2012

Parallel Sorting on a Spatial Computer

Max Orhai
Portland State University, max@shinypebble.net

Andrew P. Black
Portland State University, black@cs.pdx.edu

Parallel Sorting on a Spatial Computer

Max Orhai Andrew P. Black

Portland State University
{orhai,black}@cs.pdx.edu

August 2012

1 Introduction

A *spatial computer* [1] is a network of small computing devices that are located in a physical space, and sparsely connected together in a way that reflects their location in that space. This means that most or all connections are between physically nearby devices, so the cost of communication depends on the physical distance between the components that are communicating. Each computing device acts as a router in its immediate neighborhood, and stores a limited amount of local data.

The idea of a spatial computer was developed to facilitate the design of large-scale sensor networks, mobile robot 'swarms', and biological modeling. However, modern large-scale computer clusters and many-core processors bear an increasing resemblance to spatial computers, in that the cost of communication — whether measured in time or energy — increases with the physical distance between the communicating devices. Spatial computers offer considerable potential for parallel computation, but also new constraints for the *spatial programs* that run on them.

Spatial programming techniques suggest a new approach to the design of fundamentally parallel algorithms for practical computing in large-scale device networks, where large amounts of spatially-distributed computing capacity is available but the cost of communication is proportional to the physical distance it must span. The techniques that are beginning to be developed for these purposes may be applicable to more sophisticated algorithms and data structures, in a variety of spatial computing contexts. In particular, the representation of a problem in terms of concurrent interactions between active data elements, rather than passive data that is acted on by one or more processors, yields a simple system that makes use of our intuitions about the behavior of physical entities.

2 Collision Sort

The problem of sorting is one of the oldest and most fundamental in computer science, as well as being of great practical importance. Sorting also has a natural spatial interpretation, and for these reasons we feel that it is an interesting problem for exploring the potentially novel characteristics of spatial programs.

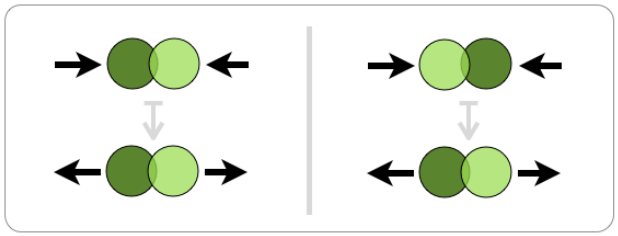

Figure 1: Two conditional collision events, each involving two particles in horizontal motion; the effect of the collision depends on their relative color. In both cases, the darker-colored particle is sent to the left, and the lighter-colored particle to the right.

The distributed sorting algorithm we present is designed for the grid topology of contemporary many-core computer systems, potentially scaled up to thousands of cores, but can also be implemented in a GPU. Our algorithm, *collision sort*, represents data by mobile 'particles' in a manifold-like distributed data structure, hosted by a regular patchwork of computing devices. Like other distributed algorithms, collision sort is designed to tolerate nondeterminism in the order in which components communicate. It can also adapt to a limited number of failed components. The techniques we develop invite new ways of thinking about large-scale dynamic data-flow systems. We suggest that some techniques developed for simulations of physical, chemical, or biological phenomena embedded in physical space may serve equally well to organize general-purpose algorithms in a spatial computer.

2.1 The Model: Colliding Particles

Imagine a distributed particle system, where the representation of space corresponds with the topology of the physical communication network connecting the component computing devices. In the simplest case, this may be a linear array of devices, each with a left and right neighbor, except for those at the ends of the array. Further imagine that time is discrete; at each time step, *particles*, representing data values, are passed between neighboring machines. We can regard an interaction between two particles as a kind of *collision*, illustrated in Figure 1. The outcome of the collision depends on the data values represented by the particles, according to the following mechanism, which is implemented in each computing device.

Consider a set of particles P, where the sortable *value* of a single particle $p \in P$ is $v(p)$. Let each particle p have the same fixed *size* s, but individual *location* $x(p)$ and *velocity* $\dot{x}(p)$, where $\dot{x}$ takes on one of the values $-c$ or $+c$ per unit time, where $c < s$. Thus, at each time step, the location $x(p)$ of particle p changes by $\pm c$. The range of x is a bounded interval $[x_{min}, x_{max}]$ which will be the key space. The sort is complete when, for any two particles p and q, $x(p) < x(q)$ if and only if $v(p) < v(q)$. At any time, each particle p may *collide* with any overlapping particle q, where $|x(p) - x(q)| \leq s$. Should p overlap multiple other particles, one may be chosen at random. If $v(p) < v(q)$ and $x(p) < x(q)$, or if $v(p) > v(q)$ and $x(p) > x(q)$, then the collision results in negating $\dot{x}(p)$ and $\dot{x}(q)$. Otherwise, $\dot{x}(p)$ and $\dot{x}(q)$ are unchanged.

Thus, depending on relative values of p and q, a collision will result either in particle p 'bouncing off', or 'passing through', particle q, as shown in Figure 1. If each particle is allowed to detect and

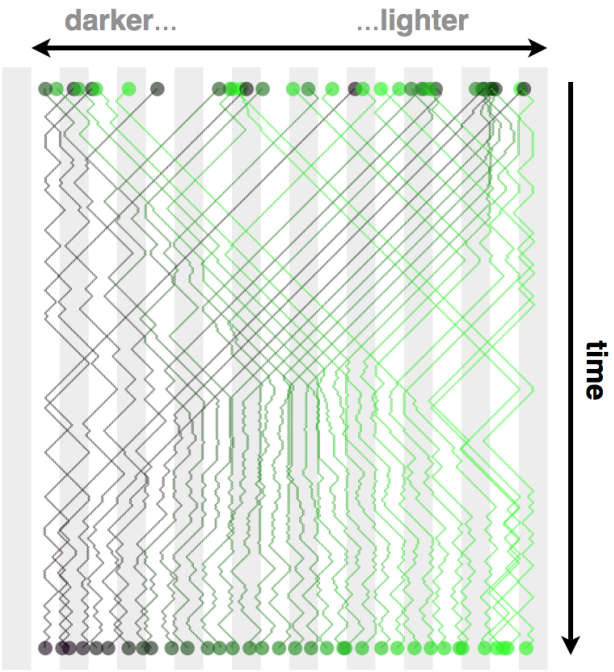

Figure 2: One-dimensional distributed collision sort with constant particle velocities. The grey vertical bands show the division of the space into regions. The wavy lines indicate the trajectories of the particles, whose colors correspond to their values.

respond to collisions once per time step, the particle system will attain a stable sorted configuration, in which $x(p) \leq x(q)$ if and only if $v(p) \leq v(q)$. This stable configuration will be attained after a number of time steps $|x_{max} - x_{min}|/c$, which is the number of steps needed to traverse the space. This number is independent of the order in which the collisions occur.

2.2 Mapping the Model to a Spatial Computer

Although the above procedure can be performed by a single sequential machine, it makes much more sense in parallel. After all, it would be surprising if the amount of work done by such a particle system did not depend on the size of particle set P. So let us now consider the communications between neighboring machines. Suppose that we have m sequential machines $\{A_i : 1 \leq i \leq m\}$ connected in a line; let's partition the space between them. Assign each machine A_i, in order, a non-overlapping region of space with size $S = |x_{max} - x_{min}|/m$, which we take to be no smaller than the size s of a particle, so that the entire range of x is covered. It is important that machine A_i be able to distinguish between its 'left' neighbor A_{i-1} and its 'right' neighbor A_{i+1}, should both of these exist. No communication channels other than these two are necessary for a one-dimensional sorting process, and no individual machine needs access to global information such as the total number of machines, the amount of data, or its own absolute position within the system.

Because particles have size $s \leq S$, each particle can be in at most two regions. Within a region, the above collision rule is executed sequentially on all of the particles in that region, in arbitrary

order. In each step, particles that move to locations greater than $i \cdot S$ are passed from machine A_i to machine A_{i+1}, and those with locations less than $(i-1) \cdot S$ are passed to machine A_{i-1}. If $i = 1$ or $i = m$, those particles that would step 'out of bounds' instead have their velocities negated: they can be thought of as 'bouncing off' the boundaries of the space.

It should be noted that the amount of real computation performed by a machine in each time step depends on the number of particles processed by that machine. In a system with n particles evenly distributed between m machines, this number will be n/m. At equilibrium, the particles will be approximately-evenly distributed; the particles diffuse somewhat like the molecules of a gas.

An unresponsive machine can be treated as a "hole" in the space, from which particles rebound. In one dimension, such a hole will partition the array of machines, causing the particles on either side of the gap to be sorted independently. In higher dimensions, particles will move around the hole, and the global sorting process can continue. The system thereby tolerates some amount of component failure. No further coordination or synchronization is needed; as the particles disperse through the space, the system spreads the load to do as much work in parallel as possible, given the limited number of machines. The evolution of a distributed collision sort in one dimension is shown in Figure 2.

2.3 Reducing Communication

One potential difficulty with partitioning the space in this way is detecting collisions across the boundaries between regions. Suppose particles p and q are located in adjacent regions corresponding to machines A_1 and A_2, so that they overlap across the boundary, *i.e.*, $x(p) + s/2 > S > x(q) - s/2$. Must A_1 and A_2 communicate to detect this collision? According to our definition of the collision mechanism, the answer would seem to be "yes". The requirement for communication would seem to increase when the dimension d of the space is greater than 1, since a particle may occupy as many as 2^d adjacent regions. Communication necessarily involves synchronization, and synchronization costs limit the scaling of concurrent algorithms. The complexity and cost of any collision-detection scheme that involves communication is unappealing, especially in an asynchronous system. Can we reduce the need for communication by some sort of approximation?

Experiment shows that we can: communication for collision-detection is unnecessary. This is illustrated in Figure 3; in this experiment, the machines access only local information, and particle values are compared not with each other, but with the mean local particle value in their region. As the figure shows, at equilibrium the particles are still sorted. The figure also shows the effect of another modification, which speeds-up convergence to the equilibrium state: the velocities of particles are no longer constrained to be $\pm c$, but instead vary between S and $-S$ in proportion to the difference between their values and the mean local particle value in the region.

The synchronization involved in communication has not been entirely eliminated, since it is still necessary to "hand off" a particle from one region to another. Without some amount of buffering, these hand-offs will require the sending and receiving machines to wait for each other. However, the communication required by the algorithm has been reduced to asynchronous message passing between nearest neighbors. With appropriately sized communication buffers, the wait can be substantially reduced in practice.

Particles tend to settle in locations that span a region boundary, existing in a dynamic equilibrium where the particles bounce back and forth between the adjacent regions. To eliminate

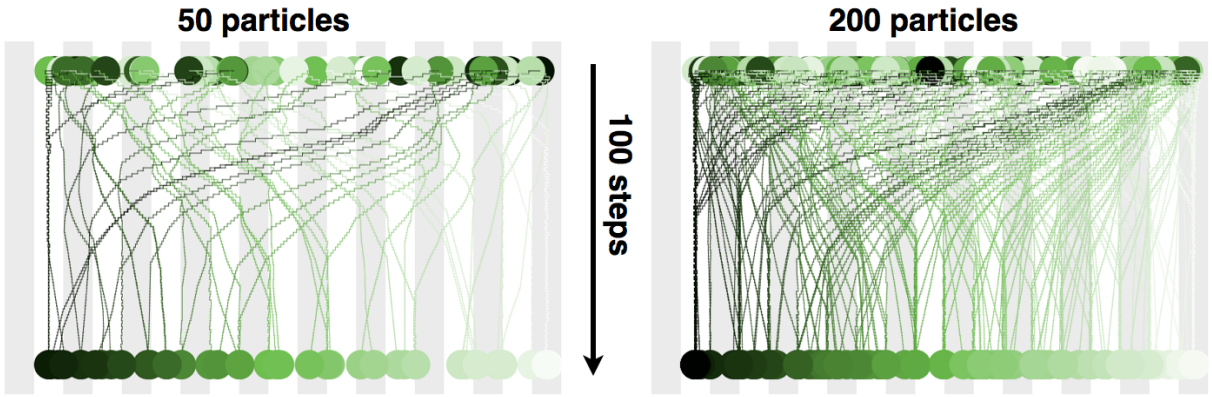

Figure 3: Collision sort with particle velocities proportional to the difference between the particle's value and the mean value in its local region. The system's evolution over 100 steps is shown with two different amounts of data.

unnecessary communication because of this phenomenon, it is convenient to set particle velocities below some threshold to zero.

2.4 Higher-dimensional Sorting

Collision sort extends very naturally into as many dimensions as we please, so long as we can connect the component machines in a square, cubic, or hyper-cubic lattice. Figure 4 shows a simultaneous two-dimensional collision sort, under conditions similar to those of Figure 3. Approximately-sorted conditions are obtained quite rapidly, although the definition of "step" in Figure 4 is a little different from that that applies in Figure 3.

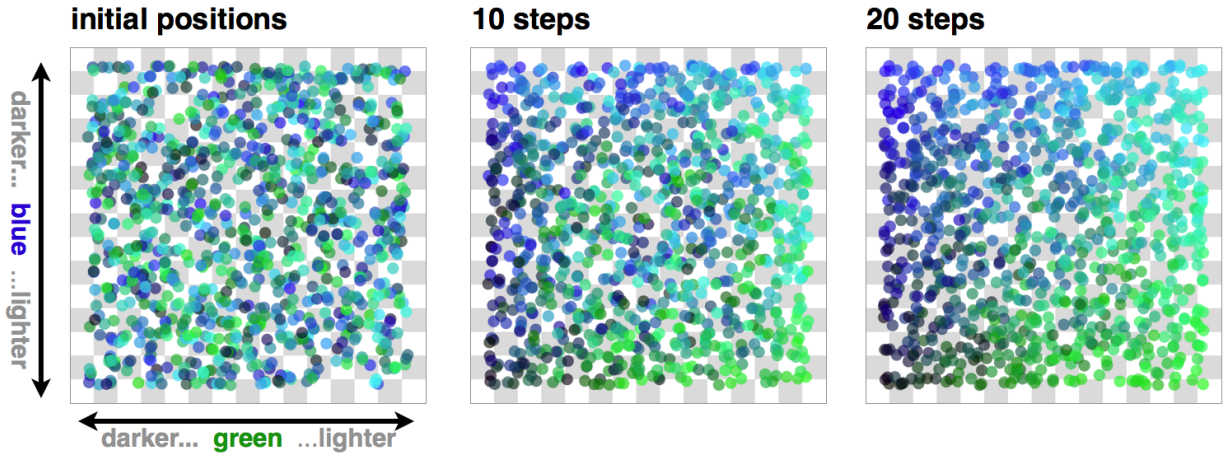

Figure 4: Variable-velocity two-dimensional collision sort of 1000 particles, each of which has both a blue (x) and a green (y) value.

The 2-dimensional model simulated in Figure 4 assumes that a particle can move not just in one of the four cardinal directions (N, E, S or W) but also in one of the four intercardinal directions (NE, SE, SW and NW). If we assume that the grid of machines that hosts the collision sort is a rectangular lattice connected only on N–S and E–W axes, a complication arises when a particle moves from a machine to one of its intercardinal neighbors, since these neighbors are not directly connected. Such a move is treated as a single step in our simulation, but would actually require two inter-machine messages in a real rectangular lattice. An implementation would have to introduce a forwarding mechanism, whereby, for example, a particle bound for the NE neighbor is sent first N and then forwarded E, without being compared for collisions during the intermediate step. This will introduce some additional delay or 'noise' into the system. The mechanism of the distributed algorithm can easily absorb this additional nondeterminism: even in the one-dimensional case, particle steps that cross boundaries between machines already take more real time than those that do not. We need to conduct further experiments to determine if it is reasonable to permit diagonal moves in a single computational "step".

3 Related Work

Conceptually, collision sort is closely related to the parallel bubble sort, sometimes called "even-odd transposition sort," which was described by Habermann in 1972 [8]. Unlike bubble sort, collision sort does not assume a one-to-one correspondence between machines and data, nor synchronized communication. The variable-velocity modification improves efficiency by allowing data to skip over unnecessary comparisons. We are not aware of any prior work on simultaneous multi-dimensional sorting.

Discretized computational models of physical particle systems have a long and rich history, dating back as far as computing pioneer Konrad Zuse's very early "Calculating Space" lattice-gas cellular automata [16]. In the 1980s, particle systems found uses in computer graphics models of fluids, fabrics, and other physical objects having complex dynamics. Modern GPU processors are capable of handling million-particle systems in real time [10], which suggests that collision sort may have an efficient parallel implementation in commodity hardware.

Particle Swarm Optimization methodology [9], developed originally to model social behavior in animals, has found applications in a variety of nonlinear or otherwise irregular optimization problems. However, Particle Swarm Optimization requires nonlocal communication, and does not sort a space, but rather searches through it. Clustering algorithms inspired by social insect behavior [6, 15] have also found industrial uses in document retrieval and mapping: *ant-based clustering* is a spatial technique that acts somewhat like a randomized bucket sort, with a swarm of agents moving passive data particles according to simple rules. It has the advantage of not relying on a global coordinate system, although there is consequently no control over where in space the data clusters form.

Research into programming languages for spatial computers is ongoing. Some approaches, such as the MIT Amorphous Computing project's Growing Point Language [4] and the more recent Proto programming language [2], approximate a discrete spatial computer by a continuous topological manifold, and work with mathematical formalisms in the continuous abstract space. Others, such as cellular automata, the MGS topological simulation system [14], or the Blob computing architecture [7], work with aggregations of discrete computing elements. The sorting algorithm we

present contains elements of both approaches: the discrete data particles move in an approximation of a continuous space maintained by a discrete network of machines.

4 Discussion

The collision sort algorithm uses both more time and more work to sort than many well-known sorting algorithms for parallel random-access machines. For example, the GPU radix sort of Satish, Harris, and Garland [13] does only $O(n)$ work at a roughly constant rate. However, this algorithm, like most published parallel sorting algorithms, assumes global uniform access to memory, and does not account for communication costs. In a distributed computing environment where communication cost is dependent on physical distance, it is not clear whether scalable sorting algorithms as fast as these are possible. The Quicksort implementation described by Gruau [7] begins with a configuration in which all data are directly connected to a central coordinating element, and thus depends on an abstraction layer to map this un-spatial configuration into a spatially constrained network. With large numbers of data elements, it is not possible for them all to communicate with a central coordinator in the same short time; the fundamental trade-off for an element of a spatial computer is between the number of other reachable elements and the distance (and thus, communication time) to the farthest of them. Scalability, failure handling, and concurrency are key challenges of distributed systems [5], and collision sort is designed to address these issues.

The springboard for this work is the idea that the theoretical tools developed to model or simulate physical, biological, and social systems that are embedded in physical space may also be useful in the development of spatially-embedded computing systems. Many of the techniques of object-oriented programming that have proven their worth in industrial software development have their origin in discrete-event simulation systems like Simula [12]. The techniques have been generalized from the specific descriptive needs of modelers to powerful abstract concepts like class hierarchies or message-passing, which can be used to create new systems as well as describe existing ones. Perhaps the systematization of spatially-oriented simulation techniques in the sciences may yield new concepts with which to create spatial computing systems: cellular automata are a paradigmatic example. In any case, as computer systems have become more diverse and more powerful, the constraints that they impose on their programs are beginning to have more to do with the constraints of physical systems, and less to do with the features of any specific model of computation. New techniques for the analysis and construction of spatially-embedded information processing systems should complement each other. The recent development of explicitly-spatial process algebras, such as Milner's Bigraphical Model [11] or the Geometric Process Algebra of Cardelli and Gardner [3], is an encouraging trend in this direction.

5 Experimental Methods

In developing the distributed algorithms of the previous sections, we have made use of the RoarVM parallel Smalltalk virtual machine and the NetLogo concurrent agent-based modeling environment. RoarVM is a multi-threaded execution environment developed for many-core processors, while NetLogo simulates the parallel execution of programs by randomizing the execution order of agents' programs within each simulated global time step. Source code for the programs that have

generated the images shown in the figures is available at http://cs.pdx.edu/~orhai/spatial-sorting. RoarVM Smalltalk is available from https://github.com/smarr/RoarVM, and NetLogo from http://ccl.northwestern.edu/netlogo/.

6 Conclusion

We have presented collision sort, a distributed sorting algorithm for spatial computers having a grid topology. This algorithm is designed to use only local communication to accomplish the global task of sorting, while tolerating nondeterminism in the order of communications as well as limited component failure.

Spatial programming techniques suggest a new approach to the design of fundamentally parallel algorithms for practical computing in large-scale device networks, where large amounts of spatially distributed computing capacity is available but the cost of communication is proportional to the physical distance it must span. Our technique is a compromise between the conventional sequential approach, where a single computing element must examine all of the data, and a massively parallel approach, in which every data element is active. The former won't scale because it is inherently sequential; the latter won't scale because it requires exponentially-increasing amounts of non-local computation. Instead we *localize* computing resources in space, and move the particles representing the data through the space so that local computation is all that is needed. It remains to be seen whether such techniques can be applied to an interesting range of problems; our intuition is that they can be, because the physics of the real world also favors local interaction over long-distance interaction

Using only asynchronous local communication, in environments whose exact configurations may be unknown or dynamic, our methods must accommodate some uncertainty or nondeterminism. Drawing from mathematical models developed to study natural phenomena, we find that some of the same approaches have utility in the construction of practical distributed algorithms in a spatial computing environment.

Acknowledgements

This work was supported by the Undergraduate Research & Mentoring Program of Portland State University's Maseeh College of Engineering and Computer Science, and by a gift from IBM. We are indebted to Uri Wilensky for the NetLogo modeling environment, and to Stefan Marr and the rest of the RoarVM team; special thanks go to Sam Adams and David Ungar of IBM Research.

References

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 43:74–82, May 2000.

[2] J. Beal. A basis set of operators for space-time computations. In *Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW)*, pages 91–97, 2010.

[3] Luca Cardelli and Philippa Gardner. Processes in space. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes*, volume 6158 of *Lecture Notes in Computer Science*, pages 78–87. Springer, Berlin–Heidelberg, 2010.

[4] D. N. Coore. *Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer*. PhD thesis, Massachusetts Institute of Technology, 1999.

[5] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Pearson Addison Wesley, Harlow, England, 2 edition, 2001.

[6] J. L. Deneubourg, S. Goss, N. Frank, Sendova A. Franks, C. Detrain, and L. Chretien. The Dynamics of Collective Sorting: Robot-Like Ant and Ant-Like Robot. In J. A. Meyer and S. W. Wilson, editors, *Proceedings First European Conference on Simulation of Adaptative Behavior: From Animal to Animats*, pages 356–365. MIT Press, Cambridge, 1991.

[7] Frédéric Gruau, Yves Lhuillier, Philippe Reitz, and Olivier Temam. Blob computing. In *Proceedings of the 1st conference on Computing frontiers*, CF '04, pages 125–139. ACM, 2004.

[8] A. Nico Habermann. Parallel neighbor sort (or the glory of the induction principle). Technical Report AD-759 248, Carnegie Mellon University, 1972. available at http://repository.cmu.edu/compsci/2087.

[9] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948, 1995.

[10] P. Kipfer, M. Segal, and R. Westermann. UberFlow: a GPU-based particle engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '04, pages 115–122. ACM, 2004.

[11] Robin Milner. Bigraphs and their algebra. *Electronic Notes in Theoretical Computer Science*, 209:5–19, April 2008. Proceedings of the LIX Colloquium on Emerging Trends in Concurrency Theory (LIX 2006).

[12] Kristen Nygaard and Ole-Johan Dahl. The development of the SIMULA languages. In Richard L. Wexelblat, editor, *History of programming languages I*, pages 439–480. ACM, New York, NY, USA, 1981.

[13] N. Satish, M. Harris, and M. Garland. Designing efficient sorting algorithms for manycore gpus. In *IEEE Int. Symp. on Parallel Distributed Processing (IPDPS 2009)*, pages 1–10, May 2009.

[14] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. A topological framework for the specification and the simulation of discrete dynamical systems. In Peter Sloot, Bastien Chopard, and Alfons Hoekstra, editors, *Cellular Automata*, volume 3305 of *Lecture Notes in Computer Science*, pages 238–247. Springer, 2004.

[15] H. Van Dyke Parunak. "Go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[16] K. Zuse. Calculating space. Technical Report AZT-70-164-GEMIT, Massachusetts Institute of Technology, 1970.