Notes on Amorphous Computing

Jacob Katzenelson[1]
AI Lab, MIT

Abstract

An amorphous computer is a multitude of small/tiny computers each with a CPU, memory and communication means. Physically these computers are similar to sand particles and they occupy a finite compact domain D in one, two, or three dimensional Eucledian space, depending on the application.

This note presents a computational model for an amorphous computer and explores its use for numerical computing, in particular, for the simulation of partial differential equations.

Integration formulas for partial differential equations of systems that satisfy conservation laws are derived. It is shown that, under appropriate conditions, the formulas provide consistent and stable and, therefore, convergent approximations to the problems on hand. In other words, one can perform integration of PDEs using the amorphous computer.

The communication between the computers is investigated and it is shown that starting with asynchronous communication one can derive necessary information needed to switch over to synchronous communication.

Key words: Amorphous computers, amorphous computer model – computation and communication, integration of PDEs on amorphous computers, integration with Voronoi diagram partition of the domain.

Contents

[1]On leave from the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel.

1 Introduction

The purpose of this note is to explore and communicate ideas related to the use of amorphous computers in numerical computing and, in particular, the simulation of partial differential equations (PDE in the sequel). For background on amorphous computers see [1] [2] and [3].

The amorphous computing model is described in the next section. Section 3 discusses finding the location of an amorphous particle computer (*particle*) from the locations of the boundary apcs. Section 4 considers the simulation of partial differential equations (PDE) of conservative laws. Section 5 discusses alternative communication schemes. The last section provides a summary and conclusions.

2 The amorphous computing model

The amorphous computer model consists of a multitude of small/tiny computers each with a CPU, memory and communication means. Physically they are similar to sand particles and we refer to them as *amorphous particle computers*, or **particle**s. The particles occupy a finite compact domain D in one, two, or three dimensional Euclidean space, depending on the application. The distribution of particles in space is random. Particles do not touch each other and their average density (number of particles per unit volume) is given. We distinguish between those particles that are on the boundary, *boundary particles*, and those in the interior, *interior particles*.

All particles are "born" with the same program resident in their memory and, as *power* is turned on, they start to execute it. Particles start listening to messages and, as a message arrives at a particle, the particle branches according to the content of the message. Particles that are at the boundary of the application domain D, *boundary particles*, are different in that *each* can be assigned individual initial values and that one of these values is a special initial message, "*thou art a boundary particle*", sent to itself as soon as power is turned on which makes it execute an initial program.

The particles communicate via broadcast messages. A message is composed of digital information and can be of any length but we would prefer it to be reasonably short. The communication mechanism is described in [3]. Basically, a message can only be heard within a radius r of its source; when a message is transmitted it occupies, TDMA-like, the entire channel; messages may collide; collision can be detected by the receivers only; a protocol, called *random-delay-broadcast*, is used to reduce the effect of the collisions.

As a result of the collisions, a message sent is not guaranteed to arrive and there is a finite, hopefully small, probability of losing messages. This probability can be reduced by changing parameters of the communication protocol, e.g., repeating a message several times.

The *random-delay-broadcast* introduces *time* into the model explicitly. When a processor has to communicate, it delays the message by a delay chosen at random (equally distributed) over a time-interval τ. τ is taken proportional to ρ_{max}, the average number of particles reachable from one processor. Thus, even if computing time and message passing time are assumed to be negligible, the expected message delay (for repeating the message three times) is $3\frac{\tau}{2}$ [3].

The particles are clocked machines with about the same clock frequency. The clocks are not synchronized but they all can reliably and individually calculate the same time intervals, T, satisfying, say,

$$T > \langle \textit{expected message delay} \rangle$$

provided T is a large multiple of the clock period.

The exact location of the interior particles is not known and it is assumed they are randomly distributed in the domain D and their expected density, i.e. expected number of particles per unit volume, is known. The expected density is denoted by I.

3 Locations

This section describes algorithms for determining the location of the interior particles given the locations of the boundary particles. For simplicity a $2D$ domain is discussed; the extensions to $1D$ and $3D$ are obvious. The distribution of the particles in space was assumed random. In this section an arbitrary *instance* of the distribution is analyzed. It is assumed, however, that the particle distribution is dense, i.e., there are many particles within a sphere of radius r or, the probability that such a sphere contains only a few particles in negligible.

A. *A infinite strip in the half plane domain:*
The algorithm is explained with the help of figure 1. When power is turned on, each boundary particle transmits a message with its x and y coordinates. These messages are heard/received by all particles within distance r from the boundary. Each of these processors gets between one and $2 \times r \times \sqrt{I}$ messages, where I is the maximum number of particle's per unit area; at most one message is received by a particle that is r away from the boundary and $2 \times r \times \sqrt{I}$ messages – by a particle that is at the boundary. The processor obtains its y coordinate by averaging the y coordinates of the messages it receives [2].

Those particles that received only one message form a new boundary, r away from the original one. After getting its y coordinate, each of the new boundary particles awaits time τ and transmits a message with its own coordinates. Thus, particles whose distance from the boundary is less or equal to $2r$ get their y coordinates.

The process repeats until a given depth of the strip is covered.

B. *A square domain*
Consider figure 2. By applying the above algorithm to the x edge and to

[2] The y coordinate can be also found by averaging the coordinate of the two farthest away boundary particles.

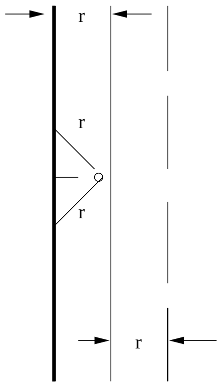

Figure 1: Location in a strip in an infinite half plane.

the y edge, we establish the coordinates of all particles within the *internal* square.

The values of the coordinates in the four corner squares have to be found somewhat differently. They are the only ones that receive both x and y coordinates and the coordinates of the corner particle which are labeled as the corner coordinates.

The particles of each corner are divided into two sets: (a) particles whose distance from the corner is large or equal to r; (b) particles whose distance from the corner is smaller than r. A particle is a member of (b) if it receives the corner coordinates; otherwise it is a member of (a). (a) particles are 'ordinary' particles; they find their x (y) coordinates by averaging the x (y) coordinates of the boundary they received, i.e., they are no different from particles along the strip of distance r from the boundary. (b) particles are different and have to solve the following equation to determine their coordinates. Let (x_1, y_1) be the coordinate of a particle in (b). Let x_{max} (y_{max}) be the x (y) border coordinate farthest from the corner the particle received. (x_1, y_1) are determined by solving the following equations:

$$(x_{max} - x_1)^2 + y_1^2 = r^2,$$

$$x_1^2 + (y_{max} - y_1)^2 = r^2.$$

C. *An arbitrary domain in the plane – method I.*
By adding particles complete the domain to a square. Apply B above to find the coordinates.

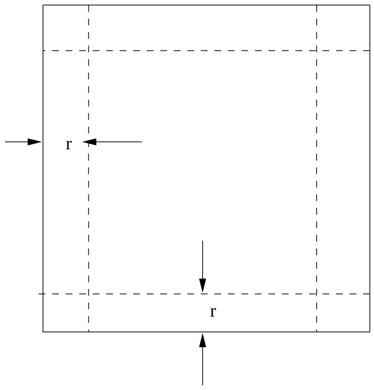

Figure 2: Location in a square.

D. *An arbitrary domain in the plane – method II.*
Our goal is to determine the distance of an particle from another particle which we call a *source.* The claim is that, under certain conditions, if all the particles in a neighborhood $\frac{1}{2}r$ around the source transmit their respective locations, the listening particle can determine its location to an accuracy that is the average distance between two adjacent particles. Since this is, by in large, a geometric problem, the word *point* is used for particle.

D.1: *The 1D case*
Consider a single source that transmits its coordinates. The message also contains a counter that is zero for the source. The source transmits one message.

Each point that receives messages does the following:

- The first message from any source is recorded and is repeated (re-transmitted) with the count increased by one.
- All further messages from the same source are treated as follows: If the message count is smaller than the recorded message, the recorded message is changed and a lower count correction message is issued. If the message count is equal or larger than the recorded count – the message is ignored.

Figure 3 illustrates the intervals covered with the counter equal to 0, 1 (1 hop), 2 (2 hops), etc. Clearly each point, knowing the value of the counter

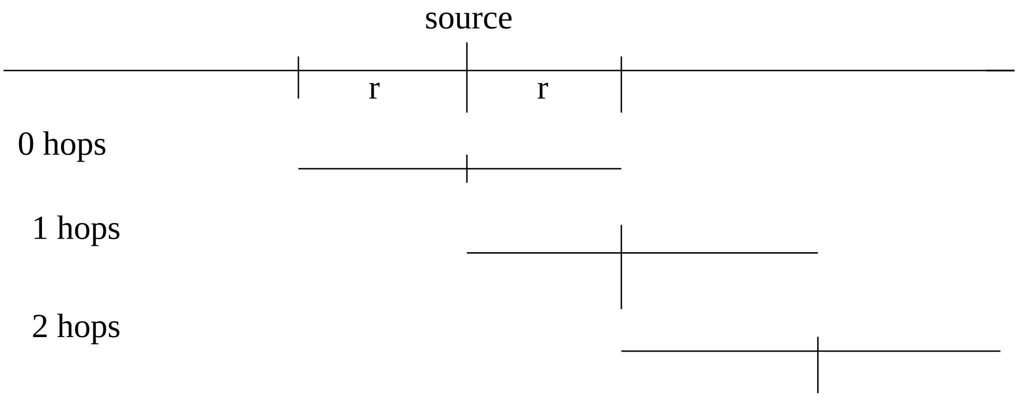

Figure 3: $1D$: Distance from a single source.

n, can establish its distance d from the source

$$n \times r \leq |d| \leq (n+1) \times r.$$

Now, add another source point, say to the left of the original one and distance $\frac{1}{2}r$ from it. Let this point transmit its coordinate and all points repeat what they received while increasing the count as above. Clearly, the second point provides an additional relation

$$n_1 \times r \leq |d_1| \leq (n_1+1) \times r.$$

where d_1 is the distance from the second point. For example, the point that needs its location measured is of actual distance 2.3 from the first point; knowing that $n = 2$ and $n_1 = 2$ establishes $2 \leq |d| \leq 2.5$. Notice that a third point is required to find the sign of —d— in case $n = n_1$.

By further increasing the number of sources, ultimately providing a "continuum"of source points over a interval of length $\frac{1}{2}r$, the accuracy can be increased arbitrarily, or to the distance between adjacent sources.

D.2: *The 2D case*

Figure 4 depicts a source and a listening/target point in $2D$. All points repeat messages as they do in the $1D$ case. Clearly, as in the $1D$ case, the listener can determine its distance from the source to an accuracy of

$$n \times r \leq |d| \leq (n+1) \times r.$$

Figure 4: $2D$: Distance from a single source.

If the length of AB is $\frac{1}{2}r$ and, if all points on the segment AB have their locations (figure 4) and transmit it as sources, a point on the line AB can calculate its distance from A to an accuracy of the maximum distance between two adjacent particles on AB. Therefore, if all points within the circle of radius $1/2r$ have their locations and transmit it as sources, a listening point plots all positions with the number of hops each position has, and the boundary between n and $n+1$ hops defines the distance with respect to each source. The actual distance from the origin can be determined to within the distance between adjacent particles.

Concluding method II: If we assume that we can form any (finite) volume of amorphous computers and then move it physically about and attach it to other volumes, the following interesting conclusions are obtained:

To obtain a circle with all points inside it having their distance, one has to take a square with the circle inside it (not touching the corners). Assign initial conditions to the edges and propogate the values from the edges to the circle. Next, the circle is moved and attached to an arbitrary region to provide one reference point from which one start assigning locations to the particles of the new region.

Chirp RADAR analog: Clearly, the accuracy of finding the location depends on r and r depends on the strength or, power, of the transmitter signal. If the signal strength can be modulated by command so that the radius of receiving the signal is changed from r to $\frac{1}{2}r$, $\frac{1}{4}r$, $\frac{1}{8}r$, etc. The accuracy resulting from a series of transmissions, starting with strength r and continuing with strength $\frac{1}{2}r$, etc. will go up until reaching the average space between particles (There is no need for further increase as the signal will not be carried if the power is too low.).

Accuracy: A question often asked[3] is “How accurate are the methods A

[3] Hal Abelson

to D?" A formal answer can be given in case A above: Let a point in the strip establish its x coordinate, x_p by averaging the x coordinates of n points on the boundary. Assume each boundary point x_i is a Gaussian random variable with an average $\overline{x_i}$ and a standard deviation σ_x, where σ_x is the same for all boundary points. Then,

$$\overline{x_p} = \frac{1}{n}\Sigma_{i=1}^{n}\overline{x_i}$$

and

$$\sigma_{x_p}^2 = \frac{\sigma_x^2}{n}.$$

A more profound result is the following. Unfortunately, the full meaning of the result relates to the integration of differential equations of the next section, section 4. Assume that each point X_i has an error in its location so that its coordinates (in $2D$ for simplicity) are $(x_i + \delta x_i, y_i + \delta y_i)$ rather than (x_i, y_i). If the deviation is such that both the point and its near-neighbors are well within the communication region determined by r, the deviation does not matter much; i.e., the same differential equation is solved by a set of points whose locations are slightly different from the location of the particles that perform the integration. Since all points use their computed locations and all constructions are done with respect to the computed locations, the results correspond to a set of particles whose locations are the computed locations rather than the actual ones.

4 Differential Equations of Conservation Laws

This section discusses the simulation of differential equations of systems that satisfy conservation laws. We expect that its analysis will result in insight to the handling of time and space in amorphous computers. Of similar interest is the issue of parallel computation while maintaining conservation.

Terminology and notations in this section follow [7].

Preliminaries: Once again we have some (general) domain D populated by particles. A (vector valued) function u is defined over the domain. This function denotes the *density* of a certain substance that is contained in the domain. A conservation law asserts that the rate of change of the substance in a fixed domain G is equal to the $flux$, f, of the substance across the boundary of G (∂G in the sequel). This is expressed as

$$\frac{d}{dt}\int_G u dx = -\int_{\partial G} f \cdot \mathbf{n} ds \tag{1}$$

where $\mathbf{n}$ is a outward normal unit vector to ∂G and ds – a surface element on ∂G. If G contains sources of u, those sources are added to the conservation law leading to

$$\frac{d}{dt}\int_G u dx = -\int_{\partial G} f \cdot \mathbf{n} ds + \int_G S dx$$

where S is the source at a point in G. For simplicity, the sources are ignored for the time being.

The above leads to (see [7])

$$\int_G (u_t + \mathrm{div} f) dx = 0,$$

and, by shrinking G, to

$$u_t + \mathrm{div} f = 0,$$

which is the differential form of the conservation law.

An application requires a relation between f and u. Usually, f is made a function of u. In the case that $f = a \cdot \mathrm{grad}\, u$, a – a scalar, this results in

$$u_t + a\nabla^2 u = 0.$$

With $a < 0$ (the flow is directed *away* from high concentration of u) this is the heat equation [12] or the diffusion equation.

4.1 The amorphous computing scheme

In the amorphous computer case each particle is associated with a point in space and carries the value of the density of the substance u at this point. We denote by u_i the quantity associated with the i-th particle. If G has only one particle, the conservation law becomes

$$\frac{d}{dt}\int_G u_i dv + \int_{\partial G} f \cdot \mathbf{n} ds = 0.$$

The relation between f and u has to be specified before attempting to solve the equation. Also the domain D occupied by the particles has to be partitioned to regions each containing a single particle. One such partition is called *Voronoi diagram*[15, 16, 17, 18].

Given points S in D, D may be $2D$ or $3D$ etc., the Voronoi region of a point x_0 is given by

$$\{x : x \in D, \|x - x_0\| < \|x - x_i\|, x_i \in S, x_i \neq x_0\}.$$

In $2D$ the boundaries of such a region are linear segments, see figure 6. In $3D$ the boundaries are sections of planes. The Voronoi diagram is the (diagram of the) boundaries of the Voronoi regions.

Two regions with a common border are called adjacent. A graph formed by connecting each point in S with all other points in S that have adjacent regions is called the *Delaunay graph* or diagram. Figure 5 illustrates the Voronoi and the Delaunay diagrams. The *near neighbors* of a point A are the points connected to A by one Delaunay arc.

Given a point x_0 and a near-neighbor x_1 there is a part of the boundary of the region of x_0 that can be associated with the Delauny branch x_0x_1; this boundary part is the section whose points are equal distance from x_0 and x_1. The boundary part is a linear segment in $2D$; in $3D$ it is a convex polygon in a plane. Depending on the distribution of the near-neighbors, the Delaunay arc x_0x_1 may or may not intersects its associated part of x_0's boundary. E.g., in figure 6 the arc U_0U_i intersects the boundary segment ab; figure 5 has several arcs that do not intersect their associated boundary segments. If each Delauny arc intersects its associated boundary part we call the distribution of the particles a *regular* distribution.

Note that the Voronoi diagram forms a partition that covers the $2D$ or $3D$ space with no overlap, and the flux leaving one particle region enters the regions of the particle's near- neighbors.

These diagrams have attracted a fair amount of attention; for our purpose, however, each particle needs only the locations of its near-neighbors. The appendix describes an algorithm in which each partiicle in a $2D$ system finds its near-neighbors. In the following it is assumed that each particle has the locations of its near-neighbors and that the distribution of particles is regular.

Assume that the flux f is related to u by

$$f = a \text{ grad } u \tag{2}$$

for some scalar a. Let N be the set of neighbors of u_0 and u_i – the density of the conserved quantity at the i-th member of N, and h_i the vector distance

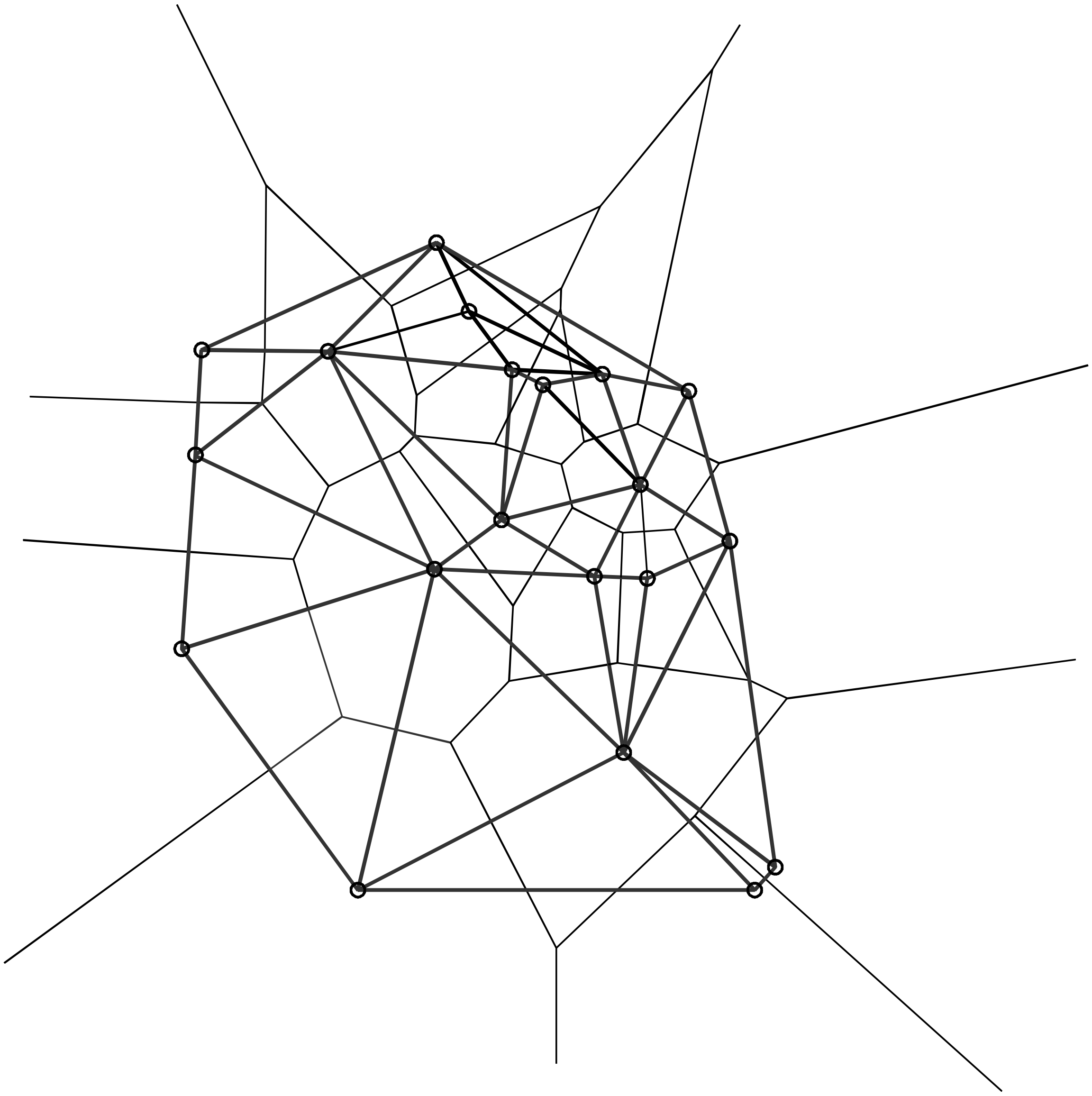

Figure 5: Voronoi and Delaunay diagrams.

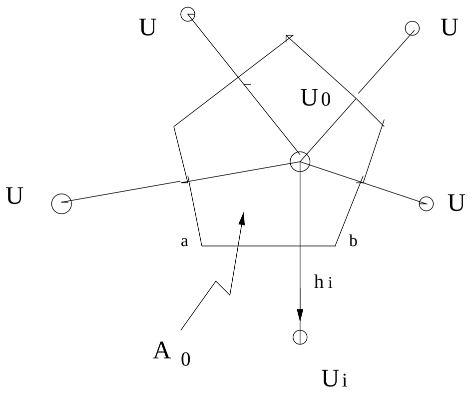

Figure 6: The area associated with an particle.

between the 0-th particle and its i-th neighbor. Taylor expansion (or, the mean value theorem) results in

$$u_i - u_0 = h_i.\text{grad } u + O(|h_i|^2)$$

where grad u is evaluated somewhere on the line joining u_0 and u_i.

Thus, since h_i is normal to its section of ∂G,

$$\begin{aligned} \int_{\partial G} f \cdot \mathbf{n} ds &= \sum_{i \in N} \int_{\partial G} f \cdot \frac{h_i}{|h_i|} \cdot ds_i & (3) \\ &= \sum_{i \in N} a \int_{\partial G} (\text{grad } u) \cdot \frac{h_i}{|h_i|} \cdot ds_i & (4) \end{aligned}$$

If grad u is evaluated at the intersection of h_i and the boundary and it is taken to be a reasonable approximation of the value of grad u on the straight boundary section intersecting h_i,

$$\begin{aligned} \int_{\partial G} f \cdot \mathbf{n} ds &= \sum_{i \in N} a(u_i - u_0) \frac{1}{|h_i|} \int_{\partial G_i} ds_i & (5) \\ &= \sum_{i \in N} a \cdot c_{0i} \cdot (u_i - u_0), & (6) \end{aligned}$$

where

$$c_{0i} = \frac{1}{|h_i|} \int_{\partial G_i} ds_i.$$

It is easy to show that $c_{0i} = c_{i0}$ which implies that the approximating method is conservative, i.e., the total amount of the conserved quantity sent from, say, the 0-th particle to the i-th particle is obtained by the i-th particle.

If u does not change too fast inside the area alloted to the 0-th particle, the following approximation can be done

$$\int_G u_0 dv \approx A_0 u_0.$$

Thus,

$$\frac{\Delta u_0}{\Delta t} = -\frac{a}{A_0} \sum_{i \in N} c_{0i}(u_i - u_0). \tag{7}$$

Note that the coefficients c_{0i} represents the irregularities of the locations of the particles. When the particles are placed on a regular grid, say with $n = 4$ or $n = 6$[4], all the c_{0i}s are equal and can be moved to the front of the summation.

Equation 7 is a major intermediate step. The equation is treated as if the left-hand side is $\frac{du_0}{dt}$ and it can be integrated using the usual integration formulas.

Let U_j^n denote the calculated (approximated) solution at the j-th particle at time $n\Delta t$. The forward Euler formula yields

$$U_0^{n+1} = U_0^n - \frac{a\Delta t}{A_0} \sum_{i \in N} c_{0i}(U_i^n - U_0^n). \tag{8}$$

The backward Euler yields

$$U_0^{n+1} = U_0^n - \frac{a\Delta t}{A_0} \sum_{i \in N} c_{0i}(U_i^{n+1} - U_0^{n+1}) \tag{9}$$

where, as usual, some algorithm has to be provided to solve this implicit equation. By using past values of U_0 higher-order formulas can be constructed similarly.

[4]I believe these are the only regular distribution in two space; $n = 3$ forms a regular distribution which is not dense.

4.2 Consistency, convergence and stability

Preliminaries: Our problem is defined as follows [8]: Let G be a compact region in one, two or three dimensional space with the boundary ∂G. The required solution is a function u defined on $G \times [0, t_f]$; u might be vector valued but, for simplicity, we consider it as scalar. The operator $L(.)$ involves u and its spatial derivatives but is independent of the time t. $g(.)$ is an operator that defines the values of u on ∂G. The initial conditions prescribe the value of u for $t = 0$ over the region G.

We are seeking a function u satisfying

$$\frac{\partial u}{\partial t} = L(u) \; in \; G \times (0, t_f], \tag{10}$$

$$g(u) = g_0 \; on \; \partial G, \tag{11}$$

$$u = u^0 \; on \; G \; when \; t = 0. \tag{12}$$

Let U_j^n denote the calculated (approximated) solution at the j-th particle at time $n\Delta t$. Let J denote the set of particles and

$$U^n = \{U_j^n, \; j \in J\}.$$

The actual value of the function u at the j-th particle and at time n is denoted by u_j^n and

$$u^n = \{u_j^n, \; j \in J\}.$$

Given an algorithm that computes U^{n+1} from U^n we are interested in the behavior of U^n as the number of particles in G is increased, or the average distance between neighboring particles, h, as well as Δt, go to zero while keeping $n\Delta t$ a constant in the time interval of interest. The way these limits are approached is called the *refinement path* and taking this limits is denoted by $\Delta t(h) \to 0$.

Norms are defined over the above quantities, for example,

$$||U^n||^2 = \{\sum_{j \in J} A_j |U_j^n|^2\}.$$

The multiplication by A_j, the area in $2D$ (or volume in $3D$) associated with the j-th point, is needed for evaluating the norm along the refinement path.

Assume (without too much lose of generality) that our algorithm or integration method, has the form

$$U^{n+1} = BU^n + F^n.$$

The *truncation error* is defined in terms of the exact solution u^n

$$T^n = u^{n+1} - [Bu^n + F^n].$$

An algorithm or an integration method is said to be *consistent* if

$$T_j^n \to 0 \;\; as \; \Delta t(h) \to 0 \;\; \forall j \in J.$$

An integration method provides a *convergent approximation* if

$$\|U^n - u^n\| \to 0 \;\; as \; \Delta t(h) \to 0, n\Delta t \to t \in (0, t_f]$$

for all u^0 for which exists a unique $u(x,t)$ on $G \times (0, t_f]$.

Let V^n and W^n be two approximated solutions derived by the same integration method for the same problem but for different initial condition. The integration method is said to be *stable* for a given norm and a refinement path if there exists a constant K such that

$$\|V^n - W^n\| \leq K\|V^0 - W^0\|, \;\; n\Delta t \leq t_f,$$

where K is independent of V^0, W^0 and of $\Delta t(h)$ on the given refinement path.

We are basically interested in the convergence property of our integration method. A theorem by P. D. Lax [9] considers the approximation of the solution to equations 10,11,12 by a linear integration formula. The theorm states that given a consistent integration method, the method is convergent if and only if it is stable.

In the sequel we shall show that the amorphous computing method of equation 8 provides a consistent and stable approximation to the solution of the conservation equation 1 with $f = a \cdot \text{grad}\, u$. It is, therefore, a convergent approximation.

The amorphous computer problem: Our conservation problem is a special case of the problem stated by equations 10, 11 and 12. For clarity, the *conservation problem* is re-stated as follows:

$$\begin{aligned}
\frac{d}{dt}\int_G u dx &= -\int_{\partial G} f \cdot \mathbf{n} ds \\
with\ f &= a \cdot \mathrm{grad}\, u, \quad a < 0; in\ G \times (0, t_f], && (13)\\
u &= u_{boundary}(t)\ on\ \partial G, && (14)\\
u &= u^0\ on\ G\ when\ t = 0. && (15)
\end{aligned}$$

The amorphous computing (forward Euler or backward Euler) method (set-up) for solving the above problem consists of the following parts:

- Particles populate the region G; ∂G passes through particles called the boundary particles. The distribution of the particles is regular.
- Each particle knows whether it is a boundary or an interior particle and has the following quantities available to it:
  - time, or rather the initial time and Δt;
  - the particle near-neighbors and their locations;
  - boundary particles have their corresponding $u_{boundary}(t)$; interior particle have their corresponding u^0.
- For $n = 1, 2, ..., n\Delta t \in (0, t_f]$, interior particle evaluates U^{n+1} according to equation 8 (forward Euler) or equation 9 (backward Euler).

Theorem:
Given a regular amorphous computer distribution there exists a $\Delta t > 0$ *for which the forward Euler method provides a consistent, stable and, therefore, convergent approximation to the conservation problem.*

Theorem: *Given a regular amorphous computer distribution the Backward Euler method provides a consistent, stable and, therefore, convergent approximation to the conservation problem.*

Proof: In the following stability is proved first followed by consistency. Since our conservation problem is a special case of the problem defined by 10, 11 and 12, convergence follows from Lax theorm.

Stability: Consider the definition of stability above.

$$U^n = V^n - W^n$$

is a solution corresponding to a null inhomogeneous term F_n and initial conditions which are the difference between those of V^n and U^n. It is sufficient to prove that

$$\|U^n\| \leq K\|U^0\|, \; n\Delta t \leq t_f,$$

Or,

$$\|B^n\| \leq K \; n\Delta t \leq t_f.$$

In the sequel we show that B is real and symetric, and therefore, diagonalizable (Theorm 8.11 in [10]). It is also shown that the eigenvalues of B are smaller than 1 from which follows the above bound on $\|B^n\|$.

For the forward Euler (equation 8) B has the following form:

$$b_{jj} = 1 + \frac{a\Delta t}{A_j} \sum_{i \in N_j} c_{ji}$$

$$b_{ji} = -\frac{a\Delta t}{A_j} c_{ji}$$

when i is a near-neighbor of j and

$$b_{ji} = 0$$

otherwise.

It follows from Greshgorian theorem ([10] page 341) that all eigenvalues of B are within the circles whose centers are b_{jj} and radii $\sum_{i,i \neq j} |b_{ji}|$. In our case, $a < 0$, this implies that all eigenvalues of B satisfy

$$|\lambda_i| \leq 1 \;\; if, \; for \; all \; j, \; \frac{|a|\Delta t}{A_j} \sum_{i \in N_j} c_{ji} \leq 1.$$

The above is guaranteed if

$$\Delta t \leq \min_j \frac{A_j}{|a| \sum_{i \in N_j} c_{ji}}, \tag{16}$$

that looks very much like the Courant-Friedrichs-Levy (CFL) stability criterion [20].

Note that this condition determines a refinement path requiring $\Delta t(h) \rightarrow 0$ in a way that $\frac{\Delta t}{A_j}$ satisfies equation 16.

For the backward Euler scheme, equation 9, U^n is expressed as

$$B^1 U^{n+1} = U^n$$

where

$$b_{jj}^1 = 1 - \frac{a\Delta t}{A_j} \sum_{i \in N_j} c_{ji},$$

$$b_{ji}^1 = \frac{a\Delta t}{A_j} c_{ji}$$

when i is a neighbor of j and 0 otherwise.

Here we are interested in the eigenvalues of $[B^1]^{-1}$. Clearly, B^1 is positive definite, and has real eigenvalues. From the Greshgorian theorem follows that all eigenvalues are within the circles whose centers are b_{jj}^1 and radii $\sum_i |b_{ji}^1|$. In our case, $a < 0$, $b_{jj}^1 \geq 1$ and $b_{jj}^1 + \sum_i b_{ji}^1 \geq 1$. Therefore, the eigenvalues of B^1, λ_i^1 satisfy $\|\lambda_i^1\| \geq 1$ and the backward Euler is stable for any (positive) Δt.

Thus, the forward Euler is stable if condition 16 is satisfied. The backward Euler is stable (as expected). The refinement should be done in a way that the corresponding conditions are satisfied.

Consistency: Consistency is shown under the assumption that the solution is *smooth* and that the refinement path imposes some relation between h and Δt. The smoothness of the solution means that (higher) derivatives exist and are finite (see, for example, [11]).

Consider the forward Euler formula (equation 8). Let u^n be the exact solution. The truncation error for some particle denoted by 0 is

$$T_0^n = u_0^{n+1} - u_0^n + \frac{a\Delta t}{A_0} \sum_{i \in N} c_{0i}(u_i^n - u_0^n).$$

To find the value of this expression when $\Delta t(h) \to 0$ one steps back through the steps by which equation 8 was developed, remembering that u^n satisfies the differential equation. Thus,

$$T_0^n = \frac{\Delta t}{A_0}\left(\frac{A_0(u_0^{n+1} - u_0^n)}{\Delta t} + a \sum_{i \in N} c_{0i}(u_i^n - u_0^n)\right).$$

Adding and subtracting $\int_{\partial G} f.\mathbf{n}ds$ and remembering that u is a solution of the original equation results in

$$
\begin{aligned}
T_0^n &= \frac{\Delta t}{A_0}(-\int_{\partial G} f.\mathbf{n}ds + a\sum_{i\in N} c_{0i}(u_i^n - u_0^n)) && (17)\\
&= \frac{\Delta t}{A_0}(-\int_{\partial G} f.\mathbf{n}ds + a\sum_{i\in N} c_{0i}(h_i.\text{grad } u_o + \frac{1}{2}D^2u(z_i; p_i - p_0))) && (18)
\end{aligned}
$$

where $D^2u(z_i; p_i - p_0)$ denotes the 2ed differential of u evaluated at a point z_i between the coordinates of the 0-th and i-th particle. Thus,

$$
\begin{aligned}
T_0^n &= \frac{\Delta t}{A_0}\frac{1}{2}D^2u(z_i; p_i - p_0) && (19)\\
&= \frac{\Delta t}{A_0}O(h^2) && (20)\\
&= O(\Delta t). && (21)
\end{aligned}
$$

Therefore, the forward Euler method is consistent. A similar proof can be constructed for the backward Euler.

Thus, it follows from the Lax theorem that the methods are also convergent. QED.

Consider the role that the coefficients c_{ij} play in the proof. The surprising result is that if all the coefficients are taken as equal to 1, and the integration formula 8 is taken to be

$$U_0^{n+1} = U_0^n - \frac{a\Delta t}{A_0}\sum_{i\in N}(U_i^n - U_0^n), \quad (22)$$

The method is still consistent and stable and therefore convergent. The method of equation 22 can be derived from the conservation laws by taking rougher approximations as the integral of the flux over the surface is replaced by the sum. One expects the method of equation 22 to be less accurate and it is somewhat surprising that it is 'well behaved' in the sense that it conserves the quantity u and the flux.

Corollary: *(a) If Δt is small enough, the amorphous computing forward Euler method with $c_{ij} = 1$ (equation 22) provides a consistent, stable and, therefore, a convergent approximation to the conservation problem. (b)The amorphous computing backward Euler method with $c_{ij} = 1$ (equation 9) provides a consistent, stable and, therefore, a convergent approximation to the conservation problem.*

4.3 Wave equations and conservation laws

The purpose of this section is to show that the amorphous computer can integrate not only the diffusion equation, as illustrated in section 4.1 and 4.2, but also other types of equations derived from conservation laws, in particular, wave equations.

Consider a conservation law in its classical form, i.e.,

$$\frac{d}{dt}\int_G u dx = -\int_{\partial G} f \cdot \mathbf{n} ds, \tag{23}$$

however, this time let the flux function be related to f by

$$f(u,x,t) = au \tag{24}$$

where a is a scalar or an appropriate matrix depending on the nature of f and u. This assumption is different than the one we made at the beginning of the section, i.e., $f = a \cdot \text{grad } u$, and which led to the diffusion equation.

Assume all functions involved are smooth enough to write the differential form of equation 23, then

$$\int_G (u_t + \text{div } au) dx = 0,$$

which implies

$$u_t + \text{div } au = 0.$$

In the one-dimensional case, if a is constant, we get

$$u_t + a\frac{\partial u}{\partial x} = 0. \tag{25}$$

This is the linear *advection* or *one-way wave equation* [11].

Amorphous computing integration formula: Consider again equations 23 and 24; let the domain be two dimensional and let the domain be partitioned as depict in figure 6. Equations 23 and 24 led to

$$\frac{d}{dt}\int_G u dx = -\int_{\partial G} au \cdot \mathbf{n} ds.$$

Approximating the left-hand side by

$$\frac{d}{dt}\int_G u dx \approx A_0 \frac{\Delta u}{\Delta t},$$

and the value of u on ∂G by the midpoint values between two neighbors

$$\int_{\partial G} au \cdot \mathbf{n} ds \approx \sum_{i \in N_0} a \cdot c_{0i} \cdot \frac{(u_i + u_0)}{2}$$

results in

$$\frac{\Delta u}{\Delta t} = \frac{a}{A_0} \sum_{i \in N_0} c_{0i} \cdot \frac{(u_i + u_0)}{2}. \tag{26}$$

It is interesting to compare this equation with equation 7 obtained for the diffusion equation and note the small but significant difference.

Actual integration formulas can be derive from equation 26 in a way similar to the way they were derived from equation 7. Consistency, stability and convergence can be investigated similarly as well.

Maxwell equations: In the following we first write the usual differential form of Maxwell equations; next, we derive conservation laws from these equations and, after that, we apply these laws to a transmission line operating under the quasi static assumptions.

Maxwell laws in differential form are:

$$\text{div } \mathbf{D} = \rho \tag{27}$$

$$\text{curl } \mathbf{H} = \mathbf{j} + \frac{\partial \mathbf{D}}{\partial t} \tag{28}$$

$$\text{curl } \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{29}$$

$$\text{div } \mathbf{B} = 0. \tag{30}$$

From 27 and 28 it is concluded that

$$\text{div } \mathbf{j} + \frac{\partial \rho}{\partial t} = 0. \tag{31}$$

Let S be a surface in E^3 with a boundary l; let S be fixed in time. From the above the following conservation forms can be readily derived :

$$\int_S \text{curl } \mathbf{E} \cdot \mathbf{ds} = \int_l \mathbf{E} \cdot \mathbf{dl} = -\int_S \frac{\partial \mathbf{B}}{\partial t} \cdot \mathbf{ds} = -\frac{d\phi}{dt}.$$

Let G be some volume with surface ∂G. From 31 follows:

$$\int_G \text{div } \mathbf{j} dG = \int_{\partial G} \mathbf{j} \cdot \mathbf{ds} = -\int_G \frac{\partial \rho}{\partial t} dG = -\frac{dQ}{dt}.$$

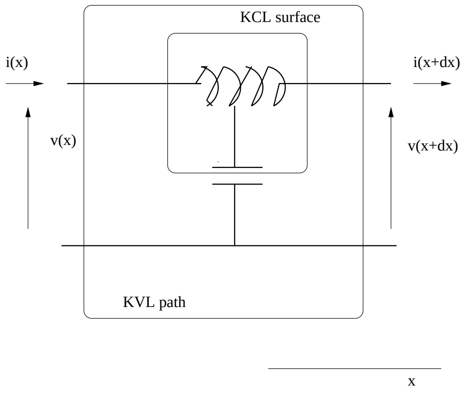

Figure 7: A differential section of a transmission line and the path (one dimension surface) along which the KVL (KCL) is evaluated.

Or,

$$\int_l \mathbf{E} \cdot \mathbf{dl} = -\frac{d\phi}{dt}. \tag{32}$$

$$\int_{\partial G} \mathbf{j} \cdot \mathbf{ds} = -\frac{dQ}{dt}. \tag{33}$$

Equation 32 corresponds to Kirchoff voltage law and equation 33 — to Kirchoff current law.

Note that the conserved quantities are the charge and the (magnetic) flux with $\mathbf{E}$ and $\mathbf{j}$ are their fluxes. However, the above equations are very different than equation 24; in equation 32 the change in ϕ is through a surface and ϕ's flux (sorry about the pun) is along the surface edge; in equation 33 the change in charge is in a volume and the charge's flux is through the volume boundary. In equation 24 G is a volume and ∂G is the boundary of the same volume. Thus, the equations are not quite symmetric with the integral taken over a path in one equation and over a surface in the second one. Moreover, there is not one conservation equations but two (or four - if you count equalities in each dimension) and each equation conserves a different quantity on a different (but related) domain.

As in the case of the diffusion equations, relations between the fluxes and the conserved quantities are required in order to solve the equations. Here these relations are of the type $E = E(Q, \phi)$ and $j = j(Q, \phi)$.

Example: Consider a linear transmission line with C (L) capacitance (inductance) to unit length. Figure 7 depicts a differential section of the line and the path l and the surface ∂G and the relative directions of i and v that we use to set the following equations.

$$v(x+\Delta x) - v(x) = -\frac{\partial \phi}{\partial t}$$

$$i(x+\Delta x) - i(x) = -\frac{\partial Q}{\partial t}$$

With $\phi = \Delta x \cdot Li$ and $Q = \Delta x \cdot Cv$ we get

$$v(x+\Delta x) - v(x) = -L\Delta x\frac{\partial i}{\partial t}$$

$$i(x+\Delta x) - i(x) = -C\Delta x\frac{\partial v}{\partial t}$$

Or,

$$\frac{\partial}{\partial t}\begin{pmatrix} i \\ v \end{pmatrix} = \begin{bmatrix} 0 & -\frac{1}{L} \\ -\frac{1}{C} & 0 \end{bmatrix} \frac{\partial}{\partial x}\begin{pmatrix} i \\ v \end{pmatrix}. \tag{34}$$

The more familiar form of the wave equation is derived from the above, i.e.,

$$\frac{\partial^2 v}{\partial t^2} = -\frac{1}{LC}\frac{\partial^2 v}{\partial x^2}.$$

Equation 34 can be treated similarly to equation 26 to derive a an approximation suitable for use of the amorpous computer scheme.

4.4 Time, messages and synchronization

This section considers the implementation of integration methods discussed above. The emphasis is not on the coding of equation 8 which is easy, but on the synchronization and the messages that transfer the data among the particles. Two kinds of implementations are considered: a) the particles are synchronized; b) the particles are not synchronized but they synchronize themselves by means of messages. The model of the particle message handling and the meaning of synchronization are discussed next.

The particle's program consists of three processes: a *receiver*, an *evaluator* and a *transmitter*. A message consists of the following parts: sender id, receiver id, and information. In the simulation of partial differential equations the information is the problem-time and state-vector at that instant of time.

The receiver maintains a queue of incoming messages. It deletes messages not addressed to it. Messages are delivered to the evaluator according to their 'time' component; Messages of lower time are delivered before messages of higher time disregarding their order of appearance.

Given a message, the transmitter chooses a random instant in τ and transmits the message at that instant.

An example of the actions of the evaluator is the following: The evaluator has a list of its neighbors; The evaluator stores the message it receives from the receiver; once all the neighbors' messages arrive, the evaluator (a) updates its own state, (b) increases its time by Δt, (c) clears its message buffer, (d) asks the transmitter to send a message with the new problem-time and state addressed to all its neighbors. Once that is done, the evaluator is ready to accept messages for $t + \Delta t$.

As mentioned in section 2, the communication between the particles is based on a time interval τ such that if the transmitting moment is chosen at random (equally distributed over τ) the probability of collision between messages sent from all particles residing within a radius r is negligible. This means that if all neighboring processors transmit using the same interval τ, an particle listening during this interval will receive all the messages sent (including his own).

Synchronous intervals: In this version all particles are synchronized in the sense that time is sampled, or divided into intervals of length τ and all particles have the same beginnings and ends of the intervals; moreover, all particles know which τ interval is the 'first' one and maintain interval count thereafter. During such a τ interval each particle sends one message to its neighbors and receives all the messages sent by the neighbors during the interval. The update of u is done when all the messages of the neighbors have been received and its own message have been sent.

Note that the integration of PDEs as presented in section 4 is a synchronous process well suited to this synchronous message exchange. Moreover, if computing time is negligible, and the total exchange, i.e., sending a message and receiving answers from all neighbors, takes time τ, u changes at the (real time) rate of

$$\frac{\Delta u_0}{\tau} = -\sum_{i \in N} c_{0i}(u_i - u_0)$$

which corresponds to the original equation 7 with Δt replaced by τ.

Asynchronous intervals: This model is asynchronous in the sense that the time intervals τ of different particles do not necessarily match or, in other words, there are phase differences between intervals of different particles although all have the same τ. In this case we say that the intervals are not synchronized or, that they are not 'in phase'. Note, however, that equation 7, is synchronized — time progresses in steps of Δt. The appeal of this model is that it assumes less about the particle than the synchronous model does.

To overcome the phase difference difficulty in this algorithm, the synchronization is done by messages. A message contains the problem-time in its time entry; an particle moves its own problem time by Δt once messages from all near-neighbors have been received and its own state updated.

If, however, an particle waits more than, say, 3τ for all its neighbors messages, it sends a repeat message which includes the the names of the missing particle's.

What happens if one of the neighbors fails and does not respond to some or all of its messages? As the algorithms are set now, they are going to fail. More work is required to find the reasons for this kind of failure and to write fail-safe programs for failures that might occur.

4.5 Higher accuracy integration formulas and the Voronoi diagram

There are other works that integrate differential equations were the domain is not sampled according to a regular grid. [23] and [24] present integration schemes of flow equations in which the sample points move with the fluid, are not on a regular grid and each sample point is assigned a a region of the domain using the Voronoi (Delaunay) diagram.

It was observed[5] that the above methods as well as our integration method presented above have $O(h^2)$ accuracy only, where h is the 'radius' of the largest Voronoi region, and achieving higher accuracy, say $O(h^n)$ for arbitrary n, is not known.

The purpose of this sub-section is to point out that the amorphous computing scheme can achieve higher accuracy and that without actually solving the above numerical analysis problem.

[5] A. Chorin

Let D be a closed bounded domain in 2 or 3D occupied by particles. The particles are randomly distributed in D and D's boundary. The distance between any two particles is non-zero.

Using the coordinates of the boundary particles all particles find their coordinates, their near-neighbors and their regions in the Voronoi diagrams.

Consider a equally spaced grid over D. Let the grid step (size) be smaller than half the distance between the closest particles. Each of the grid points falls in one of the Voronoi regions and is associated with the particle in the region. Those grid points that fall on a Voronoi boundary are associated to the left or right, etc., particles.

Now, the particles, being able computers and capable of communicating with their neighbors, can execute an arbitrary integration scheme using the *regular* grid as sample points. I.e., each particle that in the previous sections executed the operations of one sample point associated with the his own location now executes the operations of all grid points in its own Voronoi region.

If a regular integration formula on even spaced region is available the particles can execute it. Obviously, if that formula, working on a given problem, is stable and has $O(h^n)$ accuracy – so is the implementation using the amorphous computers.

While the above contributes some insight to amorphous computers (no bounds on computer power leads to interesting results) obviously it is does not achieve $O(h^n)$ accuracy with uneven samples.

4.6 Conclusions

It can be concluded from the above that amorphous computers can simulate differential equations derived from conservation laws. While only one example is carried out in detail, it is clear that other methods (see page 101 [11]) can similarly be implemented and analyzed. Moreover, the analysis techniques are a straightforward application of the body of available numerical analysis results.

It is interesting to note that to integrate the diffusion equation an particle does *not* have to know its own location or the location of its near-neighbors (see the corollary in 4.2). The particle must, however, have the ability to distinguish between messages received from its near-neighbors and messages originated elsewhere. It also has to have an estimate of A_0. As expected, boundary and/or initial conditions have to be assigned to each particle.

The numerical integration depends on the following properties of the particles: (a) The particle system is required to be regular. (b) All the particles start together and, while running, maintain the communication interval τ. (c) The communication system is reliable; i.e., the probability that a message does not arrive to its destination is negligible.

5 Communication revisited

The particle communication scheme described in section 2 belongs the a class of schemes called CDMA - channel division multiple access - or ALOHA-like schemes. These scheme were and are studied in detail by the communication networks people; reference [21] provides a very readable introduction to the area. Our own case, particles integrating partial differential equations, is somewhat different from the typical communication network model. The typical communication node is quiescent most of the time and than it transmits/receives a lot of data (burst mode). The integrating particles are very regular; at each sample time of the integration a particle transmits its new state value and receives the values computed by its near-neighbors.

This difference in behavior naturally calls for speeding up the communication by first synchronizing the particles and, once synchronized, choosing a time interval of sufficient length and assign each particle a time slot in the interval where it can communicate without collisions. Reference [22] discusses a similar communication scheme. However, the particle scenario is sufficiently different to rate a special discussion.

Synchronization is discussed next and after it – the algorithm for no collision communication.

5.1 Synchronization

The synchronization we are after is the synchronization of the time intervals τ of different processors; i.e., we would like all the particles to start counting their clocks from the "same" clock tick. Since the clocks have the same frequency, but not the same phase, sameness of clock tick is not that well defined. However, assuming that τ of $10^3 \times m$, m the clock period, is reasonable than a "phase difference" of one period between intervals of equal length is not critical. Thus, we shall attempt to have all particles count, each counts from the beginning of its interval and uses communication to adjust the count so it is the same as the count of some master particle.

Let a boundary processor be designated as a *master*. As power is turned on, and as part of the initialization process, the master transmits the message (*master*, *count*) where count is the number of the clock ticks of the master's τ interval in which the message is sent. The particles within a radius r of the master receive this message (presumably without latency), adjust their their own count to synchronize with the master's count, and, on the next τ interval, they transmit the message (*master_says*, *count*) which causes the particles receiving the message to adjust their count and transmit, etc.

If there are frequency differences between the clocks of various particles, yet these differences are small and manifest themselves only, say, after 10^8 cycles, we can use the above process to re-synchronize the count every, say, 10^6 τ cycles.

5.2 An alternative communication scheme

The communication scheme described in section 1 chooses a time interval τ that is long enough to reduce the probability of collision below a certain level. Now, once synchronization has been established (subsection 5.1), we divide time slots between the particles so that (a) there are *no collisions*; and (b) communication requires less time, i.e., the time interval τ in which a particle sends/receives a message to/from its environment can be made short.

Cidon and Sidi [22] observed that if two particles (communication nodes, etc.) are distanced more than $2r$ apart, their messages do not collide. They presented a simple centralized algorithm to assign slots to the particles to avoid collisions. This algorithm is the starting point for our discussion.

Let R be the set of particles; the N be a bound on the number of particles within a $2r$ radius; we choose a interval of length $N \times T_m$ where T_m is the time to transmit the information a particle has to transmit at each sample time. We consider the interval as consisting of N slots each of length T_m. The slot assignment algorithm follows:

1. Set i=0; if R is empty, terminate; otherwise, set $R1 = R$;

2. $i = i + 1$; if $i \geq N$ terminate;

3. Pick a particle from $R1$; call it p; this particle is assigned the i-th slot and will transmit in this slot; the particle is marked as 'assigned'. Remove from $R1$ all particles within one and two hops from p including p;

4. If $R1$ is not empty, goto step 3; else restore $R1$ by setting $R1 = R - \{all\ assigned\ particles\}$;

5. If $R1$ is not empty, go to step 2 else terminate.

Clearly, particles assigned to the same slot are too far from each other to interfere with each other messages.

Cidon and Sidi provide a distributed version of this algorithm. Yet that algorithm assumes that the particles are linearly ordered and each particle knows its place in the order; in addition, a control slot is available where each particle has and knows its mini-slot. These assumptions are not quite acceptable for the amorphous particles. A 'purely' distributed algorithm that does not make this assumption follows [6] .

Let N be an upper bound on the number of particle in a neighborhood of radius $2r$. Each particle provides a vector V of size n, $n > N$. The following algorithm assigns each particle an entry in V which is unique in the $2r$ neighborhood; i.e., each particle in a neighborhood has a different entry in V; particles in different neighborhoods might have the same V entry. The particle place in V is the slot in the $N \times T_m$ interval in which the particle transmits.

1. Each particle makes all entries in V unassigned and set itself 'unassigned'.

2. Each 'unassigned' particle randomly chooses an unassigned entry in V and $2r$-transmits[7] its choice.

[6] The term 'distributed algorithm' is not that well defined. Consider an algorithm in which one particle collects all particle coordinates. Once it has this information it applies the above algorithm to the information and inform the particles about the results. Clearly, the algorithm has distributed parts and serial parts.

[7] Transmission to all particles within a radius $k \cdot r$is done by messages that carry a counter to be decreased by one and retransmitted by the receiver until the counter becomes zero.

3. Each particle considers the messages received: If its own entry choice is unique, i.e., no other particle made the same choice, the entry becomes 'permanently' assigned to it and the particle is marked 'assigned'.

4. All particles announce their permanent assignment if any; all $2r$-neighborhood particles register the results in their own V.

5. Terminate if step 2 has been executed more than K times, K a predetermined number; otherwise go to step 2.

We would like to determine K so that the error probability (the probability that we would not have a slot for each particle in a $2r$ neighborhood) be less than a prescribed amount.

With n_p particles in the neighborhood and N entries in V the number of different arrangements of particles in entries is N^{n_p}. The number of arrangement that a certain particle be alone in an entry is $N \cdot (N-1)^{n_p-1}$. With each particle equally likely to choose any entry, the probability of a particle being alone in an entry is

$$\frac{N \cdot (N-1)^{(n_p-1)}}{N^{n_p}} = (1 - \frac{1}{N})^{(n_p-1)},$$

which is the probability of success in step 2 with N entries available in V and n_p particles.

The probability increase with N and decreases with n_p. We investigate the behavior of this probability function numerically with $N = n_p$ and $n_p < 100$. The result is a monotonic increasing function with the probability of success larger than $\frac{1}{3}$. Thus, the probability of success in K trials is better than $1 - (\frac{2}{3})^K$, which converges quite well[8].

Clearly, once the process is terminated, it is possible to establish N communication channels and transmit within a shorter interval.

The most obvious method to compare the original method of section 1 with the current one is the ratio between τ required for one method for achieving a certain error and the length of the time interval required for the alternative method. The actual computation, taking the message length into account, is left as an exercise for the interested reader.

[8] (pp(map (lambda(n)(- 1. (power (/ 2. 3) n))) (list 4 6 9 10 20 40 80 100))) value: (.8024691358024691 .9122085048010974 .9739877051262511 .9826584700841674 .9996992713401783 .9999999095622731 .9999999999999918 1.)

6 Structure and interpretation of Amorphous Computing - A Summary

The title was chosen to catch the eye of whoever looks through the contents of the notes. In spite of its tongue-in-cheek sound the title is relevant here. We are going to structure amorphous computing into two levels of abstraction that, for a lack of better name, are called the *computing level* and the infrastructure level or the *communication level*. We are going to discuss some of the conditions under which this partition into levels make sense, some of the questions associated with each level and some of the price paid for this benefit.

The computing level of abstraction assumes that every particle knows whatever it needs to know about the state of its near-neighbors. This knowledge is application dependent. In the diffusion equation of section 4 this information is the current states of the near-neighbors and who they are. This abstraction level also knows the current (sampled) time and is able to compute the particle's next state and transmits the new states to all the particle's neighbors.

The communication level does all the communication as well as the initialization, which includes finding the locations, synchronization, transmitting and receiving of messages, etc.

The advantage of the partition to these two levels is that the computing level does not need to know how the information it is using is generated and supplied. Moreover, it makes it simpler to apply standard mathematical and numerical analysis results.

As levels of abstraction usually do, a level implementation can be changed without affecting the other level. Two communication schemes illustrate this partition. In section 4.4 an particle communicated with all its neighbors simultaneously and synchronously. Another scheme is plausible and is used in [4]; each particle seeks a 'mate' and forms a pair; once a pair is formed, information is exchanged and computation is performed. The process then repeats. It is assume that the selection of mates is 'fair', i.e., the particle has equal probability to select as mate any member of its environment.

There is another subtle yet important issue in this abstraction level is the issue of speed and density of particles. The time required to get all information from the environment to an particle, i.e., the *communication speed*, has to be fast as compared with the time-constants of the physical problem being solved or simulated. Similarly, the distance between particles has to be small as compared with relevant distances[9] of the problem being solved. It does not mean that a problem cannot be solved if these two conditions are not satisfied; it just means that you cannot establish the abstraction-boundary and ignore the way the communication layer is implemented.

The abstraction point of view seperates the problems to be investigated to problems that are typical of the computing level and problems typical of the communication level. An incomplete list of the latter follows:

- Determine the locations of each particle from locations of the boundary.
- Determine the neighbors of each particle.

The problem discussed in section 4 is essentially a computation level problem.

7 Appendix: Regular Voronoi diagrams

Given points S in $2D$ the Voronoi region of a point p_0 is given by

$$\{x : x \in 2D, \|x - x_0\| < \|x - x_i\|, x_i \in S, x_i \neq x_0\}.$$

Two regions with a common border are called adjacent. A graph formed by connecting each point in S with other points in S that have adjacent regions is called the Delaunay graph or diagram. Figure 5 illustrates the Voronoi and the Delaunay diagrams. The *near neighbors* of a point A are the points connected to A by one Delaunay arc.

These diagrams had attracted a fair amount of attention [15, 16, 17, 18]. By definition (section 4) our partition of the space corresponds to the Voronoi diagram. However, the requirement under which the approximation has been made admits only distributions of points whose Voronoi diagrams satisfy the regularity requirement. Translated into the diagram terms, the regularity requirement means that each Delaunay arc between two near-neighbors intersects the corresponding Voronoi vertex.

We assume that each points has its coordinates and all points transmit their coordinate so that each point has the coordinates of all points within

[9] It is interesting that there is not a word for space constant similar to time-constant.

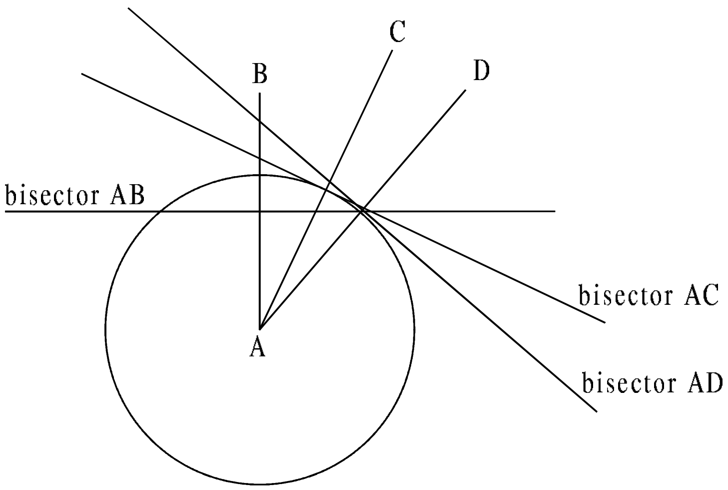

Figure 8: A later point can eliminate the effect of an 'outside' point – an illustration. C is farther away from A than B; D is farther away from A than C; bisector AD eliminate the effect of bisector AC from the Voronoi boundary of A.

radius r from it. Now, each point has to use that information to determine its near-neighbors. The sequel presents an algorithm for doing so. The algorithm is parallel, all nodes work simultaneously once the above information is available. Considering the length of [18] I can hardly believe that the algorithm is novel. It provides an introduction and demonstrates what a particle has to do in order to determine its near neighbors from the coordinates of its neighbors.

Preliminaries - elementary analytic geometry relevant to this algorithm: Let $ax + by + c$ be the equation of a line in $2D$. Let (x', y') be coordinates of a point. $d = -\frac{ax'+by'+c}{\sqrt{a^2+b^2}}$ is the distance from the point (x', y') to the line. If the line does not go through the origin and through (x', y') and if the sign of the square-root is taken to be the same as the sign of c, d is positive if (x', y') is on the same side of the line as the origin and negative otherwise. Thus, we have a simple method to determine which side of a line a point is.

A midpoint of two points (x, y) and (x_1, y_1) is the point $(\frac{x+x_1}{2}, \frac{y+y_1}{2})$.

The bisector of a point x_0 and a point x_1 is a line through the midpoint of x_0 and x_1 that is perpendicular to the segment $[x_0, x_1]$.

The following is an outline of the algorithm that each particle (point) uses to find its near-neighbors and to determines whether the point diagram is regular.

Algorithm:

Sort: Sort all the neighbors of x_0 (all nodes whose distance from x_0 is smaller or equal to r) by their distance from x_0; Put the neighbors in a queue, the smaller distance node first.

x_1: Let x_1 be the node on the top of the queue. Remove it from the queue.

Initial: Declare x_1 a near-neighbor; find the bisector of x_0 and x_1. A section of this line is a part of the Voronoi diagram. The line divides the plan to two; the plan's 'inside half' is that side where the origin x_0 is. Let S be the 'inside' half plan.

Table: We equip ourself with a table whose entries are:

point	midpoint	line/bisector	segment	in/out

The point x_1, its midpoint, and the bisector are put in the table. The 'segment' is the linear section of the Voronoi region associated with x_1; it will be added later and so is the 'in/out' entry.

Again: The point on top of the queue, call it x, is treated next. Although the formal treatment is almost the same, it makes a difference if $(midpoint\ \ x) \in S$ or not.

Assume $(midpoint\ \ x) \in S$. A part of the bisector forms the region boundary. The intersections of $bisector\ \ x$ with the segments and lines in the table are found; there are at most two intersection points. The table is updated (Two segments associated with other points might change) and x entry in the table is formed and put in the table. The in/out entry is set to $true$.

Assume $(midpoint\ \ x) \notin S$. This case is more subtle. Clearly if $(bisector\ \ x)$ does not intersect S (actually, does not intersect the lines and segments in the table) the situation is simple and the point is ignored. If the bisector does intersect S, a segment of it is a candidate to become the region boundary. That means trouble because the Delauny arc $(x\ \ x_0)$ and the segment x generates do not intersect. On the other hand, it is too early to declare the points not regular, since a further point (still in the queue) might remove the 'outside' point bisector so that it does not have any effect – see figure 8. Thus, as above, the intersections of *bisector* x with the segments and lines in the table are found; there are at most two intersection points. The table is updated (Two segments associated with other points might change) and x entry in the table is formed and put in the table. The *in/out* entry is set to $false$.

Radius: If S is bounded (the table contains only finite length segments) the radius R of S (distance of farthest corner of S from the origin) is evaluated and all points whose distance from the origin is larger than $2R$ are dropped from the queue.

Termination: If the queue is not empty goto *again*. Otherwise, check the table: if the table has an entry with $in/out = false$ announce that the system of points is not regular and terminate; If such an entry is not found terminate.

The above algorithm is executed in parallel by all particles. If the algorithm terminates without an error message, each particle's table has the coordinates of the particle's near-neighbors.

Acknowledgment

I thank the members of the Amorphous Computing group for introducing me to the subject. I thank Professor H. Abelson for his penetrating remarks on the manuscript. I thank Prof. A. Chorin, University of California, Berkeley for explaining subtleties in the Lax theorem, and Mr. Kevin Lin, University of California, for reading and commenting on the manuscript. Professors R. Rom and M. Sidi, Technion, directed the author to computer network literature.

Support for this research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-96-1-1228.

References

[1] H. Abelson, T. Knight and G. J. Sussman, "Amorphous computing," *White paper*, October 1995. http://www.swiss.ai.mit.edu/ switz/amorphous/.

[2] D. Coore "Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer", Ph.D. Thesis, Electrical Engineering and Computer Science Department, MIT, 1999.

[3] Radihika Nagpal, "A computational Model for an Amorphous Computer," unpublished.

[4] Erik Rauch, private communication.

[5] . Abelson, D. Allen, D. Coore, C.P. Hanson, G. Homsy, T.F. Knight,Jr., R. Nagpal,E. Rauch, G.J. Sussman, R. Weiss, "Amorphous Computing," Communications of the ACM, Vol. 43, No. 5, May 2000, pp 74-82.

[6] T. M. Apostal, "Mathematical Analysis," Addison-Wesley Publishing Company, Reading MA, 1960, Chapter 6.

[7] Peter D. Lax, "Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves," SIAM, Philadelphia, Pennsylvania, 1972 (monogram).

[8] K. W. Morton and D. F. Mayers, "Numerical Solutions of Partial Differential Equations - An Introduction," Cambridge University Press, 1994.

[9] R. D. Richtmyer and K. W. Morton, "Difference Methods for Initial-value Problems," Interscience Publishers, 1967 (2ed edition).

[10] G. H. Golub and C. F. Van Loan, "Matrix Computation," 2ed Edition, The John Hopkins University Press, 1989.

[11] R. J. LeVeque, "Numerical Methods for Conservation Laws," Birhauser Verlag, 1992.

[12] R. Courant and D. Hilbert, "Methods of Mathematical Physics," John Wiley & Sons, 2ed edition, 1989, p.154, p. 188.

[13] H. Abelson, A. A. Berlin, J. Katzenelson, W. H. McAllister, G. J. Rozas, G. J. Sussman, and J. Wisdom, "The Supercomputer Toolkit: A general framework for special-purpose computing," International Journal of High Speed Electronics, Vol. 3, Nos. 3 and 4, pp 337-361, 1992.

[14] J. Katzenelson and A. Goikhman, "The Supercomputer Toolkit and Its Applications," EE Memo 1165, Department of Electrical Engineering, Technion, June 1998. Accepted for publication Int. Journal of High Speed Electronics and Systems.

[15] Joseph O'Rourke, "Computational Geometry in C," Cambridge University Press, 1994.

[16] F. P. Preparata and M. I. Shamos, "Computational Geometry - An Introduction", Springer Verlag 1985.

[17] Aurenhammer F. "Voronoi Diagrams - A servey of a foundmental geometric data structure," ACM Computing Surveys, 23, 345-405, 1991.

[18] A computational geometry data base: http://www-ma2.upc.es/ geomc/geombibe.html

[19] W. Feller, "Introduction to Probability Theory and its Applicatio," John Wiley & Son, Inc. 1950.

[20] G. Strang, "Introduction to Applied Mathematics," Wellesley-Cambridge Press, Wellesley, MA, 1986, page 579.

[21] A. S. Tanenbaum, "Computer Networks," Prentice-Hall, 1988.

[22] I. Cidon and M. Sidi, "Destributed assignment Algorithms for Multihop Packet Radio Networks," rans. on Computers, Vol 38, No. 10, Octover 1989.

[23] C. Borgers and C. S. Peskin, "A Lagrangian fractional step method for the incompressible Navier-Stokes equations on the periodic domain," J. of Comp. Physics, 70, pp. 397-438, 1987.

[24] G. Russo and J. A. Strain, "Fast Triangulation Vortex methods for the 2-D Euler equations," J. Comput. Physics, 111, pp 291-323, 1994.