Spatial Sorting Algorithms for Parallel Computing in Networks

Max OrHai
Portland State University
orhai@pdx.edu

Christof Teuscher
Portland State University
ECE department
teuscher@pdx.edu

***Abstract*— Many basic techniques in computer science have been founded on the assumption that physical computing resources are scarce but orderly, and that the cost of effective direct communication between physically distant parts of a computer system is affordable. In large scale cluster computing installations, fine-grained parallel computing hardware, or wireless mesh networks, these familiar assumptions may not hold. In this paper we present adaptations of two especially simple classic sequential sorting algorithms, namely bubble sort and insertion sort, to parallel execution in spatially constrained networks of devices, using particle systems and asynchronous automata graphs. In both cases we are able to get significant speed-ups of these naïve $O(n^2)$ algorithms, attaining linear time complexity or better given sufficient parallelism. For insertion sort, we obtain these results without depending on any but statistical properties of the network medium. We discuss potential extensions of these physically and biologically inspired techniques to more complex parallel algorithms.**

I. Introduction

The contemporary computing environment already contains networks of large numbers of devices, which may be connected in unknown or varying ways. It is reasonable to expect that these networks will become even larger, more distributed, and more dynamic in the future, as individual computing devices, and their components, become smaller and less expensive. To best take advantage of the parallel computing potential that large-scale device networks offer, we must minimize the overall cost of communication through physical space, while being able to accommodate the uncertainty of an unknown or irregular network topology.

A spatial computer [1] is a network of computing devices which are located in a physical space, and connected in a way that reflects their locality in that space. The cost in time, energy, or reliability of communicating through such a network is proportional to the physical distance that messages must travel. It is generally assumed that the intended function of a spatial computer will correspond in some way to its shape. Some approaches, such as the MIT *Amorphous Computing* project's *Growing Point Language* (GPL) [2] and the more recent *Proto* programming language [3], treat the spatial computer as a discrete approximation of a topological manifold, and work with formalisms in the continuous abstract space. Others, such as the *TOTA* physical tuple-space [4], do not introduce a continuous abstraction, but work directly with the discrete computing elements. Both approaches are represented here.

In this paper we discuss two inherently parallel spatial adaptations of simple canonical sorting algorithms. The first, *collision sort*, is a generalization of bubble sort that represents data as mobile agents in a partitioned computing manifold. The second algorithm is a parallel version of insertion sort which incrementally embeds a space-conserving active data structure in an arbitrary static network, given a minimum average per-node connectivity. This embedding is an example of how a simple developmental process can coordinate the growth of active data structures in a discrete spatial computing medium, without the use of a coordinate system or reference to any ideal or pre-existing global structure. We will end with some speculation on the use of these techniques for the growth of adaptive parallel data-flow programs.

II. Related Work

Discretized computational models of physical particle systems have a long and rich history, from computing pioneer Konrad Zuse's early "Calculating Space" lattice-gas cellular automata [5] to lattice Boltzmann models in physics [6]. In the 1980s, particle systems found uses in computer graphics modeling fluids, fabrics, and other physical objects having complex dynamics. More recently, *Particle Swarm Optimization* methodology [7], developed originally to model social behavior in animals, has found applications in a variety of nonlinear or otherwise irregular optimization problems. Spatial partitioning has been used as a method of organizing data structures to efficiently represent euclidean space [8] and coordination of parallel computing resources for spatial data [9]. Modern GPU processors are capable of handling million-particle systems in real time [10]. However, we are not aware of the prior use of a particle system in a partitioned abstract space for general purpose sorting.

The other algorithm we present draws on research in parallel computing models loosely based on cellular automata, such as Tomassini's *Generalized Automata Networks* [11], and the MIT Center for Bits and Atoms' *Reconfigurable Asynchronous Logic Automata* [12]. These models suggest an inherently parallel approach to fundamental computing tasks that explicitly represents spatial relationships between computing elements. The MIT *Amorphous Computing* project [1], and in particular Coore's *Growing Point Language* (GPL) [2] is another major inspiration for this work. However, the approach presented here is considerably simpler: it does not rely on a gradient propagation mechanism or any pre-existing description of the desired geometric configuration of the system, nor does it require that the underlying network approximate a topological manifold. Unlike classic graphical path-finding methods, such as Dijkstra's Algorithm [13], or

more recent *Ant Colony Optimization* (ACO) approaches like that of Caro and Dorigo [14], we do not seek global optima like shortest paths, but rather paths that remain open to extension, while localizing and limiting exploration of the graph. Wireless sensor network communication protocol research [15], [16] represents another broad source of influence for our work. However, these networks are usually highly application-specific, while our project seeks general principles and methods which can be applied in a variety of applications.

III. Collision Sort

Particle Swarm Optimization [7] is generally used to find global optima in an abstract space, but a similar technique can be used to arrange the particles themselves relative to each other, without direct regard to the properties of the space. As a simple example, consider a one-dimensional particle system, whether discrete [17] or continuous [18], which allows pairwise particle interactions or collisions. If we consider the particles not as physically realistic entities but rather as spatial tokens, or agents, representing data to be ordered, then we may take these interaction events, contrary to real-world physics, as opportunities to *decrease* the entropy of the system, by performing a comparison between the particles and conditionally modifying their velocities based on the results of the comparison. In the examples shown in Fig. 1, the darker particle of the pair deflects or continues always to the left, and the lighter colored to the right, regardless of their initial orientation; particles must have access to the orientation of the sorting axis, although they need not know their exact position.

A closed interval or line segment containing these particles between reflective bounding endpoints (as shown in Fig. 2) will allow them to approach a total linear order in time proportional to the size of the space and the simulated velocity of the particles. However, particles which move too fast may miss opportunities for collisions. If the interval is divided into sub-intervals, each may be assigned to a processor, and these processors' particle data need not be shared except (via a potentially asynchronous communication interface) at the point of partition. This simple principle may be generalized in several ways: for example to multiple dimensions, to multi-particle collisions, or to particle velocities modulated in a more sophisticated manner than just reversing directions. With more complex orderings than the linear one, this technique may potentially be used to coordinate the formation of complex global structure from initially unordered data. The parallel, non-deterministic, and asynchronous nature of collision sort may lend itself to GPU computing, to tiled multi-processor systems having local caches, or (on a larger scale) to dense network computing environments, whether Mesh, Grid or Cloud. With sufficient parallelism, this sorting method can be quite fast, but realistic implementations will probably be nondeterministic not only in terms of the process dynamics, but also in terms of exact final locations when the process is complete: all that can be guaranteed is the relative ordering of data within the space. For example, Fig. 3 shows successive logical time steps in a $10,000$ particle collision sort along two axes corresponding to the redness and blueness of the particles: in a single time step each particle makes one comparison with all other particles that it intersects and jumps to a new location.

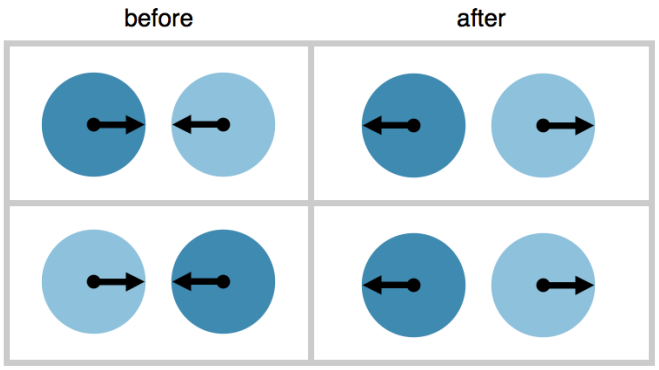

Fig. 1. Simplest possible one-axis particle sorting collision rules. The darker particle of the pair deflects or continues always to the left, and the lighter colored to the right, regardless of their initial orientation.

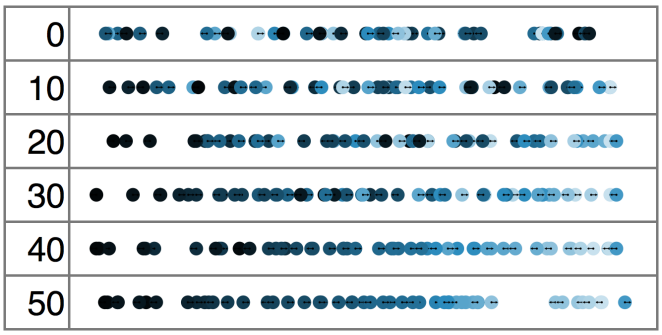

Fig. 2. Time steps in a single-axis pairwise collision sort of 50 particles, in a space 30 particle-widths wide. Each particle has a fixed velocity of 0.5 particle widths per step.

Fig. 4 shows the erformance of the one-dimensional pairwise collision sort with constant velocities and particle counts. The sequences end when the space gets too crowded and particles jump past each other. As one can see, the velocity greatly influences the number of time steps required until the sequence is totally sorted.

In some applications the sorting task may never finish at all, but require continuous sorting on temporal streams of data. Collision sort is well adapted to this job, as it can accommodate any degree of continuous disruption less than its capacity to reduce entropy. We speculate that it may coexist with other particle system rules under some conditions, but have not investigated this possibility in this paper.

IV. Morphogenic Embedding

Following the assumptions of the *Amorphous Computing* project [1], we model a spatial computer as a stochastic mesh network of simple, immobile, reactive computing devices, each connected to all other devices in a circular neighbor-

Fig. 3. Time steps ($t = 0$ (left), $t = 5$ (middle), and $t = 10$ (right)) in a two-axis collision sort with 10,000 particles, each with velocity proportional to the difference between its color and the average color of all other particles which it intersects.

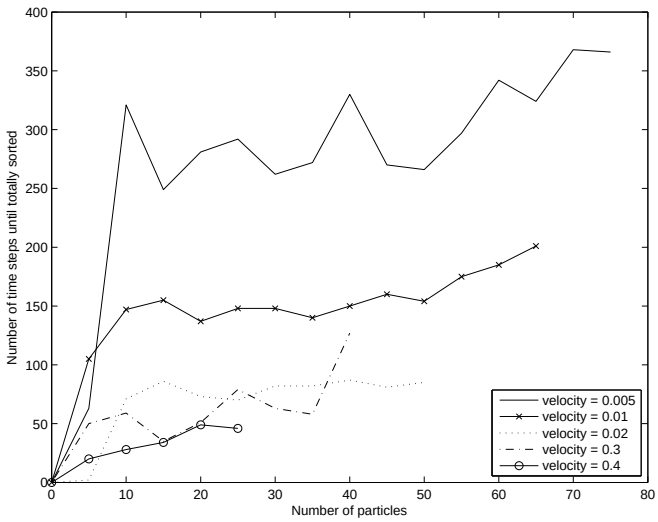

Fig. 4. Performance of the one-dimensional pairwise collision sort with constant velocities and particle counts. The sequences end when the space gets too crowded and particles jump past each other.

hood within a two-dimensional euclidean space, as might be expected in a wireless mesh network of roughly co-planar devices. Devices (or "nodes") in this model are able to store a limited amount of data, to distinguish between their immediate neighbors, and to reliably transmit messages to one or more specific neighbors. They may have methods of discovering the distance to a particular neighbor, but no direct way to measure angles. However, the linear embedding technique we present will work in other kinds of static graphs having an average degree distribution greater than two, although not very efficiently in graph structures optimized for short path lengths, such as trees or stars. The process operates more effectively in graphs with greater connectivity, subject to the costs of communication in such dense networks. The present experimental model conserves the spatial properties of path length and node density, while avoiding dead ends by a limited look-ahead process decribed in the following section. Using a graph that is a stochastic discrete approximation of the euclidean plane allows us to easily visualize the process taking place within it, but this is strictly optional as regards the embedding process itself.

V. Linear Insertion Sort

The algorithm we present is easy to follow (see Fig. 5 and 6); it is an extremely simplified caricature of the cell sorting process which occurs in the development of multicellular organisms [19]. The goal is to incrementally discover (or "grow") a cul-de-sac-avoiding and distance-conserving path in the underlying graph which connects exactly the same number of nodes as the number of items to be sorted, although this number is not known in advance. This path behaves like a linked-list data structure, so we refer to it as a "linkage."

Begin by choosing any inactive node in the meshwork to be the *root* entry point for data to be sorted, represented by atomic tokens, which can be transferred between nodes, but not duplicated. In this example, the data consists of small integers. When the first datum is transmitted from outside the system to the source node, the node merely stores it. For simplicity, we assume that each node has a fixed data storage capacity of just one token, plus another single-token buffer used for communication. While a node's buffer is full, it is not able to accept any new incoming tokens. Given a second numeric datum, a node compares it with the number in its store; the lesser of the two is placed into the store, while the greater is temporarily stored in the node's communication buffer while it recruits one of its inactive immediate neighbors. This *extension* process has two phases: first a message is broadcast to every neighbor, requesting each to respond with a count of its own neighbors. This count may be obtained by another on-demand broadcast and response cycle, or we might assume that there are periodic single-hop broadcast liveness messages sent by each node, which allow all nodes to track the identities and quantity of

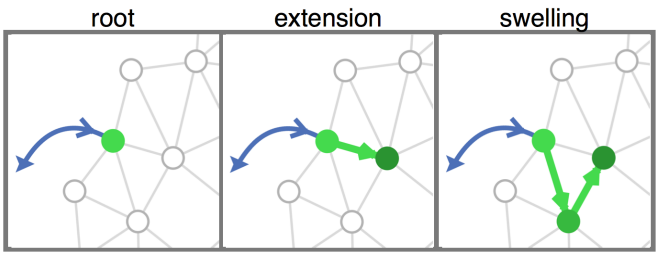

Fig. 5. Operations to embed a linkage within a mesh. The blue arrow indicates the *root* node.

their immediate neighbors. The nearest node with the highest number of inactive neighbors is chosen for recruitment: a channel is established between the currently active node and the newly recruited node, such that data tokens in the buffer of the upstream node are transmitted to the new downstream node when the latter is able to accept them. If the newly recruited node does not already have a copy of the sorting program described here, the program must be also transmitted at the time of recruitment. Each node runs the same program, and each new token received by the root node will eventually result in a new node being recruited into the linkage at its end. Data may be removed from the system by passing a *pull* token to the root node, which responds with the contents of its store and passes the *pull* request downstream, causing data to shift up by a single node; the end node will deactivate upon receipt of a *pull* token, after discharging its store. Fig. 6 illustrates the operations each nodes is executing.

As insertion sort is stable, all data in the stores of the linked nodes is always fully sorted at any time in the process, although tokens must traverse the linkage from buffer to buffer before settling in the appropriate place. This traversal occurs in pipelined parallel fashion, allowing the process to attain linear execution times despite doing work proportional to the square of the number of data items.

The process just described is sufficient to sort any number of data tokens in linear time, as long as the linkage does not run out of space or grow itself into a cul-de-sac. The time steps taken by the embedded insertion sort algorithm to sort a randomized sequence of a given length is shown in Fig. 7. However, we can do better. As it stands, any token displaced from storage in a node will precipitate a ripple of displacement throughout the rest of the structure until a new node is recruited at the end. But, if any two nodes in an existing linkage share a common inactive neighbor not already in the linkage, there is an opportunity for an alternative form of growth we call *swelling*. To find such a helpful mutual neighbor in the appropriate place, an active node n, upon receiving from its immediate upstream neighbor m a new token representing a value less than that in its current store, may request of an inactive neighbor n' to, if it is able to connect to node m, take the new token and ask m to switch its output channel to give input to n'. Node n' also must connect its own output channel back to n,

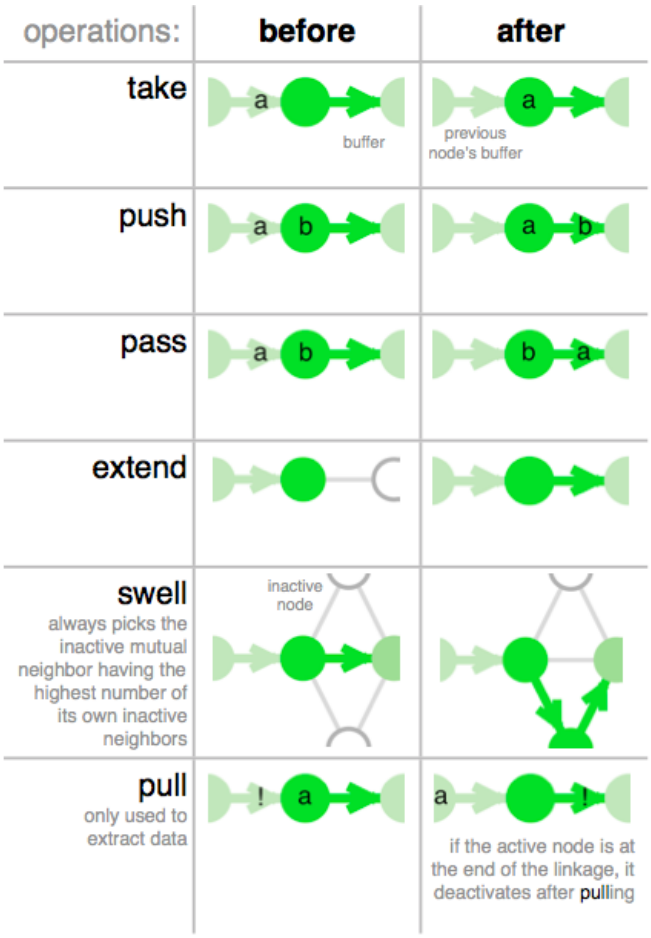

Fig. 6. Illustrations of the operations each nodes is executing.

and then the swelling is complete. With this modification, less overall work is done and the nodes occupied by the algorithm tend to cluster together in the space underlying the graph structure, which may be desirable in minimizing overlap with other component processes in a spatial computer. Clustering also provides an opportunity for shortcuts in the linkage, potentially supporting a version of the skip-list data structure [20]. These shortcuts could be useful in this case because insertion sort maintains the order of its stored data while working, allowing multiple comparisons to be made for a single token by a single linkage node with multiple downstream shortcuts; however, we have not at present implemented skip-list functionality in our model.

VI. Dynamic Dataflow Programs

Sorting is an especially well-studied domain in computer science, and the graph-embedded insertion sort discussed above constitutes a straightforward adaptation of an especially simple sorting algorithm. The method of morphogenetic embedding itself, however, may be applicable to supporting a wider variety of data structures and their associated algorithms within a spatial computer. If we align the component nodes in a regular grid, rather than an irregular

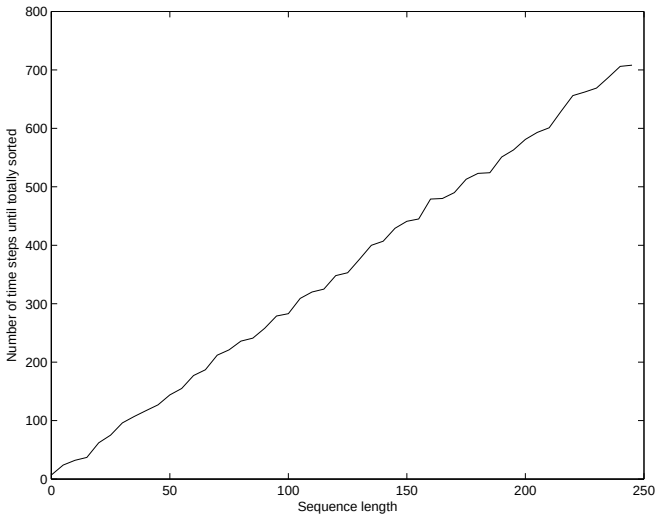

Fig. 7. Time steps taken by the embedded insertion sort algorithm to sort a randomized sequence of given length.

mesh substrate, we would have a general purpose dynamically adaptive, asynchronous data-flow programming system very similar to the *Reconfigurable Asynchronous Logic Array* (RALA) model [12], although at a rather coarser granularity: the RALA system uses tokens to represent only single-bit binary data or a null value, and the individual computing elements perform only boolean functions on these tokens. By contrast, the tokens in the present model represent whole integers, and the nodes are able to execute entire non-trivial programs composed from branching conditionals, comparison, storage, and communication primitives. This work may be seen as bringing some of the ideas of RALA to a spatial computing framework like that modeled by the Proto spatial computing language [21]. Conversely, reconfigurable logic nodes connected in an irregular mesh may be able to support spatial structures like those of Proto with a simpler underlying technology more like that of RALA. In general, a node with degree k may join or switch between $k-1$ branches of an embedded dataflow structure: a node must have at least two neighbors to conduct information through the network. In a regular square lattice, direct three-way branching is possible; a hexagonal lattice can directly support up to five-way branching structures. Irregular or stochastic graphs, such as that shown in Fig. 8, are capable of much higher average degree, although the degree distribution may also be much wider. Fig. 9 shows the longest sequences attained by the insertion sort algorithm when embedded in random graphs having a specific (discretized) Gaussian degree distribution, such as would occur if the nodes were placed randomly into a bounded but edgeless manifold. The higher the mean of the degree distribution, the longer the average maximum linkage length. The standard deviation does not have a significant effect on the linkage length. However, further experimentation and analysis is needed to find the optimum trade-offs in terms of node capabilities, network density and configuration, and programmability for

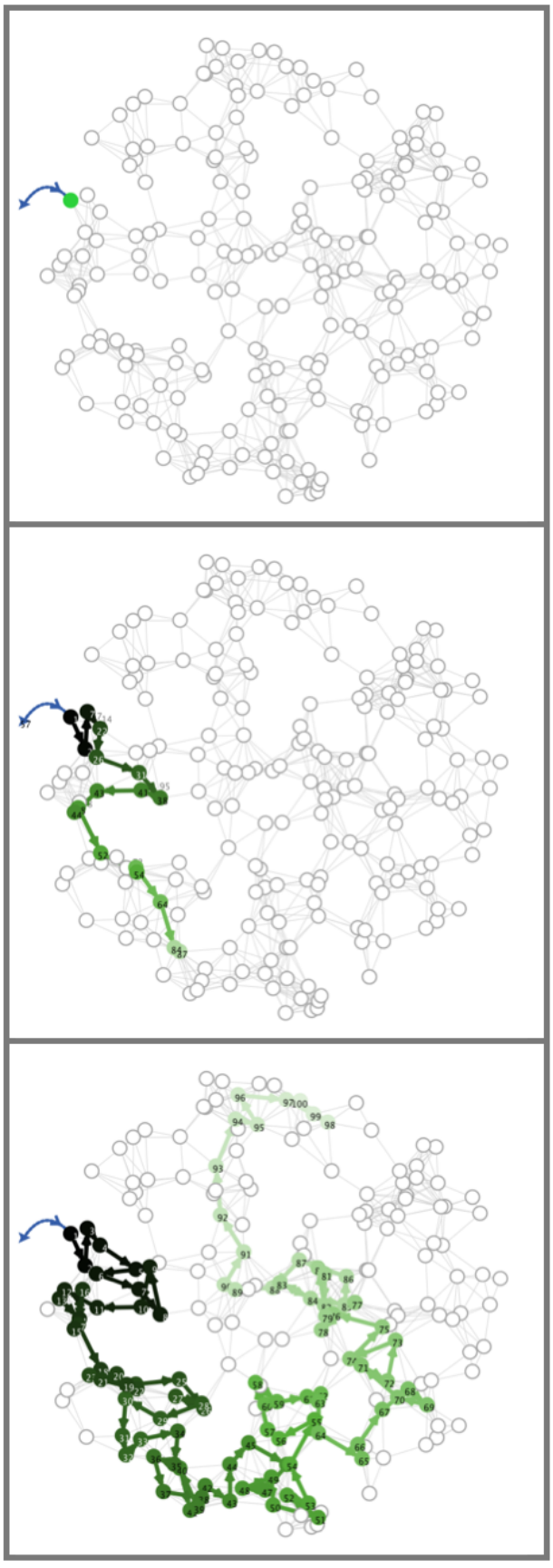

Fig. 8. Steps in the growth of an insertion sort linkage. The blue arrow indicates the *root* node.

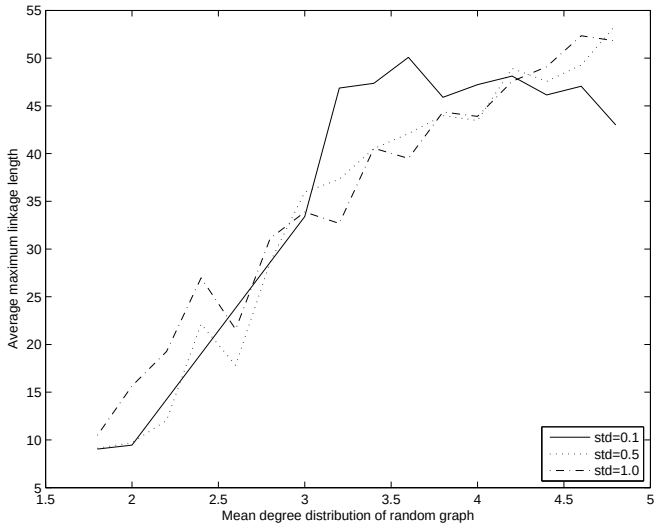

Fig. 9. The longest sequences attained by the insertion sort algorithm when embedded in random graphs having a specific (discretized) Gaussian degree distribution, such as would occur if the nodes were placed randomly into a bounded but edgeless manifold. $N = 100$ nodes. Average over 10 sorting linkages with randomly selected root nodes and over 5 randomly generated graphs for the given Gaussian degree distribution mean and standard deviation.

these types of systems.

VII. Conclusion

Spatial programming techniques show promise for the design of inherently parallel fundamental 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 distance it must span. We have shown two distinct parallel variations on canonical sorting algorithms, and we believe that the techniques we have developed for this purpose, which build on existing research in particle systems and network routing protocols as well as elementary methods in graph theory and algebraic topology, may be applicable to a variety of more sophisticated algorithms and data structures in a wide range of parallel computing contexts.

Acknowledgments

This work was supported by Portland State University's Maseeh College of Engineering and Computer Science Undergraduate Research & Mentoring Program and the PSU Foundation. We are indebted to Uri Wilensky for the NetLogo programming language. Special thanks to Andrew Black, Sam Adams, and David Ungar of IBM Research for provoking and supporting these investigations.

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*, vol. 43, pp. 74–82, May 2000.

[2] D. N. Coore, "Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer," Ph.D. dissertation, Massachusetts Institute of Technology, 1999, aAI0800439.

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

[4] M. Mamei and F. Zambonelli, "Spatial Computing: The TOTA Approach," in *Self-star Properties in Complex Information Systems*, ser. Lecture Notes in Computer Science. Springer Berlin, Heidelberg, 2005, vol. 3460, pp. 360–360.

[5] K. Zuse, "Calculating space," Massachusetts Institute of Technology, Tech. Rep. AZT-70-164-GEMIT, 1970.

[6] S. Chen and G. D. Doolen, "Lattice Boltzmann method for fluid flows," *Annual Review Of Fluid Mechanics*, vol. 30, pp. 329–364, 1998.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, 1995, pp. 1942–1948.

[8] H. Samet, *Applications of Spatial Data Structures*. Reading, MA, USA: Addison-Wesley, 1990.

[9] S. Wang and M. P. Armstrong, "A quadtree approach to domain decomposition for spatial interpolation in grid computing environments," *Parallel Computing*, vol. 29, no. 10, pp. 1481–1504, 2003.

[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*. New York, NY, USA: ACM, 2004, pp. 115–122.

[11] M. Tomassini, "Generalized automata networks," in *International Conference on Cellular Automata for Research and Industry*, 2006, pp. 14–28.

[12] N. Gershenfeld, D. Dalrymple, K. Chen, A. Knaian, F. Green, E. D. Demaine, S. Greenwald, and P. Schmidt-Nielsen, "Reconfigurable asynchronous logic automata: (RALA)," in *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. New York, NY, USA: ACM, 2010, pp. 1–6.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[14] G. D. Caro and M. Dorigo, "AntNet: A mobile agents approach to adaptive routing," IRIDIA, Free University of Brussels, Tech. Rep., 1997.

[15] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 102–114, 2002.

[16] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[17] K. Lindgren and M. G. Nordahl, "Universal computation in simple one-dimensional cellular automata," *Complex Systems*, no. 4, pp. 299–318, 1990.

[18] D. Duchier, J. Durand-Lose, and M. Senot, "Massively parallel automata in euclidean space-time," in *Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*, Sept. 2010, pp. 104–109.

[19] R. Rosen, *Essays on Life Itself*. Columbia University Press, 2000, ch. Morphogenesis in Networks, pp. 224–245.

[20] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, pp. 668–676, June 1990.

[21] M. Viroli, B. Beal, Jacob, and M. Casadei, "Core operational semantics of Proto," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2011, pp. 1325–1332.