q' receives a message and starts representing the particle p in some pass pc such that

$$TimeDil * (m-2) \leq pc < TimeDil * (m-1) - 1.$$

The processor Q' executes the code in Section 5.3.2 and sends messages to itself and other processors till the pass number reaches $TimeDil * (m-1) - 1$. This shows that processor Q' sends a message about particle p to all its nearest-neighbor processors in pass $TimeDil * (m-1) - 1$. P, being a nearest-neighbor processor of Q', receives a message about particle p in the next pass, $TimeDil * (m-1)$, which is what is to be proved. □

Proof of Timing Constraints

The following two theorems prove that the SE Algorithm satisfies the timing constraints given in Section 5.4. This will conclude the proof of overall correctness of the SE algorithm.

Theorem 5.4.4 *Consider a processor P, a particle p and some k such that*

- $\mathrm{Voronoi}(P, \mathcal{L}) \cap \mathrm{ls}(p, k) \neq \emptyset$, *and*
- $P \notin \mathrm{Furthermost}(p, k)$.

Let $[pc_{start}, pc_{end}]$ be the maximal pass interval in which P represents p in the line segment $\mathrm{ls}(p, k)$. Then

$$[TimeDil * (k-1), TimeDil * k) \cap [pc_{start}, pc_{end}] \neq \emptyset, \quad (5.5)$$

$$\exists \epsilon \text{ (an integer } > 0) \text{ such that } 0 \leq pc_{start} - TimeDil * (k-1) \leq \epsilon, \quad (5.6)$$

$$pc_{end} \geq TimeDil * k - 1 \text{ and} \quad (5.7)$$

$$\text{if } \mathrm{ls}(p, k+1) \cap \mathrm{Voronoi}(P, \mathcal{L}) \neq \emptyset,$$

$$\exists \delta > 0 \text{ such that } pc_{end} - (TimeDil * k - 1) < \delta. \quad (5.8)$$

Furthermore, we may choose $\epsilon = TimeDil$ and $\delta = 2$ in (5.6) and (5.8), respectively.

Proof: In this theorem we will prove stronger versions of (5.6) and (5.8). Specifically, we will show later in the proof the following two inequalities (which automatically prove (5.6), with $\epsilon = TimeDil$ and (5.8), with $\delta = 2$).

$$0 \leq pc_{start} - TimeDil * k \leq TimeDil \text{ and} \quad (5.9)$$

$$\text{if } \mathrm{ls}(p, k+1) \cap \mathrm{Voronoi}(P, \mathcal{L}) \neq \emptyset,$$

$$pc_{end} - (TimeDil * k - 1) < 2. \quad (5.10)$$

From Theorem 5.4.2 and Theorem 5.4.3, we know that processor P receives a message about particle p in some pass pc such that

$$TimeDil * (k-1) \leq pc < TimeDil * k - 1. \quad (5.11)$$

The processor P checks to find that the particle needs to be represented and represents the particle at least from pass $pc + 1$.

Therefore, we have

$$pc_{start} \leq pc + 1. \quad (5.12)$$

We know that a processor cannot start representing a particle p in line segment k before the pass $TimeDil * (k-1)$. Therefore, we have

$$TimeDil * (k-1) \leq pc_{start}. \quad (5.13)$$

From (5.13) and (5.11), we have that

$$[TimeDil * (k-1), TimeDil * k) \cap [pc_{start}, pc_{end}] \neq \emptyset.$$

Thus (5.5) is proved. From (5.11), we know that $pc_{start} < TimeDil * k$. Therefore,

$$pc_{start} - TimeDil * (k-1) < TimeDil.$$

Thus, (5.9) is proved.

Once the processor P receives a message about particle p, the processor sends self-messages about particle p till $\mathrm{ls}(p,k)$, the current line segment being simulated no longer intersects $\mathrm{Voronoi}(P,\mathcal{L})$. Let $\mathrm{ls}(p,k_1)$ be the first line segment that does not intersect $\mathrm{Voronoi}(P,\mathcal{L})$.

Clearly

$$k_1 > k \tag{5.14}$$

$$\geq k+1. \tag{5.15}$$

The processor finds that it does not represent the particle p any more and hence stops representing the particle at the end of the pass, in at least pass $TimeDil * (k_1 - 1) + 1$. Therefore, we have that

$$pc_{end} = TimeDil * (k_1 - 1). \tag{5.16}$$

From (5.15) and(5.16), we can deduce that

$$\begin{aligned} pc_{end} &= TimeDil * (k_1 - 1) \\ &\geq TimeDil * k - 1, \end{aligned}$$

thus proving (5.7).

We have defined k_1 to be the number of the first line segment that does not intersect $\mathrm{Voronoi}(P,\mathcal{L})$. Suppose $\mathrm{ls}(p,k+1) \cap \mathrm{Voronoi}(P,\mathcal{L}) \neq \emptyset$. Then we have that $k_1 = k+1$. From (5.16), we have that

$$\begin{aligned} &pc_{end} - (TimeDil * k - 1) \\ &= TimeDil * (k_1 - 1) - (TimeDil * k - 1) \\ &= TimeDil * (k+1-1) - TimeDil * k + 1 \\ &= 1 < 2. \end{aligned}$$

Thus (5.10) is proved. □

Theorem 5.4.5 *Consider a particle p, processors P_1 and P_2, such that for some k_1 and k_2,*

- $\mathrm{Voronoi}(P_1,\mathcal{L}) \cap \mathrm{ls}(p,k_1) \neq \emptyset$,
- $\mathrm{Voronoi}(P_2,\mathcal{L}) \cap \mathrm{ls}(p,k_2) \neq \emptyset$,

- *$P_1 \notin \mathrm{Furthermost}(p, k_1)$, and*
- *$P_2 \notin \mathrm{Furthermost}(p, k_2)$.*

Let $[pc_{start1}, pc_{end1}]$ and $[pc_{start2}, pc_{end2}]$ be the maximal pass intervals in which processors P_1 and P_2 represent p in line segments $\mathrm{ls}(p, k_1)$ and $\mathrm{ls}(p, k_2)$, respectively.

Then, we have that

$$(k_1 < k_2) \Longrightarrow (pc_{start1} < pc_{start2}). \tag{5.17}$$

Proof: From Theorem 5.4.2 and Theorem 5.4.3, we know that processor P_1 receives a message about particle p in some pass in the pass interval

$$[TimeDil * (k_1 - 1), TimeDil * k_1).$$

If the processor P_1 receives a message anytime before this (about particle p), the message is ignored as the processor checks and finds that it is not representing the particle.

Similarly, the processor P_2 starts simulating the particle in some pass in the pass interval $[TimeDil * (k_2 - 1), TimeDil * k_2)$. We have that

$$pc_{start2} \geq TimeDil * (k_2 - 1) and \tag{5.18}$$
$$pc_{start1} < TimeDil * k_1. \tag{5.19}$$

Suppose $k_1 < k_2$. We have from (5.18) that

$$pc_{start2} \geq TimeDil * (k_2 - 1) \tag{5.20}$$
$$> TimeDil * (k_1 - 1) \tag{5.21}$$
$$\geq TimeDil * k_1. \tag{5.22}$$

From (5.19) and (5.22), we have that

$$pc_{start1} < pc_{start2},$$

which is what is to be proved. □

5.4.3 Simulation Accuracy of the SE Algorithm

In this section, we will quantify the inaccuracy in the tradeoff between the FE and the SE algorithm. Specifically, we will show that there for each line segment $\mathrm{ls}(p, k)$, there is at most one pass in which either

- p is present in the Voronoi cell of a processor P, but is not represented by P, or

- p is not present in the Voronoi cell of a processor P, but is represented by P.

In the SE algorithm, we note that a processor P whose Voronoi cell intersects a line segment $\text{ls}(p, k)$, receives a message about particle p exactly

$$pc = \min_{h \in \text{Head}(p,k)} PathLength(P, h)$$

passes after a processor in $\text{Head}(p, k)$ has received a message about p (as in the FE algorithm). Such a processor P can receive a message with the tag S, pc passes after a processor in $\text{Head}(p, k)$. In this case, according to the SE algorithm, the processor P ignores the message in this pass. This is a pass in which a particle present in the Voronoi cell of P is not being represented by P. We also note that the number of such passes is at most one because in the very next pass, P receives a message with the tag U about the particle p and starts representing p.

Consider a processor P that intersects a line segment $\text{ls}(p, k)$ such that P does not intersect $\text{ls}(p, k + 1)$. Let $[pc_{start}, pc_{end}]$ be the maximal pass interval in which P represents the particle p in $\text{ls}(p, k)$. From Theorem 5.4.4, we have that

$$pc_{end} - (TimeDil * k - 1) < 2.$$

In other words, we have that

$$pc_{end} - (TimeDil * k - 1) \leq 1.$$

pc_{end} is a pass in which a particle p that is not present in the Voronoi cell of P is represented by P. We also note that the number of such passes is at most one (follows from the above inequality).

5.4.4 Performance of SE Algorithm

In this section we will show that the SE algorithm performs better than the FE algorithm under certain reasonable assumptions, given that (5.1) is satisfied. We note that the FE algorithm and the SE algorithm have similar structures, i.e., they eliminate spurious messages and send out messages about the particles actually being represented. First, we show that, under certain reasonable conditions, a single pass in the SE algorithm completes in less time than a single pass in the FE algorithm.

Then, we will show that, under certain reasonable assumptions, the total time for simulation of a single line segment in the SE algorithm is smaller than the time for simulation of a single line segment in the FE algorithm.

For comparing the performance of the algorithms, we will ignore all the steps that are identical in the algorithms. The steps that are non-identical can be considered to be of two types. The first type are the operations that do not involve a for-loop. These are assumed to be unit time operations and thus do not significantly affect the overall execution time of a pass. The steps that involve the for-loops in both the algorithms account for a large portion of the execution time. We note that the execution time of any pass is delayed by the maximum time any pass might take to execute (in the respective algorithms). Therefore, the execution time of a pass is the same as the maximum execution time of any pass. With all this in mind, we will proceed to compute the approximate maximum execution times of a pass in both the algorithms. We point out to the reader that Steps 2, 3 and 4 in the FE algorithm correspond to Steps 3, 7 and 8 in the SE algorithm, respectively.

Consider the FE algorithm. The maximum execution time possible for a single pass will be reached when a processor receives N messages and actually represents N_{VOr} particles. Suppose a single instance of the for-loop in Step 2 takes t_0 time to execute. The total amount of time the for-loop takes to execute is $N * t_0$. Similarly suppose that a single instance of the for-loops in Steps 3 and 4 take times t_1 and t_2, respectively, to execute. We have that execution time for Step 3 is $N * t_1$ and execution time for Step 4 is $N_{\mathrm{VOr}} * t_2$. We note that checking of the condition in Step 3 takes more time than the subsequent operations that are performed (when the condition is found to be true).

Therefore, the total execution time (in the worst case) of a single pass is

$$N * (t_0 + t_1) + N_{\mathrm{VOr}} * t_2. \tag{5.23}$$

We will compute the worst case execution time for a pass in the SE algorithm in a similar way. The maximum execution time possible for a single pass is reached when a processor P receives N messages and P actually represents N_{VOr} particles. As before, we note that the for-loops dominate the computation times. The execution time for Step 3 is $N * t_0$, as a single instance of the for-loop in this step takes the same time to execute as a single instance of the for-loop in Step 2 in the FE algorithm.

The execution time for Step 6 is $\lceil N/2 \rceil * t_2$ because

- *Ignore-seq* contains a maximum of $\lceil N/2 \rceil$ particles and the number of times the for-loop executes is bound by the size of *Ignore-seq*, and
- a single instance of this for-loop takes the same amount of time to execute as a single instance of the for-loop in Step 4 of the FE algorithm.

The execution time for Step 7 is $\lceil N/2 \rceil * t_1$ because

- *storage* contains a maximum of $\lceil N/2 \rceil$ particles and the number of times the for-loop executes is bound by the number of particles present in the *storage*, and
- a single instance of this for-loop takes the same amount of time to execute as a single instance of the for-loop in Step 3 of the FE algorithm.

The execution time for Step 8 is $N_{\text{vor}} * t_2$ because

- a maximum of N_{vor} particles can actually be present in the Voronoi cell of the processor, and
- a single instance of this for-loop takes the same amount of time to execute as a single instance of the for-loop in Step 4 of the FE algorithm.

Therefore, the total execution time for a single pass in the SE algorithm is

$$N * t_0 + (\lceil N/2 \rceil + N_{\text{vor}}) * t_2 + t_1 * \lceil N/2 \rceil. \tag{5.24}$$

Assuming that

$$t_1 > t_2, \tag{5.25}$$

we have

$$\begin{aligned} &(5.23) - (5.24) \\ = &\lfloor N/2 \rfloor * (t_1 - t_2) \\ &> 0. \end{aligned}$$

Therefore, we have that a pass in the SE algorithm takes less time to execute than a pass in the FE algorithm. Assumption that (5.25) is true is reasonable as the operations of storing and sending out messages can be implemented very efficiently (one might even skip the idea of having an intermediate storage) whereas the inherent complexity of the computation in the line segment check makes it impossible to reduce its execution time.

The time taken by the algorithms to simulate a single line segment is the number of passes multiplied by the execution time for a single pass. For ease of presentation, we will abbreviate $PathLgthDil$ by P. Therefore, we have that the time taken by the FE algorithm to simulate a single line segment is

$$t_0 * (N * P) + t_1 * (N * P) + t_2 * (N_{\text{vor}} * P). \tag{5.26}$$

The time taken by the SE algorithm to simulate a single line segment is

$$\begin{aligned} &t_0 * (N * P + N) + t_1 * \lceil N/2 \rceil * (P + 1) + \\ &t_2 * (\lceil N/2 \rceil * (P + 1) + N_{\text{vor}} * P + N_{\text{vor}}). \end{aligned} \tag{5.27}$$

Assuming

$$t_1 > \left(\frac{N}{\lfloor N/2 \rfloor * P - \lceil N/2 \rceil}\right) * t_0 + \left(\frac{\lceil N/2 \rceil * (P + 1) + N_{\text{vor}}}{\lfloor N/2 \rfloor * P - \lceil N/2 \rceil}\right) * t_2, \tag{5.28}$$

we have that

$$(5.26) - (5.27) > 0,$$

Dimension	2	3	4	5	6	7
coefficient of t_0	2	3	1	1	2/3	7/11
coefficient of t_2	4	11	3	4	8/3	31/11

Table 5.1: Values of coefficients for A_n lattices.

Dimension	2	3	4	5	6	7
coefficient of t_0	1	3/2	1/2	5/9	1/3	7/20
coefficient of t_2	3	13/2	5/2	26/9	7/3	43/20

Table 5.2: Values of coefficients for Z^n lattices.

and therefore, the SE algorithm takes less time to simulate a single line segment than the FE algorithm.

Tables 5.1 and 5.2 give the values of the coefficients of t_0 and t_2 in (5.28) for typical lattices A_n and Z^n respectively. We note that $N_{\text{VOT}} \leq N$.

Assumption that (5.28) is true is reasonable because for the lattices A_n and Z^n of dimensions greater than 1, one can see from extrapolating Tables 5.1 and 5.2 that (5.28) can be proved from

$$t_1 > 3 * t_0 + 11 * t_2,$$

and this is reasonable because of the fact that the operations which take time t_0 and t_2 are operations which involve storing and sending out messages and can be implemented very efficiently, whereas the inherent complexity of the step that takes time t_1 makes it impossible to reduce the execution time.

Chapter 6

Summary and Future Work

In this paper we took a look at a *literal/analogical* approach to handle simulation of physical events. We summarized the representation of the euclidean space and the algorithm for simulation of motion of a single particle from [She91]. The algorithm for the single particle case as well as the ones presented in this paper executed in synchronized passes. A pass consisted of receiving messages, performing local computations and sending out messages at the end of the pass. An algorithm is described by the code to be executed for a single pass.

Initially, we considered extending the single particle algorithm to handle multi-particles by having it execute a pass in the single particle algorithm on all the messages received in the pass. In the worst case, for this possibility, we found that a processor can receive a maximum of N messages, where N is the total number of particles in the simulation and so considered the further possibility of executing a single pass in the single particle algorithm N times in a pass for this multi-particle algorithm. We found that this method worked but was awfully slow. Therefore, we looked at a different algorithm, called the FE algorithm in which a pass in the single particle algorithm was executed for only $\min(N, N_{\mathrm{VOT}} * (N_{nnp} + 1))$ times, where N_{VOT} is the maximum number of particles that can be present in the Voronoi cell of a processor at any point of time and N_{nnp} is the number of nearest-neighbor processors of any processor in the lattice. We note that this is a considerable improvement over the method in which we execute a single pass in the single particle algorithm N times. We also showed that we cannot reduce the number of times a pass in the single particle algorithm needs to be executed to less than $\min(N, N_{\mathrm{VOT}} * (N_{nnp} + 1))$.

We presented the SE algorithm as an alternative algorithm that should be applied only when $N < 2 * N_{\mathrm{VOT}} * (N_{nnp} + 1)$. The SE algorithm resulted from an attempt at reducing further the execution time to simulate motion of multi-particles by allowing some additional approximations in the simulation. Specifically, in the SE algorithm, the simulation can lag behind the physical event by a maximum of one pass. In this algorithm, we executed a single pass in the single particle algorithm on a *subset* of messages received in a pass, thus cutting down the execution time for a single pass. We used the concept of S-U tags, where a tag is associated with every message. We compared the performance of the SE algorithm to the FE algorithm and confirmed, under certain reasonable hypotheses, the lower execution time of a pass in the SE algorithm.

As we mentioned before, this paper is just a small step towards achieving the goal of simulating any physical event. More work needs to be done before we can make it a reality. Future work that needs to be done includes

1. developing algorithms to simulate motion at varying speeds,
2. extending the algorithms developed so far to arbitrary curves (From [She91, CRS91a] it is clear how to apply the algorithm in this paper to straight line and circular motion),
3. developing algorithms to simulate motion of particles, taking into consideration, the effect of collisions between particles.
4. extending the algorithms to simulate the motion, elasticity, ... of multi-particle objects,
5. integrating the above extensions/new algorithms into a single system capable of doing all of the above,
6. extending the single wave motion simulation algorithm in [She91, CRS91b] to handle multiple, non-interacting waves using the approach followed in this paper, and
7. extending the proposed architecture [CRS90b] and programming language interface in which simulations are done.

Chapter 7

Illustration of Execution of the SE Algorithm

Consider a particle p, a positive integer k and a processor P. The line segment $\mathrm{ls}(p,k)$ can have different orientations with respect to the Voronoi cell of P. We will present, in the form of tables, the behavior of the SE algorithm in all these different orientations. We divide space into three regions with respect to P, namely

1. $\mathrm{out}(P)$, which is the set of points not present in the Voronoi cell of P,
2. $\mathrm{boundary}(P)$ which is the set of points present in the boundary of the Voronoi cell of P, and
3. $\mathrm{in}(P)$, which is the set of points present inside the Voronoi cell of P, but not in the boundary of the Voronoi cell of P.

For simplifying the analysis, the line segment $\mathrm{ls}(p,k)$ is divided into three parts namely

1. $\mathrm{lt}(p,k)$, the left end point of the line segment $\mathrm{ls}(p,k)$,
2. $\mathrm{rt}(p,k)$, the right end point of the line segment $\mathrm{ls}(p,k)$, and
3. $\mathrm{ols}(p,k)$, the line segment $\mathrm{ls}(p,k)$ excluding the points $\mathrm{lt}(p,k)$ and $\mathrm{rt}(p,k)$.

We define that a division, DLS, of the line segment *is present in* a division, DOS, of space iff

- $\mathrm{DLS} = \mathrm{lt}(p,k)$, $\mathrm{DOS} \in \{\mathrm{out}(P), \mathrm{in}(P), \mathrm{boundary}(P)\}$ and $\mathrm{DLS} \in \mathrm{DOS}$;
- $\mathrm{DLS} = \mathrm{rt}(p,k)$, $\mathrm{DOS} \in \{\mathrm{out}(P), \mathrm{in}(P), \mathrm{boundary}(P)\}$ and $\mathrm{DLS} \in \mathrm{DOS}$;

- DLS $=$ ols(p,k), DOS $\in \{\text{out}(P), \text{boundary}(P)\}$ and DLS $\subset$ DOS; or
- DLS $=$ ols(p,k), DOS $=$ in(P) and DLS $\cap$ DOS $\neq \emptyset$.

The different orientations that we talk about are the result of different combinations (as in the above definition) of the division of line segment and the division of space. For example, one possible orientation is

1. lt(p,k) is present in boundary(P),
2. rt(p,k) is present in out(P), and
3. ols(p,k) is present in out(P).

Consider the maximal, possibly empty, pass interval $[pc_{start}, pc_{end}]$ in which a processor P represents the line segment ls(p,k). The values of pc_{start} and pc_{end} depend on various factors (like $PathLgthDil$) for any given orientation. We recall that the simulation in the SE algorithm can lag behind reality by at most one pass. Assuming this worst case lag of one pass, we will present, in Tables 7.1 and 7.2, the values of pc_{start} ($\geq TimeDil * (k-1)$) and pc_{end} ($\leq TimeDil * k - 1$), respectively, for the various physically possible orientations of ls(p,k) with respect to P.

For ease of presentation, we will abbreviate $TimeDil$ by TD and assume that

$$\min_{h \in \text{Head}(p,k)} (\text{PathLength}(h,P)) = i.$$

The letter B in a column means that the division of ls(p,k) corresponding to that column is present in boundary(P). Similarly, the letters O and I in columns mean that the divisions of ls(p,k) corresponding to those columns are present in out(P) and in(P), respectively.

lt(p,k)	ols(p,k)	rt(p,k)	Additional condition	Start pass
B	B	B	$-$	$TD*(k-1)+1$
B	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})=\emptyset$	$TD*(k-1)+1$
B	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})\neq\emptyset$	$TD*(k-1)$
B	I	B	$-$	$TD*(k-1)+1$
B	I	O	$-$	$TD*(k-1)+1$
B	I	I	$-$	$TD*(k-1)+1$
O	O	B	$-$	$-$
O	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})\neq\emptyset$	$TD*(k-1)$
O	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})=\emptyset$	$-$
O	I	B	$i\neq PathLgthDil$	$TD*(k-1)+1+i$
O	I	B	$i=PathLgthDil$	$-$
O	I	O	$-$	$TD*(k-1)+1+i$
O	I	I	$i\neq PathLgthDil$	$TD*(k-1)+1+i$
O	I	I	$i=PathLgthDil$	$-$
I	I	B	$-$	$TD*(k-1)$
I	I	O	$-$	$TD*(k-1)$
I	I	I	$-$	$TD*(k-1)$

Table 7.1: Illustrating execution of the SE algorithm: start pass

lt(p,k)	ols(p,k)	rt(p,k)	Additional condition	End pass
B	B	B	$-$	$TD*k-1$
B	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})=\emptyset$	$TD*k-1$
B	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})\neq\emptyset$	$TD*k-1$
B	I	B	$-$	$TD*k-1$
B	I	O	$-$	$TD*k-1$
B	I	I	$-$	$TD*k-1$
O	O	B	$-$	$-$
O	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})\neq\emptyset$	$TD*(k-1)$
O	O	O	$\text{ls}(p,k-1)\cap\text{Voronoi}(P,\mathcal{L})=\emptyset$	$-$
O	I	B	$i\neq PathLgthDil$	$TD*k-1$
O	I	B	$i=PathLgthDil$	$-$
O	I	O	$-$	$TD*k-1$
O	I	I	$i\neq PathLgthDil$	$TD*k-1$
O	I	I	$i=PathLgthDil$	$-$
I	I	B	$-$	$TD*k-1$
I	I	O	$-$	$TD*k-1$
I	I	I	$-$	$TD*k-1$

Table 7.2: Illustrating execution of the SE algorithm: end pass

Bibliography

[AG89] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., 1989.

[Aur90] F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. Technical Report B-90-9, Institute for Computer Science, Dept. of Mathematics, Freie Universität Berlin, November 1990.

[CN93] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups.* Springer Verlag, second edition, 1993.

[CRS90a] J. Case, D. S. Rajan, and A. M. Shende. Optimally representing euclidean space discretely for analogically simulating physical phenomena. In K. V. Nori, editor, *Foundations of Software Technology & Theoretical Computer Science*, volume 472 of *Lecture Notes in Computer Science*, pages 190–203. Springer Verlag, Berlin, 1990. See [CRS91c].

[CRS90b] J. Case, D. S. Rajan, and A. M. Shende. Parallel processor computer, Dec 1990. U.S. Patent Application submitted.

[CRS91a] J. Case, D. Rajan, and A. Shende. Simulating uniform motion in lattice computers I: Constant speed particle translation. Technical Report 91-17, University of Delaware, June 1991.

[CRS91b] J. Case, D. Rajan, and A. Shende. Spherical wave front generation in lattice computers. Technical Report 91-16, University of Delaware, June 1991.

[CRS91c] J. Case, D. S. Rajan, and A. M. Shende. Lattice computers for optimally representing euclidean space. Technical Report 91-15, University of Delaware, June 1991. Expands on and corrects slightly [CRS90a].

[CRS92] J. Case, D. Rajan, and A. Shende. Representing the spatial/kinematic domain and lattice computers. In K. Jantke, editor, *Proceedings of the Third International Workshop on Analogical and Inductive Inference*, volume 642 of *Lecture Notes in Artificial Intelligence*, pages 1–25. Springer-Verlag, Berlin, Dagstuhl Castle, Germany, October 1992.

[GL87] P. M. Gruber and G. Lekkerkerker. *Geometry of Numbers.* North-Holland Mathematical Library, 1987.

[Her64] I. N. Herstein. *Topics in Algebra.* Blaisdell Publishing Co., 1964.

[Qui87] Michael J. Quinn. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill Book Company, 1987.

[RS91] D. Rajan and A. Shende. Efficiently generated lattices and root lattices. Technical Report 92-02, University of Delaware, August 1991. Journal version submitted.

[She91] A. M. Shende. *Digital Analog Simulation of Uniform Motion in Representations of Physical n-Space by Lattice-Work MIMD Computer Architectures.* Ph. D. Dissertation, SUNY at Buffalo, Buffalo, N.Y., 1991. TR 91-14, Department of Computer Science.