Simulating Uniform Motion in Lattice Computers I: Constant Speed Particle Translation

John Case and Dayanand S. Rajan*
Computer Science Department
103 Smith Hall
University of Delaware
Newark, DE 19716

Anil M. Shende
Department of Mathematics
Haverford College
Haverford, PA 19041

CR Classification Numbers: I.3.5;G.1.0;C.1.2;F.1.1;F.2.2;I.6.1;C.2;J.2;J.7

Abstract

Euclidean n-space can be discretely represented as an n-dimensional lattice, and a mesh computer can be naturally associated with each (possibly finite) subset of such a lattice, where the processors are at the lattice points in the subset and the mesh's interconnections correspond to minimal length vectors between the lattice points associated with processors. An algorithm template is presented for *analogical* simulation, in such mesh computers, of constant speed, particle motion in euclidean n-space, *and* a particular instance of the template is refined for constant speed straight-line particle translation. Discussed are the possibilties of using this algorithm for simulating constant speed solid body motion and the possible utility of this algorithm for circumventing, in a new way, the difficulties, pointed out by Frisch and Hasslacher, in simulating 3-dimensional fluid flow in lattice gas cellular automata.

*Research supported in part by NSF Grant CCR 8713846 at the University of Delaware.
The email address for communication regarding this paper is 'case@cis.udel.edu'.

1 Introduction

In [CRS90a, CRS91a] euclidean n-space is *discretely* represented as an n-dimensional lattice[1] $\mathcal{K}$ and a *bounded* region R of euclidean n-space is represented by $(R \cap \mathcal{K})$, the corresponding *finite* subset of the lattice. In [CRS90a] the *mesh computer* (or simply *mesh*) $\mathcal{M}_{\mathcal{K}}(K)$ associated with subsets K of $\mathcal{K}$ is defined as having its synchronized, identical processors centered on the lattice points of K and with two way, immediate, identical communication links placed between a pair of these processors iff the lattice points corresponding to the pair of processors are at the $\mathcal{K}$-minimal *euclidean* distance apart.[2]

We are interested in analogically simulating, in such mesh computers $\mathcal{M}_{\mathcal{K}}(K)$, the *physical* motion of particles in euclidean n-space. A crucial eventual aim of our approach (see also [CRS90b, She91]) is to have the *simulation* of particle motion be carried out (approximately) linear in *real time*. We have in mind that the traversal of a simulated particle in a mesh be quite *analogous* to the traversal of the corresponding actual particle in euclidean n-space.

In *this* paper we present an algorithm template, CURVE (see Section 6), for carrying out simulations of the important special case of *constant speed* single particle motion. CURVE works in a large class of meshes based on the n-dimensional lattices called *well-represented* (Definition 5 in Section 3). Then (Section 7) we refine CURVE to obtain an example algorithm, TRANSLATE, for the special case of constant speed, (straight-line) translation of single particles. TRANSLATE can, then, be an integral part of an algorithm for producing constant speed, straight-line motion of solid objects (which consist of an aggregate of single particles). As will be clear, our algorithms necessarily carefully control the *timing* of the message sending that simulates particle motion.

While the underpinning we employ herein bears a resemblance to that of cellular automata approaches [Fey82, Min82, FT82, Tof84, TM87, Mar84, SW86, Mac86, Tof77b, Tof77a, Vic84, Wol83, Svo86, FHP86, Has87] our philosophy of algorithmic locality is not subject to the same apparent computational limitations. For example, although 2-dimensional *analogical* fluid flow simulations (based on the lattice A_2) can be carried out quite successfully with the cellular automata approach [SW86, Mac86], there are problems (for example using the lattice A_3) in the 3-dimensional case [FHP86, Has87]. We believe our algorithm TRANSLATE of Section 7 can serve as another basis for circumventing these problems. circumventing these problems. [Has87] notes that projecting from a fourth dimension (with two velocities) helps for *some* cases of fluid flow in the cellular automata approach. We expect our approach will be more general purpose for tackling a *variety* of cases computationally. Essentially TRANSLATE enables us to send particles in *any mesh-representable direction* at (approximately) constant speed. In A_3, while there are 12 directions available locally [CS88, FHP86, Has87], many more directions are representable in a mesh based on A_3. The limitations of the

[1] $\mathbf{R}^m$ is (by definition) the set of m-tuples of reals. An *n-dimensional lattice* is (by definition [CS88, GL87]) a set of points (vectors) $\mathcal{K}$ from $\mathbf{R}^{n'}$, for some $n' \geq n$, such that (i) $\mathcal{K}$ contains at most and at least n points which are linearly independent over the real numbers [Her64], (ii) $\mathcal{K}$ is closed under both vector addition of its points and scalar multiplication [Her64] *by integers*, and (iii) the set of sums of the squares of the coordinates of the non-zero points of $\mathcal{K}$ is bounded below by a non-zero constant. The origin of $\mathbf{R}^{n'}$, i.e., the n'-tuple of zeroes, is contained in $\mathcal{K}$, and will be denoted by $\vec{0}$. The minimal length of vectors in $\mathcal{K}$ is denoted MinLgth($\mathcal{K}$). Examples of n-dimensional lattices include $\mathbf{Z}^n \stackrel{\text{def}}{=} \{(k_1, \ldots, k_n) \mid k_1, \ldots, k_n \text{ are integers}\}$ and $A_n \stackrel{\text{def}}{=} \{(k_1, \ldots, k_{n+1}) \mid k_1, \ldots, k_{n+1} \text{ are integers summing to zero}\}$.

[2] Especial interest is attached to the cases where K is finite since, then, it may be possible to construct the computer $\mathcal{M}_{\mathcal{K}}(K)$ in the real world.

cellular automata approach pointed out in [FHP86, Has87] are fundamentally tied to the geometric shape and symmetry of the *local* connectedness (e.g., the 12-connectedness of A_3). With TRANSLATE we can, in effect, add more virtual directions to try to circumvent these limitations in new ways and add generality. Much remains to be tested. Our approach would, then, likely involve feasible increases in the resource of communication bandwidth between near neighbors to handle extra directions.

2 Lattices and Voronoi Cells

In this section we introduce a few concepts and results of use in our algorithms.

Definition 1 *Suppose $\mathcal{K}$ is an n-dimensional lattice, and K is a subset of $\mathcal{K}$. Suppose P and O are distinct points in $\mathcal{K}$.*

1. *The set of* basic directions in $\mathcal{K}$, *denoted* $\mathrm{Dirs}(\mathcal{K})$, *is the set of minimal length vectors in $\mathcal{K}$.*
2. *The* nearest neighbors of P, *denoted* $\mathrm{Nbrs}_{\mathcal{K}}(P)$, *is the set of points Q, in $\mathcal{K}$, so that $(Q-P) \in \mathrm{Dirs}(\mathcal{K})$.*
3. *The* Graphical Path Length of P with respect to O, *denoted* $\mathrm{GraphPL}_{\mathcal{K}}(P,O)$, *is defined as:*

$$\mathrm{GraphPL}_{\mathcal{K}}(P,O) = \begin{cases} 0, & \text{if } P = O; \\ 1 + \min\{\mathrm{GraphPL}_{\mathcal{K}}(Q,O) | Q \in \mathrm{Nbrs}_{\mathcal{K}}(P)\}, & \text{otherwise.} \end{cases}$$

4. *N is an* Inner Neighbor of P with respect to O in $\mathcal{K}$, *iff* $N \in \mathrm{Nbrs}_{\mathcal{K}}(P)$, and

$$\mathrm{GraphPL}_{\mathcal{K}}(N,O) = \mathrm{GraphPL}_{\mathcal{K}}(P,O) - 1.$$

   $Inner_{\mathcal{K}}(P,O)$ is the set of all points N, such that N is an inner neighbor of P with respect to O in $\mathcal{K}$.

5. *A sequence of points $(P =)P_0, P_1, \ldots P_m(= Q)$, $P_i \in \mathcal{K}, 0 \leq i \leq m$, is a* shortest path from P to Q in $\mathcal{K}$, *iff* $\forall i, 0 < i \leq m, P_{i-1} \in Inner_{\mathcal{K}}(P_i, P)$.[3]
6. *K is* convex with respect to $\mathcal{K}$ *iff every shortest path, in $\mathcal{K}$, between any two points in K is contained in K.*
7. *For any point P in K we say that K is* P-convex with respect to $\mathcal{K}$ *iff, for every point Q in K, $Inner_{\mathcal{K}}(Q,P) \subseteq K$.*

The following lemma characterizes subsets K of $\mathcal{K}$ which are convex with respect to $\mathcal{K}$ for lattices $\mathcal{K}$.

Lemma 1 *Suppose $\mathcal{K}$ is a lattice, and $K \subseteq \mathcal{K}$. Then K is convex with respect to $\mathcal{K}$ iff, for each point P in K, K is P-convex with respect to $\mathcal{K}$.* □

Definition 2 *Let K be a subset of an n-dimensional lattice $\mathcal{K} \subseteq \mathbf{R}^{n'}, n' \geq n$.*

[3] $\mathrm{GraphPL}_{\mathcal{K}}(P,Q) = m = \mathrm{GraphPL}_{\mathcal{K}}(Q,P)$.

1. *For points P and Q in $\mathcal{K}$, $d(P,Q)$ denotes the* euclidean *distance between P and Q, (P,Q) denotes the standard inner product in $\mathbf{R}^{n'}$ of the vectors P and Q in $\mathbf{R}^{n'}$, and $\|P\| \stackrel{\text{def}}{=} d(P,\vec{0})$.*

2. *K is* non-trivial *iff there are three distinct non-collinear points, P, Q, and O, in K so that $d(P,O) \neq d(Q,O)$.*

Definition 3

1. *Let $n \leq n'$. Then a function $E : \mathbf{R}^n \rightarrow \mathbf{R}^{n'}$ is an* isomorphism from $\mathbf{R}^n$ *(onto its image, $E(\mathbf{R}^n) = \{E(x) \in \mathbf{R}^{n'} \mid x \in \mathbf{R}^n\}$) iff, for any x, y in $\mathbf{R}^n$, and any real numbers α, β, (a) $E(\alpha x + \beta y) = \alpha E(x) + \beta E(y)$, (i.e., E is a linear transformation) and, (b) $\exists$ a real number $\sigma > 0$ so that $\|E(x)\| = \sigma \|x\|$ (which implies that E preserves angles). The image of an isomorphism from $\mathbf{R}^n$ is called a* perfect representation of $\mathbf{R}^n$. *(It can be easily shown that isomorphisms, as defined, are one-to-one.)*

   Suppose $\mathcal{K} \subseteq \mathbf{R}^{n'}, n' \geq n$ is an n-dimensional lattice. Then, $\mathbf{E}^n_{\mathcal{K}}$ denotes the unique *perfect representation of $\mathbf{R}^n$ containing $\mathcal{K}$.*[4]

2. *Suppose $\mathcal{K} \subseteq \mathbf{R}^{n'}, n' \geq n$ is an n-dimensional lattice. For each point P in $\mathcal{K}$, the* Voronoi cell around P in $\mathcal{K}$, *denoted* $\text{Voronoi}(P,\mathcal{K})$, *is the set of points in $\mathbf{E}^n_{\mathcal{K}}$ that are closer to P than to any other point in $\mathcal{K}$. The* closure of the Voronoi cell around P in $\mathcal{K}$ *is denoted by* $\overline{\text{Voronoi}(P,\mathcal{K})}$.

We now state some facts regarding Voronoi cells around points in lattices.[5]

Proposition 1 *Suppose $\mathcal{K}$ is an n-dimensional lattice. Then, Voronoi cells around points in $\mathcal{K}$ tessellate $\mathbf{E}^n_{\mathcal{K}}$, i.e., (1) for any two distinct points, P and Q, in $\mathcal{K}$, $\text{Voronoi}(P,\mathcal{K}) \cap \text{Voronoi}(Q,\mathcal{K}) = \emptyset$, and (2) $\overline{\bigcup_{P \in \mathcal{K}} \text{Voronoi}(P,\mathcal{K})} = \mathbf{E}^n_{\mathcal{K}}$. (Here the overbar denotes closure.)* □

Proposition 2 *Each Voronoi cell around a lattice point is a translate of the Voronoi cell around every other lattice point.* □

So, it suffices to define concepts, and state properties of Voronoi cells around points in a lattice $\mathcal{K}$, in terms of $\text{Voronoi}(\vec{0},\mathcal{K})$.

Proposition 3 *Suppose $\mathcal{K}$ is an n-dimensional lattice. Then, $\overline{\text{Voronoi}(\vec{0},\mathcal{K})}$ is a convex, n-dimensional polytope*[6] *bounded by $(n-1)$-dimensional faces.* □

[4] For example, the plane $\{(x,y,z) \in \mathbf{R}^3 \mid x + y + z = 0\}$ is the perfect representation of $\mathbf{R}^2$ containing A_2.

[5] See [CS88] for more information about lattices and Voronoi cells around points in lattices. The Voronoi cells in [CS88] correspond to the *closure* of our Voronoi cells. Voronoi cells (and diagrams) of sets of points in *two* dimensional euclidean space have been studied extensively by researchers in Computational Geometry. Similarly, some line and curve drawing algorithms in Graphics turn out to light up a picture element (pixel) on a *two* dimensional square grid of pixels iff the line or curve passes through the Voronoi cell around the center of the pixel. [Aur90] is an excellent survey of all these applications of Voronoi cells. Nonetheless, the way Voronoi cells are employed in these areas differs significantly from our usage.

[6] Essentially, an *n-dimensional polytope* is the n-dimensional analog of a 3-dimensional polyhedron.

Definition 4 *Suppose $\mathcal{K}$ is an n-dimensional lattice. A vector (point) P in $\mathcal{K}$ is called a* relevant vector in $\mathcal{K}$ *iff (1) P is perpendicular to a face, F_P, of $\overline{\text{Voronoi}(\vec{0}, \mathcal{K})}$, and (2) $\frac{1}{2} \cdot P \in F_P$. The set of relevant vectors in $\mathcal{K}$ is denoted* $\text{Rel}(\mathcal{K})$.

We now state some facts about the set of relevant vectors in lattices. The following two propositions essentially provide a method to calculate membership in the closure of a Voronoi cell in finite time.[7]

Proposition 4 *For a lattice, $\mathcal{K}$, $\text{Rel}(\mathcal{K})$ is finite.* □

Proposition 5 *Suppose $\mathcal{K}$ is a lattice. Then, $x \in \overline{\text{Voronoi}(\vec{0}, \mathcal{K})} \iff (\forall R \in \text{Rel}(\mathcal{K}))\left[(x, R) \leq \frac{(R,R)}{2}\right]$.* □

3 Well-Represented Lattices and Meshes

As we noted in Section 1, our algorithms work in a wide class of n-dimensional lattices called *well-represented*. In this section we define these, provide some useful technical results about Voronoi cells in them, and define an associated class of meshes called *suitable* in which our algorithms work.

Definition 5 *Suppose $\mathcal{K}$ is an n-dimensional lattice. $\mathcal{K}$ is said to be* well-represented *iff*

1. *there is a basis for $\mathcal{K}$ containing only minimal length vectors (we will then say that $\mathcal{K}$ is* minimally generated*),*[8]
2. *there is an $n' \geq n$ and an n-dimensional lattice $\mathcal{K}' \subseteq \mathbf{Z}^{n'}$ that is isomorphic (angle preserving) to $\mathcal{K}$ (we will then say that $\mathcal{K}$ is* integrally representable by $\mathcal{K}'$*),*[9] *and*
3. $\text{Rel}(\mathcal{K}) = \text{Dirs}(\mathcal{K})$.

We then say that $\mathcal{K}'$ well-represents *$\mathcal{K}$ in $\mathbf{Z}^{n'}$.*

Clause 3 in Definition 5 just above is important. In our algorithms, we will naturally want to send messages across the face of a Voronoi cell into its neighbor. It is useful for the *timing* of such message passing if the relevant vector between the corresponding lattice points of the Voronoi cells is minimal length, for, then, in a mesh based on the lattice, the processors corresponding to the lattice points are *directly* connected. See also Proposition 6 below.

Convention 1 *Henceforth, $\mathcal{L}$ will denote an n-dimensional, well-represented lattice that well-represents itself in $\mathbf{Z}^{n'}, n' \geq n$. We will abbreviate $\mathbf{E}^n_{\mathcal{L}}$ to $\mathbf{E}^n$.*

---

[7] Also see [CS88].

[8] This guarantees that the graph induced by $\mathcal{M}_{\mathcal{K}}(\mathcal{K})$ (with processors as vertices, and bi-directional communication channels as undirected edges) is connected. See [CRS90a].

[9] This enables us to carry out computations for our algorithms in $\mathcal{M}_{\mathcal{K}}(\mathcal{K})$ using integer and integer vector arithmetic only. See also [CRS90a, CRS90b].

3.1 Voronoi Cells in Well-Represented Lattices

In this section we state some results that will be used in Section 6.5 below.

Proposition 6 $x \in \overline{\text{Voronoi}(\vec{0}, \mathcal{L})} \iff (\forall D \in \text{Dirs}(\mathcal{L}))\ \left[(x, D) \leq \frac{(D,D)}{2} = \frac{\text{MinLgth}(\mathcal{L})^2}{2}\right]$.

Proof. Follows immediately From Proposition 5 and Definition 5. □

Definition 6 *For any two points a and b in $\mathbf{R}^n$, $[a, b) = \{x \in \mathbf{R}^n | \exists t \in [0, 1), x = a + t \cdot (b - a)\}$, i.e., $[a, b)$ is the line segment, in $\mathbf{R}^n$ from a to b including the point a and not including the point b.*

Lemma 2 *Let L be a finite subset of $\mathcal{L}$. Suppose a and b are two points in the convex hull of L, so that $a \in \overline{\text{Voronoi}(P, \mathcal{L})}$ and $b \in \overline{\text{Voronoi}(Q, \mathcal{L})}$, $P, Q \in L$. Then, there exists a finite sequence of points $(P =)P_0, P_1, \ldots, P_k(= Q)$, each $P_i, 0 \leq i \leq k$ in $\mathcal{L}$, so that*

1. $\forall i, 0 \leq i < k, (P_{i+1} - P_i) \in \text{Dirs}(\mathcal{L})$,
2. $\forall i, 0 \leq i \leq k, ([a, b) \cap \overline{\text{Voronoi}(P_i, \mathcal{L})}) \neq \emptyset$, *and*
3. $[a, b) \subset \overline{\cup_{i=0}^{k} \text{Voronoi}(P_i, \mathcal{L})}$.

□

We will use the following obvious fact in the proof of Proposition 4 below.

Lemma 3 *Suppose A, B are real numbers. Then,*

$$(A \leq B \text{ and } [A, B] \cap [0, 1] \neq \emptyset) \iff (\max\{0, A\} \leq \min\{1, B\}).$$

□

Lemma 4 *Let L be a finite subset of $\mathcal{L}$. Let x_1 and x_2 be two points in the convex hull of L. Let*

$$\begin{aligned} \text{Dirs}_0(x_1, x_2, \mathcal{L}) &= \{D \in \text{Dirs}(\mathcal{L}) | ((x_2 - x_1), D) = 0\} \text{ and} \\ \text{Dirs}_+(x_1, x_2, \mathcal{L}) &= \{D \in \text{Dirs}(\mathcal{L}) | ((x_2 - x_1), D) > 0\}. \end{aligned}$$

Then, for all points $P \in L$, there is a point in $([x_1, x_2) \cap \overline{\text{Voronoi}(P, \mathcal{L})})$ iff

$$\begin{aligned} &\max\{0, \max\{\frac{(P - x_1, D) - (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)} \mid D \in \text{Dirs}_+(x_1, x_2, \mathcal{L})\}\} \\ &\leq \min\{1, \min\{\frac{(P - x_1, D) + (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)} \mid D \in \text{Dirs}_+(x_1, x_2, \mathcal{L})\}\} \\ &< 1, \end{aligned} \tag{1}$$

and

$$\begin{aligned} &\max\{((P - x_1), D) - (\text{MinLgth}(\mathcal{L})^2/2) \mid D \in \text{Dirs}_0(x_1, x_2, \mathcal{L})\} \\ &\leq 0 \\ &\leq \min\{((P - x_1), D) + (\text{MinLgth}(\mathcal{L})^2/2) \mid D \in \text{Dirs}_0(x_1, x_2, \mathcal{L})\}. \end{aligned} \tag{2}$$

PROOF. Using Proposition 6, $[x_1, x_2)$, has a non-empty intersection with $\overline{\text{Voronoi}(P, \mathcal{L})}$, iff there is a real number $t \in [0, 1)$, so that, for each D in $\text{Dirs}(\mathcal{L})$, $|((t(x_2 - x_1) + x_1 - P), D)| \leq \text{MinLgth}(\mathcal{L})^2/2$. Notice that the value of the left hand side of this inequality unchanged if D is replaced by $(-D)$. If $D \in \text{Dirs}_0(x_1, x_2, \mathcal{L})$, then this inequality reduces to

$$(-\text{MinLgth}(\mathcal{L})^2/2) - ((x_1 - P), D) \leq 0 \leq (\text{MinLgth}(\mathcal{L})^2/2) - ((x_1 - P), D). \tag{3}$$

If, on the other hand, $D \in \text{Dirs}_+(x_1, x_2, \mathcal{L})$, then the inequality reduces to

$$\frac{(P - x_1, D) - (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)} \leq t \leq \frac{(P - x_1, D) + (\text{MinLgth}(\mathcal{L})^2/2)}{((x_2 - x_1), D)}. \tag{4}$$

Now then, $[x_1, x_2) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$ iff (i) for each D in $\text{Dirs}_0(x_1, x_2, \mathcal{L})$, (3) holds, *and* (ii) there is a t in $[0, 1)$, so that, for each D in $\text{Dirs}_+(x_1, x_2, \mathcal{L})$, (4) holds. Now this lemma follows from Lemma 3. □

3.2 Meshes Based on Well-Represented Lattices

For a point P in a n-dimensional lattice $\mathcal{K}$, we will write $\mathcal{M}_{\mathcal{K}}(P)$ for $\mathcal{M}_{\mathcal{K}}(\{P\})$.

Definition 7 *Suppose L is a subset of $\mathcal{L}$. Then, $\mathcal{M}_{\mathcal{L}}(L)$ is a* suitable mesh *iff (1) $n > 1$, (2) L is convex with respect to $\mathcal{L}$ and non-trivial,* and *(3) $\mathcal{M}_{\mathcal{L}}(L)$ satisfies Assumption 1 below.*

Assumption 1

1. *All communication channels in $\mathcal{M}_{\mathcal{L}}(L)$ are identical in their abilities to carry data between processors. We assume that all communication channels are capable of carrying the same amount of data in the same amount of time.*

2. *There is a common, finite set of basic instructions that every processor in $\mathcal{M}_{\mathcal{L}}(L)$ can execute. These are the only basic instructions that any processor in $\mathcal{M}_{\mathcal{L}}(L)$ can execute. Furthermore, the set of basic instructions must be sufficient to encode, at least, the specific application of Algorithm* CURVE *described in Section 7, below. For example, for receiving and sending immediate interprocessor information it would be helpful to have a system of in and out buffers in each processor.*

3. *The actual time to execute the same basic instruction by any processor in $\mathcal{M}_{\mathcal{L}}(L)$ is a constant. By* instruction cycle of a processor *we mean the minimum possible time interval within which any single instruction can be fully executed by any processor. Hence, in each processor, each basic instruction takes the same instruction cycle amount of time to execute.*

4. *All processors in $\mathcal{M}_{\mathcal{L}}(L)$ execute basic instructions synchronously, i.e., instruction cycles of all processors share a same, actual time interval. This discretizes time, for the mesh as a whole, into intervals of equal length.* `SimClock` *is the instruction counter for each $\mathcal{M}_{\mathcal{L}}(P)$.*

5. *There is a basic instruction*

$$\textbf{idle till } \texttt{SimClock} = X \tag{5}$$

and its semantics dictates that, it iterates passes of itself until, if ever, `SimClock`$= X$ *is true; if* `SimClock`$= X$ *is true, then the next instruction after (5) is instruction number* $X + 1$. *(Of course, then, each pass of the instance of the* **idle till** *takes exactly one instruction cycle.*[10]*)*

6. *In* one *instruction cycle, each processor,* $\mathcal{M}_{\mathcal{L}}(P)$ *in* $\mathcal{M}_{\mathcal{L}}(L)$, *can* simultaneously *send messages to any subset of the processors in* $\mathcal{M}_{\mathcal{L}}(\text{Nbrs}(P) \cap L)$, *one message to each element of the subset, possibly different messages to different elements of the subset.*

Convention 2 *In addition to Convention 1, henceforth,* L *will denote a subset of* $\mathcal{L}$ *so that* $\mathcal{M}_{\mathcal{L}}(L)$ *is a suitable mesh. Also, from now on, since we will be dealing only with the lattice* $\mathcal{L}$, *we will drop the subscript* $\mathcal{L}$.

4 Representation of Curves in Well-Represented Lattices

Suppose a particle starts at a point in L, and traces, over a period of time, a curve in $\mathbf{E}^n$ in which the points (in $\mathbf{E}^n$) that the curve passes through are given by $\rho(t)$, $0 \leq t \in \mathbf{R}$, t a time paramenter. Further suppose for this paper that the traversal is at constant speed. We would like to simulate the motion of this particle in $\mathcal{M}(L)$.

Since we will be performing the simulation in a mesh (based on a lattice) consisting of discrete points, we need to define the representation, in the mesh, of the curve ρ. We would like each lattice point to represent a region of euclidean space around it. We will call this region of euclidean space around a lattice point P, the *region of influence of* P. We would, then, like to have a lattice point be a part of the representation of a curve iff the representation of that curve passes through the region of influence of that lattice point. A natural region of influence, of a lattice point P, that one might choose, is the set of points x, in $\mathbf{E}^n$ so that x is closer, or as close, to P than every other lattice point, i.e., the closure of the Voronoi cell (Definition 3) around P in $\mathcal{L}$.

We approximate ρ by the sequence of points, C, on ρ, so that $\rho(0) \in C$ and the length of the arc segment of ρ between any two consecutive points, along ρ, in C, is a positive, real constant s. Then, we treat ρ as "equivalent" to the curve, ρ_s, which is made up (for each k, a non-negative integer) of straight line segments between consecutive points, $\rho_s(k)$ and $\rho_s(k+1)$ in C.[11] Our algorithm, actually, will simulate the motion of a particle, whose motion is given by ρ, as if the particle were moving along ρ_s, traversing each arc segment (of arc length s) in the same amount of time.

Remark 1 *We adopt as our (static) representation of* ρ_s *in* $\mathcal{M}(L)$ *the set* $\{X \in L \mid \exists x \in \mathbf{E}^n, x \in \textit{range}(\rho_s) \textit{ and } x \in \overline{\text{Voronoi}(X, \mathcal{L})}\}$.

[10] For many hardware designs, for example, that of [CRS90b], it suffices to implement (5) with a suitably chosen number, IT, of instruction cycles per pass. Such implementations are not, in general, particularly straightforward, so for simplicity here, we have assumed a hardware for which IT is 1.

[11] We assume that, for all $k \geq 0$ $\rho_s(k+1) \neq \rho_s(k)$.

5 Simulating the Uniform Motion of a Particle

The progress along a curve, ρ_s, in $\mathcal{M}(L)$ is achieved in the simulation by the passage of messages from processor to neighboring processor. Thus the fastest that the particle could travel in the simulation is s units of euclidean distance in one unit of simulation time.[12] This fastest speed cannot be achieved for the following reason. It may be the case that at simulation time t_s the particle is in the Voronoi cell around the lattice point P, and at simulation time $t_s + 1$, the particle is in the Voronoi cell around a lattice point which is *not* a neighbor of P. Nevertheless, we can overcome this problem by introducing a dilation factor for simulation time (or, in other words, slow down the simulation). Towards this, we define a few concepts first.

Definition 8 *Suppose the non-trivial straight line segment* $[a, b) \subseteq \mathbf{E}^n$.

1. *A sequence of points from* $\mathcal{L}, \langle P_1, \ldots, P_k \rangle$, *is a* covering path for $[a, b)$ in $\mathcal{L}$ *(written:* $\mathrm{Cov}(\langle P_1, \ldots, P_k \rangle, [a, b), \mathcal{L})$*) iff*
   (a) $(\forall i, 1 \leq i < k)[P_i$ *is a neighbor of* $P_{i+1}]$,
   (b) $(a \in \overline{\mathrm{Voronoi}(P_1, \mathcal{L})}$ *and* $b \in \overline{\mathrm{Voronoi}(P_k, \mathcal{L})})$,
   (c) $(\forall i, 1 \leq i \leq k)[([a, b) \cap \overline{\mathrm{Voronoi}(P_i, \mathcal{L})}) \neq \emptyset)$, *and*
   (d) $[a, b) \subseteq \cup_{1=1}^{k} \overline{\mathrm{Voronoi}(P_i, \mathcal{L})}$.
2. *The length of such a covering path* $\langle P_1, \ldots, P_k \rangle$ *is (by definition)* k *and is denoted by* $\mathrm{CPL}(\langle P_1, \ldots, P_k \rangle)$.
3. $\mathrm{MinCov}([a, b), \mathcal{L}) \stackrel{\mathrm{def}}{=} \min\{\mathrm{CPL}(\langle P_1, \ldots, P_k \rangle) \mid \mathrm{Cov}(\langle P_1, \ldots, P_k \rangle, [a, b), \mathcal{L})\}$.
4. $\mathrm{MPL}_s(\mathcal{L}) \stackrel{\mathrm{def}}{=}$
   $\max\{\mathrm{MinCov}([a', b'), \mathcal{L}) \mid [a', b')$ *is a non-trivial line segment in* $\mathbf{E}^n$ *and* $d(a', b') \leq s\}$.

Clearly, then, if the speed, in the simulation, of the particle is slowed down by a factor of, at least $\mathrm{MPL}_s(\mathcal{L})$, we do not have the problem mentioned above.

It may be difficult to calculate the value of $\mathrm{MPL}_s(\mathcal{L})$, for arbitrary well-represented lattices $\mathcal{L}$. It is likely possible, though not easy, to calculate $\mathrm{MPL}_s(\mathcal{L})$ for $\mathcal{L}$ in certain families of lattices (e.g., root lattices [CS88]) since, in these cases, there exist (often tedious) algorithms to generate the set of vertices of Voronoi cells around the origin of $\mathcal{L}$. Nonetheless, we can calculate $\mathrm{MPL}_s(\mathbf{Z}^n)$ and $\mathrm{MPL}_s(A_n)$ rather easily.[13]

In Propositions 7 and 8 below, we give the values for $\mathrm{MPL}_s(\mathbf{Z}^n)$ and $\mathrm{MPL}_s(A_n)$, respectively.[14] Due to space constraints, we will give the proof for Proposition 7, but not for Proposition 8. The proof of the latter proposition is similar to the proof of the former except for the use of Lemma 5. $2\mathrm{Voronoi}(\vec{0}, \mathcal{L})$ denotes the magnification of the Voronoi cell around the origin, in $\mathcal{L}$, by a factor of 2.

[12] In $\mathcal{M}(L)$, an instruction cycle (see Assumption 1) is the smallest unit of simulation time. Nonetheless, for most algorithms, processors will need to perform some computations upon receiving messages before sending any messages. For any algorithm, if the amount of time (number of instruction cycles) needed to perform these computations by any processor is bounded by a constant, T, that can be determined prior to executing the algorithm, then T can be chosen as the unit of simulation time. For the algorithm presented in Section 7 below, we show that T exists and can be calculated prior to executing the algorithm.

[13] See Footnote 1 for definitions of $\mathbf{Z}^n$ and A_n. Both these lattices are well-represented. See [CRS90a] for our interest in these lattices, particularly in A_n.

[14] The upper bound on s in each of these propositions can, perhaps, be improved; for our purposes, this upper bound suffices.

Proposition 7 $\forall s < 1/2, \mathrm{MPL}_s(\mathbf{Z}^n) = (n+1)$.

PROOF. For all $s < 1/2$, every point within s units of Voronoi$(\vec{0}, \mathbf{Z}^n)$ is contained in 2Voronoi$(\vec{0}, \mathbf{Z}^n)$. Now consider, C^n, the union of the standard lattice cubes, each of side 1, and each with the origin of $\mathbf{Z}^n$ as one of its vertices. Then, clearly, 2Voronoi$(\vec{0}, \mathbf{Z}^n)$ is contained in C^n. Therefore, for any segment $[a, b)$ of euclidean length $< 1/2$, so that $a, b \in \mathbf{R}^n$ and $a \in$ Voronoi$(\vec{0}, \mathbf{Z}^n)$, MinCov$([a, b), \mathbf{Z}^n) \leq (n+1)$. Furthermore, consider a segment $[a', b')$ so that its length is less than $1/2$, $a' \in$ Voronoi$(\vec{0}, \mathbf{Z}^n)$, and the segment contains the point of intersection of the Voronoi cell around the origin and around the point whose co-ordinates are all 1's. Clearly, MinCov$([a', b'), \mathbf{Z}^n) = (n+1)$. Therefore, $\forall s < 1/2, \mathrm{MPL}_s(\mathbf{Z}^n) = (n+1)$. □

Lemma 5 *Suppose* $x \in$ 2Voronoi$(\vec{0}, A_n)$ *and* $x \in \overline{\text{Voronoi}(P, A_n)}$, *where* $P = (p_0, p_1, \ldots, p_n)$. *Then,* $\forall i, 0 \leq i \leq n, |p_i| \leq 1$. □

Proposition 8 $\forall s < \mathrm{MinLgth}(A_n)/2, \mathrm{MPL}_s(A_n) = \lfloor \frac{n+1}{2} \rfloor + 1$. □

6 Algorithm CURVE

In this section, we describe an algorithm *template*, CURVE, for the simulation of a particle moving along a curve ρ_s (see Section 4 above) at constant speed.[15]

Under the assumption that the values of all boolean expressions and all declared variables (see Sections 6.1 and 6.3 below) can be computed exactly, we show, via Theorem 1, that the suitable mesh $\mathcal{M}(L)$, executing Algorithm CURVE, simulates a particle moving along the curve ρ_s so that the speed of the particle in the simulation is (approximately) constant. Theorem 1 expresses the correctness of the algorithm. Corollary 1 expresses the correctness of the representation, in $\mathcal{M}(L)$, of the curve ρ_s by Algorithm CURVE.

To completely specify instances of CURVE, it turns out to suffice to further specify:

1. additional contents of messages, and

2. how to perform the tests in step 2 of Algorithm CURVE (see Section 6.4 below).

In Section 6.6, we elaborate on the tests that need to be performed in step 2 in Algorithm CURVE, and show that all these tests involve rational expressions in $\rho_s(k)$ and $\rho_s(k+1)$. Later, in Section 7, we will enumerate the contents of messages, and will give the exact expressions for $\rho_s(k)$, for a specific instance of Algorithm CURVE, namely translation at constant speed. There, we will also show that, in this specific instance, the values of all boolean expressions and declared variables can be computed exactly using integers only. Due to space limitations, we present only this one instance of Algorithm CURVE here. Details for another instance, namely the constant angular-speed revolution of a particle, can be found in [She91].

We make the following assumptions regarding the execution of (an instance of) Algorithm CURVE.

[15] For simplicity we'll hereinafter refer to this algorithm template as an algorithm.

Assumption 2

1. *When running Algorithm* CURVE *we think of the algorithm being present in each processor in* $\mathcal{M}(L)$.

2. *For purposes of simplicity and exposition, at any time that* $\mathcal{M}(L)$ *is running one of the instances of Algorithm* CURVE, *there is not some unrelated process, e.g., another instance of Algorithm* CURVE, *running at the same time. (We do not discuss here, for example, the interesting problem of the simulation of two or more particles undergoing different, or perhaps the same, kinds of motion.)*

3. *For an instance of Algorithm* CURVE, *simulating a particle traversing a curve* ρ *at constant speed, there will be a specified processor,* $\mathcal{M}(I) = \rho(0)$, *where the motion simulated by that instance will begin. This* $\mathcal{M}(I)$ *will be called, only at the beginning of the simulation, the Initiator.*

4. *At any time, a processor is said to have completed a* pass *of Algorithm* CURVE *iff at that time it completes executing step 3 of Algorithm* CURVE.

5. *As soon as a processor,* $\mathcal{M}(P)$, *has completed a pass of Algorithm* CURVE, $\mathcal{M}(P)$ *goes back to executing step 1 of Algorithm* CURVE.

6. *Algorithm* CURVE, *in each processor,* $\mathcal{M}(P)$, *has its own, predefined, "automatic" variables called* `SimClock` *and* `PassCount` *that can be accessed.* `SimClock`, *at each time, contains the integer number of instruction cycles completed since the beginning of the simulation.* `PassCount`, *at each time, contains the integer number of passes completed by* $\mathcal{M}(P)$ *since the beginning of the simulation.* `SimClock` *is "automatic" in the sense that, with each instruction cycle completed by* $\mathcal{M}(P)$, *this variable is automatically incremented by one in* $\mathcal{M}(P)$. `PassCount` *is "automatic" in the sense that, with each pass, of Algorithm* CURVE, *completed by* $\mathcal{M}(P)$, *this variable is automatically incremented by one in* $\mathcal{M}(P)$.

7. *A simulation begins, exactly when an external (to the mesh) input turns on all the processors to begin executing Algorithm* CURVE *and that external input initializes the variable* `Done` *to* **false** *in the Initiator and to* **true** *in all the other processors in* $\mathcal{M}(K)$. *The initialization of the Initiator will sometimes be referred to as its activation.*

8. *For our purposes of simplicity and exposition, at any time after a simulation begins, no external input which influences the progress of simulation is provided to any processor in the mesh.*

9. *We will assume that* $s < \text{MinLgth}(\mathcal{L})/2$, *is a rational number.*

6.1 Constants

const `PathLgthDil`;

This is the extra temporal dilation factor $\text{MPL}_s(\mathcal{L})$ introduced in Section 5 above. Its value is set based on the lattice $\mathcal{L}$ on which the mesh is based and the value of s.

const `CompLen`;

`CompLen` is an integer constant dependent on the coding of the particular instance of Algorithm CURVE being executed. Its value, which can be calculated prior to the execution of the algorithm, is the maximum time that any processor takes to complete one pass of the algorithm.

6.2 Types

type `Directions`= Dirs(L);

type *MessageType* = ⟨ A record whose fields hold information appropriate
for executing Algorithm CURVE. ⟩;

Values of this type are finite sequences of software entities, which are either integers, or integer vectors. Such values represent the messages that can be sent from a processor to any one of its neighbors. These messages are the only means for a processor to communicate with neighboring processors.

type *DirectionBuffer* = **record**
InMessage, *OutMessage* : *MessageType*;
SendTime : `integer`;
⋮
end;

Values of this type are aggregates of values of type *MessageType*, integers, or integer vectors, and values of basic types (integer and boolean). Values of this type contain all relevant information pertaining to basic directions in the mesh.

6.3 Variables

var `Done`: **boolean**;

At the beginning of a simulation, the boolean variable `Done` in each processor, $\mathcal{M}(P)$, except the *Initiator*, has the value **true**. The *Initiator* has the value **false**. This variable is set to **false** automatically when $\mathcal{M}(P)$ receives a message. It can also be set to **true** or **false** by an algorithm.

var *D* : `Directions`;

This is a variable ranging over the values of type `Directions`.

var *InDirs*, *SendDirs*, *SendThisPass* : **set of** `Directions`;

These are variables that range over the subsets of the set of values of type `Directions`. *InDirs* is used to store the set of directions from which a processor gets messages; at the beginning of each simulation it has the value $\emptyset$. *SendDirs* is used to store the set of directions in which the processor needs to send messages, and *SendThisPass* is used to store the set of basic directions in which the processor needs to send messages at the end of the current pass.

var *Buf* : *DirectionBuffer* **indexed by** `Directions`;

This variable stores an indexed set, whose elements are elements of *DirectionBuffer* indexed by an element of `Directions`. We do not assume any ordering on the set `Directions`, and hence no ordering on the elements

of this variable. By definition, this set *does not* contain two elements that are indexed by the same element of `Directions`. We note here two ways to implement this variable, but any implementation achieving this effect will do. One way to implement this variable is as an array variable, indexed by the values of type `Directions`, and containing values of type *DirectionBuffer*. An alternate method for implementing this variable is as a function from `Directions` to values of type *DirectionBuffer*. The aim is to be able to associate, with each basic direction, a value of type *DirectionBuffer*. At any stage in the simulation, for each basic direction in the mesh there is a value of type *DirectionBuffer*.

var REPR : **boolean**;

Roughly, in each processor, REPR = **true** during a pass is used to denote whether the phenomenon being simulated is being represented by that processor sometime during that pass.

REPR is initialized to **false** in each processor, at the beginning of a simulation. Specific instances of Algorithm CURVE set and reset this variable in steps 1 and/or 2 of Algorithm CURVE.

var *leftend*;

This variable is initialized externally for the *Initiator*, $\mathcal{M}(I)$, to 0. When, if ever, a processor, $\mathcal{M}(P)$, receives a message from another processor, $\mathcal{M}(P)$ initializes the value of this variable from an appropriate field in that message. If the value received is k, this means that the particle being simulated is on the line segment $[\rho_s(k), \rho_s(k+1))$. $\mathcal{M}(P)$ can use and set *leftend*.

var t_{in};

This variable is initialized externally for the *Initiator*, $\mathcal{M}(I)$, to 0. When, if ever, a processor, $\mathcal{M}(P)$, receives a message, $\mathcal{M}(P)$ initializes the value of this variable from an appropriate field in that message. If the value received is k, this means that this message was sent, to $\mathcal{M}(P)$, at the end of k passes. $\mathcal{M}(P)$ can use and set t_{in}.

6.4 Executable Part of Algorithm CURVE

We describe Algorithm CURVE as if it were being executed by processor $\mathcal{M}(P)$.

begin

1. **if** `Done` **then goto** step 3;
   else
   if (**not** REPR) **then**
   Initialize Parameters using external input, if any, and
   $\{ Buf\langle D\rangle.InMessage | D \in InDirs \}$, if non-empty;
   endif;
   endif;
2. **if** $[\rho_s(leftend), \rho_s(leftend+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} = \emptyset$ **then**
   REPR := **false**;
   Done := **true**;
   goto step 3;

else

REPR := **true**;

$SendDirs := \emptyset$;

if not $(\rho_s(leftend+1) \in \text{Voronoi}(P, \mathcal{L}))$ **then**

$SendDirs := \{D | ((\rho_s(leftend+1) - P), D) \geq \text{MinLgth}(\mathcal{L})^2/2\}$;

if $(\rho_s(leftend+1) \in \overline{\text{Voronoi}(P, \mathcal{L})})$ **then**

for each $D \in SendDirs$,

$Buf\langle D\rangle.OutMessage.t_{in} :=$

$\texttt{PathLgthDil} \cdot (leftend+1)$;

$Buf\langle D\rangle.OutMessage.leftend := leftend+1$;

endfor;

else

for each $D \in SendDirs$,

$Buf\langle D\rangle.OutMessage.t_{in} := t_{in}+1$;

$Buf\langle D\rangle.OutMessage.leftend := leftend$;

endfor;

endif;

endif;

$t_{in} := t_{in}+1$;

$leftend := (t_{in} \textbf{ div } \texttt{PathLgthDil})$;

$SendThisPass :=$

$\{D \in SendDirs \mid Buf\langle D\rangle.OutMessage.t_{in} = \texttt{PassCount}\}$;

3. **idle till** $\texttt{SimClock} = (\texttt{CompLen} \cdot \texttt{PassCount}) - 1$;

simul send

$Buf\langle D\rangle.OutMessage$ in direction D, for each $D \in SendThisPass$;

endsend;

end.

6.5 Correctness of Algorithm CURVE

We first define some terminology useful in expressing the correctness of Algorithm CURVE.

Notation 1

1. *For natural numbers $a < b$, $(a, b]$ will denote the set $\{x \in \mathbf{N} | a < x \leq b\}$.*[16]

2. *For $S \subseteq \mathbf{N}$, and natural numbers $a < b$, $(a, b] \sqsubseteq S$ will denote the statement $(\exists y \in \mathbf{N})(\forall x \in \mathbf{N} \mid a < y \leq x \leq b)[x \in S]$.*[17] *Note that, if $(a, b] \subseteq S$ then $(a, b] \sqsubseteq S$.*

[16] Context will resolve any ambiguity as to whether intervals are real or integer.

[17] Of course $(a, b] \sqsubseteq S$ is mathematically equivalent to $b \in S$.

3. *For succinctness, we will sometimes use PL for* `PathLgthDil` *and CL for* `CompLen`.

4. *If during a simulation time*[18] *interval $(t_s, t_s + 1]$, REPR is* **true**, *in the copy of the program (encoding Algorithm* CURVE*) being executed by $\mathcal{M}(P)$, then we say that $\mathcal{M}(P)$ is in the state REPR at simulation time $(t_s + 1)$. $REPR(P)$ is the set of all simulation time units at which $\mathcal{M}(P)$ is in the state REPR.*

5. *We say that* $\mathcal{M}(P)$ is executing the k-th segment passes[19] *when and only when it is executing a pass in which leftend $= k$, and before the end of that pass, $\mathcal{M}(P)$ is in state REPR.*

6. *We will refer to the beginning of the first pass in the execution, by $\mathcal{M}(P)$, of the k-th segment passes of the algorithm, as the* beginning of the kth segment passes *for $\mathcal{M}(P)$.*

7. *If during a pass in the execution of the k-th segment passes of the algorithm by $\mathcal{M}(P)$, REPR is* **true**, *then that pass is the first pass, executed by $\mathcal{M}(P)$ in the k-th segment passes.*

8. *We will refer to the value of t_{in} in $\mathcal{M}(P)$ at the beginning of the k-th segment passes for $\mathcal{M}(P)$, by $t^k_{in}(P)$.*

9. *If, in the k-th segment passes, $\mathcal{M}(P)$ computes SendDirs to be a non-empty set, then the common value to which $Buf\langle D\rangle.OutMessage.t_{in}$ is set, for each D in SendDirs will be referred to as $t^k_{out}(P)$.*

Now the following lemmas help us prove Theorem 1 stated below. We sketch proofs wherever necessary.

Lemma 6 *A processor, $\mathcal{M}(P)$, other than the Initiator, starts executing step 2 of the algorithm iff it gets a message from some neighbor.* □

Lemma 7 *If $\mathcal{M}(P)$ executes the k-th segment passes of the algorithm, then $[\rho_s(k), \rho_s(k+1))$ has a non-empty intersection with the Voronoi cell around P.*

Proof. Follows from the definition of k-th segment passes and the algorithm. □

Lemma 8 *In the k-th segment passes, SendDirs computed by $\mathcal{M}(P)$ is non-empty iff $\rho_s(k+1) \notin$ Voronoi$(P, \mathcal{L})$. Furthermore, if SendDirs is non-empty, then $\mathcal{M}(P)$ sends messages to all those neighbors whose Voronoi cells might possibly contain $[\rho_s(k), \rho_s(k+1))$.*

Proof. Follows from the computation of *SendDirs* in the algorithm and Proposition 6. □

Lemma 9 *Suppose $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$ and $\rho_s(k+1) \in \overline{\text{Voronoi}(Q, \mathcal{L})}$. Suppose $REPR(P) \neq \emptyset$. Then, $t^k_{in}(Q) \leq t^k_{in}(P) + PL$.* □

Proof. It suffices to consider the following two cases.

[18] Recall that basic simulation time is measured in instruction cycles.
[19] It is possible for $\mathcal{M}(P)$ to execute several passes of Algorithm CURVE with *leftend* $= k$.

1. $\rho_s(k+1)$ is on some face of Voronoi$(P, \mathcal{L})$, or

2. $\rho_s(k+1) \notin \overline{\text{Voronoi}(P, \mathcal{L})}$.

In Case 1, the first **for each** loop in Step 2 is executed and the lemma is true. In Case 2, Lemma 2 applies, and this lemma follows. □

Theorem 1 $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset \iff (CL \cdot PL \cdot k, CL \cdot PL \cdot (k+1)] \sqsubseteq REPR(P)$.

Proof. We prove each direction by an induction on k. The base cases are correct due to (1) our assumption that the particle being simulated starts at a lattice point, (2) the choice of s, and (3) the convexity of Voronoi cells (Proposition 3). The induction is carried through by invoking Proposition 6, Lemma 2, and Lemmas 6–9. □

We have the following corollary.

Corollary 1 *$REPR(P) \neq \emptyset$ iff P is in the static representation of the curve ρ_s.* □

Clearly the simulated particle travels s units of euclidean distance per $CL \cdot PL$ instruction cycles worth of time. It follows from Theorem 1 just above that the cumulative, relative error in the timing of the message arrivals that simulates the particle motion tends to zero as the particle move along. Furthermore, if we let $t(P)$ be the time at which a message reaches $\mathcal{M}_{\mathcal{L}}(P)$, then there are positive constants c and ϵ so that, for each processor $\mathcal{M}_{\mathcal{L}}(P)$ on the representation of ρ_s, $\mathcal{M}_{\mathcal{L}}(I)$, $|t(P) - c \cdot d(P, I)| < \epsilon$.

6.6 Elaboration of Steps in Algorithm CURVE

As mentioned in Section 6.4, the only step that needs elaboration is step 2. To do this, it is enough to elaborate upon:

1. performing the check $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$,

2. performing the check $\rho_s(k) \in \text{Voronoi}(P, \mathcal{L})$,

3. performing the check $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$, and

4. computing *SendDirs*.

We assume here that the computations are being performed by processor $\mathcal{M}(P)$.

Performing the check, $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$, in step 2

We assume that $\rho_s(k)$ can be computed for any $k \in \mathbf{N}$. Notice that the verification of inequalities (1) and (2), in Lemma 4, involves evaluating expressions that are rational in $\rho_s(k)$ and $\rho_s(k+1)$ when these are substituted in for x_1 and x_2, respectively.

Performing the checks $\rho_s(k) \in \text{Voronoi}(P, \mathcal{L})$ and $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$.

By Proposition 6,

$$\rho_s(k) \in \text{Voronoi}(P, \mathcal{L}) \Longleftrightarrow \forall D \in \text{Dirs}(\mathcal{L}), |((\rho_s(k) - P), D)| < \text{MinLgth}(\mathcal{L})^2/2, \text{ and}$$

$$\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})} \Longleftrightarrow \forall D \in \text{Dirs}(\mathcal{L}), |((\rho_s(k) - P), D)| \leq \text{MinLgth}(\mathcal{L})^2/2.$$

Again, notice that the verification of each inequality for a direction D involves evaluating expressions that are rational in $\rho_s(k)$.

Computing *SendDirs*

In Algorithm CURVE

$$\begin{aligned} \mathit{SendDirs} \quad := \quad & \{D \in \text{Dirs}(\mathcal{L}) \mid ((\rho_s(k) - P), D) \geq \text{MinLgth}(\mathcal{L})^2/2\}, \\ & \text{for some values of } k. \end{aligned}$$

Clearly, for each D, the evaluation of the condition for the inclusion of D in *SendDirs* involves expressions that are rational in $\rho_s(k)$.

Inspection of the expressions involved in performing the checks $\rho_s(k) \in \text{Voronoi}(P, \mathcal{L})$, and $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$ shows that, if these checks can be performed using integers only, then so can *SendDirs*.

7 Example Application: Particle Moving Along a Straight Line at Uniform Speed

Lemma 10 *Let q_1, q_2, q_3 and q_4 be rational numbers. Let $a = \sqrt{q_1} \cdot q_2$, and $b = \sqrt{q_3} \cdot q_4$, where $\sqrt{q_1}$ and $\sqrt{q_3}$ are positive square roots of q_1 and q_3, respectively. Then, the check $(a > b)$ can be performed using integers only.* □

We assume that a particle is being translated from a *Initiator* $\mathcal{M}(I)$, in $\mathcal{M}(L)$, along the direction, given by the point *Destination*, in L.

Let $\mathcal{M}(P')$ denote an arbitrary processor in L.

Convention 3 *To avoid a plethora of expressions of the form $P' - I$ and $Destination - I$ in formulas, we make the following conventions that* in this section

1. *P denotes $P' - I$ and*
2. *$Dest$ denotes $Destination - I$.*

7.1 Contents of Messages

Processor $\mathcal{M}(P')$ must receive the following in its messages.

1. the representation of the point P $(= P' - I)$.

2. the representation of the point $Dest$ ($= Destination - I$).

3. an integer value, t, in the field t_{in} such that $\texttt{CompLen} \cdot t$ is the simulation time at which $\mathcal{M}(P)$ was sent this message.

4. an integer value, l, in the field *leftend* so that the interval $[\rho_s(l), \rho_s(l+1))$ is the interval on ρ_s that the simulation is attempting to represent when $\mathcal{M}(P)$ executes step 2.

Given the values *leftend* and $(t_{in} - leftend \cdot \texttt{PathLgthDil})$, and assuming that the value of the constant `PathLgthDil` is accessible to every processor, it is possible to evaluate t_{in}. Actual implementations could use this fact to reduce the number of bits used in passing the information in items 3 and 4 above.

7.2 Choice of ρ_s

In the elaboration of step 2 of Algorithm TRANSLATE as an instance of the algorithm template CURVE, we will use the following expression for $\rho_s(k)$, and we will show that all the checks, and computations that need to be performed in step 2 are computable. Note, however, that the numerical value of $\rho_s(k)$ itself may not be rational. As previously indicated we take here $s < \text{MinLgth}(\mathcal{L})/2$, a rational number.

$$\rho_s(k) = \frac{s \cdot k}{\|Dest\|} Dest.$$

7.3 Testing: $[\rho_s(k), \rho_s(k+1)) \cap \overline{\text{Voronoi}(P, \mathcal{L})} \neq \emptyset$

This section is a massive application of Lemma 4.

The following lemma shows that the test for membership in the sets $\text{Dirs}_+(\cdot, \cdot, \mathcal{L})$ and $\text{Dirs}_0(\cdot, \cdot, \mathcal{L})$ can be performed using integer and integer vector arithmetic only. Its proof follows from the definition of the two sets from Lemma 4.

Lemma 11 *Suppose* $D \in \text{Dirs}(\mathcal{L})$. *Then*

1. $D \in \text{Dirs}_+(\rho_s(k), \rho_s(k+1), \mathcal{L}) \iff (Dest, D) > 0$, *and*

2. $D \in \text{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L}) \iff (Dest, D) = 0$.

□

For computing the maxes and mins in (1) and (2) of Lemma 4, it is enough to prove the following two lemmas. For each of the expressions in these two lemmas, it is easy to derive equivalent expressions which involve rational arithmetic only. Then an application of Lemma 10 completes the proofs of the two lemmas.

Lemma 12 *If* $D, D' \in \text{Dirs}_+(\rho_s(k), \rho_s(k+1), \mathcal{L})$, *then it is possible to perform the following comparisons using integers only.*

1. $$\frac{-\text{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > \frac{-\text{MinLgth}(\mathcal{L})^2/2 + (P, D') - (\rho_s(k), D')}{((\rho_s(k+1) - \rho_s(k)), D')},$$

2. $\dfrac{\text{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > \dfrac{\text{MinLgth}(\mathcal{L})^2/2 + (P, D') - (\rho_s(k), D')}{((\rho_s(k+1) - \rho_s(k)), D')}$,

3. $\dfrac{-\text{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > 0$, *and*

4. $\dfrac{\text{MinLgth}(\mathcal{L})^2/2 + (P, D) - (\rho_s(k), D)}{((\rho_s(k+1) - \rho_s(k)), D)} > 1$. □

Lemma 13 *If* $, D' \in \text{Dirs}_0(\rho_s(k), \rho_s(k+1), \mathcal{L})$, *then it is possible to perform the following comparisons using integers only.*

1. $((P - \rho_s(k)), D) \pm \text{MinLgth}(\mathcal{L})^2 > ((P - \rho_s(k)), D') \pm \text{MinLgth}(\mathcal{L})^2$, *and*

2. $((P - \rho_s(k)), D) \pm \text{MinLgth}(\mathcal{L})^2 > 0$. □

7.4 Tests: $\rho_s(k) \in \text{Voronoi}(P, \mathcal{L})$ and $\rho_s(k) \in \overline{\text{Voronoi}(P, \mathcal{L})}$.

It is easy to show that the condition $|((\rho_s(k) - P), D)| \leq \text{MinLgth}(\mathcal{L})^2/2$ can be checked, using integers only, for each direction D in $\text{Dirs}(\mathcal{L})$.

8 Related Work

In [CRS90a, CRS91a], we note that the accuracy of our approach depends on the choice of the discretization of euclidean n-space used to base mesh computers on, as well as suitable algorithms, in particular their timing, that could be run on such mesh computers to simulate physical phenomena. It should be noted that such suitable algorithms are an important, integral part of this approach. In [CRS91b] we present an algorithm for simulating uniform radiation from a point source in bounded mesh computers based on lattices in general, and more "natural" algorithms in bounded mesh computers based on the lattices A_n and $\mathbf{Z}^n$, respectively. In this paper, we discussed mesh computers based on well-represented lattices (see Section 3 above.). We also presented Algorithm template CURVE for the simulation of constant speed particle motion in a mesh computer based on a well-represented lattice, and then, in Algorithm TRANSLATE, gave an instance of CURVE for the simulation of constant speed particle translation in such mesh computers. In [She91] we present algorithms for the simulation of constant angular speed particle revolution. It turns out that the class of well-represented lattices intersects with the class of *root* lattices (see [CS88] for a discussion on root lattices). Several characterizations of root lattices are given in [RS91]. Interestingly, it also turns out that a direct consequence of one of these characterizations of root lattices is that the "natural" algorithms, mentioned just above, for uniform radiation from a point source can be generalized to work on a bounded mesh computer based on any root lattice.

References

[Aur90] Franz 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.

[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 [CRS91a].

[CRS90b] John Case, Dayanand S. Rajan, and Anil M. Shende. Parallel processor computer, Dec 1990. U.S. Patent Application submitted by SUNY, Research Foundation.

[CRS91a] John Case, Dayanand S. Rajan, and Anil M. Shende. Lattice computers for optimally representing euclidean space. Technical Report 91-15, University of Delaware, June 1991. Expands on and corrects slightly [CRS90a].

[CRS91b] John Case, Dayanand S. Rajan, and Anil M. Shende. Spherical wave front generation in lattice computers. Technical Report 91-16, University of Delaware, June 1991.

[CS88] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer Verlag, 1988.

[Fey82] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[FHP86] U. Frish, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier Stokes equation. *Physical Review Letters*, 56(14):1505–1508, April 1986.

[FT82] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4), 1982.

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

[Has87] Brosl Hasslacher. Discrete fluids. *Los Alamos Science*, (15):175–217, 1987. Special Issue.

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

[Mac86] Thinking Machines. Introduction to data level parallelism. Technical Report 86.14, Thinking Machines, April 1986.

[Mar84] Norman Margolus. Physics–like models of computation. *Physica 10D*, pages 81–95, 1984.

[Min82] Marvin Minsky. Cellular vacuum. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[RS91] Dayanand S. Rajan and Anil M. Shende. Efficiently generated lattices and root lattices. Technical Report 92-02, University of Delaware, August 1991. Journal article under review.

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

[Svo86] Karl Svozil. Are quantum fields cellular automata? *Physics Letters A*, 119(4), December 1986.

[SW86] James B. Salem and S. Wolfram. Thermodynamics and hydrodynamics with cellular automata. In S. Wolfram, editor, *Theory and Applications of Cellular Automata.* World Scientific, 1986.

[TM87] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines.* MIT Press, 1987.

[Tof77a] Tommaso Toffoli. Cellular automata machines. Technical Report 208, Comp. Comm. Sci. Dept., University of Michigan, 1977.

[Tof77b] Tommaso Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer and System Sciences*, 15:213–231, 1977.

[Tof84] Tommaso Toffoli. CAM: A high–performance cellular–automaton machine. *Physica 10D*, pages 195–204, 1984.

[Vic84] Gérard Y. Vichniac. Simulating physics with cellular automata. *Physica 10D*, pages 96–116, 1984.

[Wol83] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983.