Distributed Agreement in Tile Self-Assembly

Aaron Sterling*

Department of Computer Science, Iowa State University. sterling@cs.iastate.edu

Abstract. Laboratory investigations have shown that a formal theory of fault-tolerance will be essential to harness nanoscale self-assembly as a medium of computation. Several researchers have voiced an intuition that self-assembly phenomena are related to the field of distributed computing. This paper formalizes some of that intuition. We construct tile assembly systems that are able to simulate the solution of the wait-free consensus problem in some distributed systems. This potentially allows binding errors in tile assembly to be analyzed (and managed) with positive results in distributed computing, as a "blockage" in our tile assembly model is analogous to a crash failure in a distributed computing model. We also define a strengthening of the "traditional" consensus problem, to make explicit an expectation about consensus algorithms that is often implicit in distributed computing literature. We show that solution of this strengthened consensus problem can be simulated by a two-dimensional tile assembly model only for two processes, whereas a three-dimensional tile assembly model can simulate its solution in a distributed system with any number of processes.

1 Introduction

One emerging field of computer science research is the algorithmic harnessing of molecular self-assembly to produce structures (and perform computations) at the nano scale. In his Ph.D. thesis in 1998, Winfree [19] used tiles on the integer plane to define a self-assembly model which has become an influential tool. As noted by several researchers (for example [1]), problems in algorithmic tile self-assembly share characteristics with better-studied problems in distributed computing: asynchronous computation, the importance of fault tolerance, and the limitations of local knowledge. In this paper, we formalize a connection between the two fields, by constructing models of tile assembly that simulate solutions to the wait-free consensus problem in some distributed systems.

The tile self-assembly literature has considered two main classes of models: models in which tiles bind to one another in an error-free manner, and models in which there is a positive probability that mismatched tiles will bind to one another. In an error-permitting model, if mismatched tiles bind, they can produce a *blockage*—an unplanned tile configuration that stops a particular section of an assembly from being able to accrete tiles.

As blockages do occur in wetlab self-assembly experiments, it is natural to ask how we could make our self-assembly computations as resilient against blockages as possible. Researchers have investigated mechanisms to limit the chance of blockages through error-correction (for example [3] [5]), or, relatedly, for a tile assembly to "heal" itself in the event of damage [14]. Like other error-correcting codes, these mechanisms can consume significant overhead, and only reduce without eliminating the possibility of a blockage. Our interest in this paper, therefore, is to build a framework for robust self-assembly even in the presence of one or more unhealable blockages. Of course, if we consider a situation in which multiple subassemblies grow independently, then the blockage of one subassembly will have no effect on the others. The problem arises when otherwise independent subassemblies send information to, or receive information from, one another, and need to coordinate

* This research was supported in part by National Science Foundation Grants 0652569 and 0728806.

based on that information—hence our motivation to import the consensus problem into the world of tile assembly.

The most common types of processor faults modeled in distributed computing are *crash failure* (where a processor stops functioning) and *Byzantine failure* (where a processor can behave maliciously and take "worst-possible" steps). Several other types of failure have been defined; in general, their severity lies between crash failure and Byzantine failure. We will focus on shared objects that can be simulated in the presence of a tile self-assembly analogue of crash failures, to construct a theoretical foundation for synchronized fault-tolerance in self-assembly. In the long run, we believe that the combination of error-correction and distributed computing techniques (to manage a variety of failures) will produce self-assembling systems with high fault-tolerance.

The consensus problem was originally defined by Lamport for a system of distributed processors, as an abstraction of the transaction commit problem in database theory. It has since been shown to have wide application to the study of distributed systems; see, for example, Attiya and Welch [2] for a textbook introduction. In brief, given a system of n distributed processors, a *solution to the consensus problem* is an algorithm that ensures all nonfaulty processors agree on the same value. (There is also a "validity" condition to ensure the algorithm is not trivial.) The consensus problem for a system of n processors is called n-consensus, and a consensus algorithm is termed *wait-free* if up to $n-1$ processors can crash in an n processor system, and even so all correctly working processors will decide on the same value.

This paper is part of a larger program to connect the fields of self-assembly and distributed computing. By reducing models of self-assembly to models of distributed processors, one can apply known distributed computing impossibility results to obtain limits to the power of self-assembly [16] [17]. On the other hand, by simulating models of distributed computing in self-assembly, one can use strong techniques like ultrametrics to generalize known self-assembly results [18]. The objective of the current paper is to explore which distributed objects can (and cannot) be simulated by self-assembling systems, in order to clarify how positive results of distributed computing can apply to self-assembly.

We have organized the rest of the paper as follows. Section 2 provides further background about tile self-assembly and distributed computing. In Section 3, we construct a tile assembly simulation of wait-free 2-consensus. In Section 4, we define a strengthening of the consensus problem, and show that two-dimensional tile assembly systems cannot simulate solutions to it for systems of three or more processes, but three-dimensional tile assembly systems can. Section 5 concludes the paper and provides suggestions for future research.

2 Background

2.1 Tile self-assembly background

Winfree's objective in defining the Tile Assembly Model was to provide a useful mathematical abstraction of DNA tiles combining in solution in a random, nondeterministic, asynchronous manner [19]. Rothemund [12], and Rothemund and Winfree [13], extended the original definition of the model. For a comprehensive introduction to tile assembly, we refer the reader to [12]. Intuitively, we desire a formalism that models the placement of square tiles on the integer plane, one at a time, such that each new tile placed binds to the tiles already there, according to specific rules. Tiles have four sides (often referred to as north, south, east and west) and exactly one orientation, *i.e.*, they cannot be rotated.

A tile assembly system $\mathcal{T}$ is a 5-tuple $(T, \sigma, \Sigma, \tau, R)$, where T is a finite set of tile types; σ is the *seed tile* or *seed assembly*, the "starting configuration" for assemblies of $\mathcal{T}$; $\tau : T \times \{N, S, E, W\} \to \Sigma \times \{0, 1, 2\}$ is an assignment of symbols ("glue names") and a "glue strength" (0, 1, or 2) to the north, south, east and west sides of each tile; and a symmetric relation $R \subseteq \Sigma \times \Sigma$ that specifies which glues can bind with nonzero strength. In this model, there are no negative glue strengths, *i.e.*, two tiles cannot repel each other.

In this paper, we allow for the possibility of errors in binding between tiles. While, in general, binding errors can cause unplanned configurations to be built, we will make a simplifying assumption that the only binding errors that might occur are *tile blockages*, tile mismatches that prevent any further tiles from binding to the subassembly at which the blockage occurred. In particular, no erroneously bound tile can be enclosed by tiles that attach later in the process of self-assembly.

A *configuration of* $\mathcal{T}$ is a set of tiles, all of which are tile types from $\mathcal{T}$, that have been placed in the plane, and the configuration is *stable* if the binding strength (from τ and R in $\mathcal{T}$) at every possible cut is at least 2. An *assembly sequence* is a sequence of single-tile additions to the frontier of the assembly constructed at the previous stage. Assembly sequences can be finite or infinite in length. The *result* of assembly sequence $\overrightarrow{\alpha}$ is the union of the tile configurations obtained at every finite stage of $\overrightarrow{\alpha}$. The *assemblies produced by* $\mathcal{T}$ is the set of all stable assemblies that can be built by starting from the seed assembly of $\mathcal{T}$ and legally adding tiles. If α and β are configurations of $\mathcal{T}$, we write $\alpha \longrightarrow \beta$ if there is an assembly sequence that starts at α and produces β. An assembly of $\mathcal{T}$ is *terminal* if no tiles can be stably added to it.

We are, of course, interested in being able to *prove* that a certain tile assembly system always achieves a certain output. In [15], Soloveichik and Winfree presented a strong technique for this: local determinism. An assembly sequence $\overrightarrow{\alpha}$ is *locally deterministic* if (1) each tile added in $\overrightarrow{\alpha}$ binds with the minimum strength required for binding; (2) if there is a tile of type t_0 at location l in the result of α, and t_0 and the immediate "OUT-neighbors" of t_0 are deleted from the result of $\overrightarrow{\alpha}$, then no other tile type in $\mathcal{T}$ can legally bind at l; the result of $\overrightarrow{\alpha}$ is terminal. Local determinism is important because of the following result.

Theorem 1 (Soloveichik and Winfree [15]). *If $\mathcal{T}$ is locally deterministic, then $\mathcal{T}$ has a unique terminal assembly.*

2.2 Distributed computing background

Distributed computing began as the study of networks of processors, in which each processor had limited local knowledge. However, much of the distributed computing literature now speaks in terms of systems of *processes*, not processors, to emphasize that the algorithms or bounds obtained from the theorem apply to any appropriate collection of discrete processes, whether they are part of the same multicore chip, or spread across a sensor network.

One of the most-studied distributed computing models is the *asynchronous shared memory model*, in which processes are modeled as finite state machines that can read and write to one or more shared memory locations called *registers*. The model is asynchronous because there is no restriction on when a process will execute its next computation step, except that any nonfaulty process can delay only a finite length of time before taking its next step. We refer the reader to [2] for exposition and technical details of such a model. We now provide a formal definition of the consensus problem for a distributed system.

Definition 1. *Let $\mathcal{M}$ be an asynchronous shared memory model such that each process p_i in $\mathcal{M}$ has two special state components: x_i, the* input*; and y_i, the* decision value*. Let V, the* set of possible decision values*, be a finite set of positive integers. We require that $x_i \in V$ for all i. For each i, y_i starts out containing a null entry, y_i is write-once, and the value written cannot be erased. A* solution to the consensus problem *is an algorithm that guarantees the following.*

Termination *Every nonfaulty process p_i eventually writes a value to y_i.*

Agreement *For any i, j, if p_i and p_j are nonfaulty and write to y_i and y_j, then $y_i = y_j$. All nonfaulty processes decide on the same decision value.*

Validity *If all processes hold the same input value at the start of execution of the algorithm, then any value decided on must be that common input value.*

We will consider *fault-tolerant consensus*, in which one or more processes can fail in some way. The simplest type of failure—and the only type we will consider in this paper—is *crash failure*, meaning that at some point a process ceases operation and never takes another step.

It is a classic result of distributed computing that there is no deterministic algorithm that solves the consensus problem in an asynchronous shared memory model, in the presence of even a single crash failure, when the only registers available to the system are read/write registers [4]. One way to overcome that impossibility result is by making the registers more powerful. These more powerful registers are often called *objects* or *shared objects*, to emphasize they might be a special process segment unto themselves, not just a single memory location. For a comprehensive introduction to the theory of shared objects with different consensus strengths, see [7]. We will use only the following key definitions in this paper.

Definition 2. *Object O* solves wait-free n-process consensus *if there exists an asynchronous consensus algorithm for n processes, up to $n-1$ of which might fail (by crashing), using only shared objects of type O and read/write registers. In distributed system $\mathcal{M}$, O is a* consensus object *if each process in $\mathcal{M}$ can invoke O with a command* `invoke(O,v)`*, where $v \in V$ is a possible decision value, and O will ack with command* `return(O,` v_{out}`)`*, where $v_{out} \in V$ is a possible decision value, and O returns the same value v_{out} to all processes that invoke it. O is an n-*consensus object *if O is a consensus object and n is maximal such that O solves wait-free n-process consensus.*

Finally, we define the notions of configurations and execution segments of a distributed system. Intuitively, we consider the events of a system to be the read and write invocations (and acks) performed upon (and returned by) registers by the processors of the system; and configurations and execution segments are built up from the instantaneous state of the system, and events that are then applied to it.

Definition 3. *Let $\mathcal{M}$ be a distributed system with n processes and one n-consensus object O. A* configuration *of $\mathcal{M}$ is an $(n+1)$-tuple $\langle q_1, \ldots, q_n, o\rangle$, where q_i is a state of p_i and o is a state of O. The* events *in $\mathcal{M}$ are the computation steps by the processes, the transmission of a consensus decision value from O, and the (possible) crashes of up to $n-1$ of the processes. A* legal execution segment *of $\mathcal{M}$ is a sequence of form $C_0, \phi_0, C_1, \phi_1, C_2, \phi_2 \ldots$ where each C_i is a configuration, ϕ_i is an event, and the application of ϕ_i to C_{i-1} results in C_i.*

3 Simulating 2-consensus in two dimensions

We turn now to the simulation in tile assembly of shared objects that solve wait-free consensus problems. For clarity of exposition, we will discuss in detail how to simulate 2-consensus by a two-

dimensional tile assembly system, and in the next section sketch how to extend the simulation to more processes, and to a third tiling dimension.

Winfree showed how to simulate the behavior of a Turing machine by means of a "wedge construction" in tile assembly [19]. Patitz and Summers [11] extended this wedge construction so that the notion of "simulate" would mean that each row of the wedge records the state of the Turing machine tape after one move of the head, and, if the Turing machine halts, a row of tiles is built along the side of the wedge, so a special "halting tile" binds to the base of the wedge, as a marker that the simulation has halted. Inspired by [11], we recently demonstrated a construction whereby tiles can simulate distributed processors in a message-passing system (Theorem 6 of [18]), and we will now modify that machinery, to construct a tile simulation of processors that communicate with shared objects. (For reasons of space, we will not formally define what it means for a wedge construction to "simulate" a distributed processor p, but the intuition is that we add an input buffer and an output buffer to Winfree's original construction by having every other row of the wedge check to see whether a message has been received, and to incorporate the message if one has arrived; and we simulate the sending and receipt of a constant-size number of messages by using a unique tile that encodes the message to build a ray along the edge of the wedge, toward the intended destination of the message, and a ray back along the edge of the wedge to simulate the *ack*—the character that acknowledges receipt of the message.)

Intuitively, the main objective of this section is to construct a tile assembly system that behaves as shown in Figure 1: there are three "modules" or subassemblies: one (π_1) to simulate p_1, another (π_2) to simulate p_2, and a third to simulate a 2-consensus object. In addition, there are "rays" of tiles built from π_1 and π_2, to simulate the values that p_1 and p_2 are writing to the 2-consensus object; and "rays" of tiles built in response, to simulate the acks received by p_1 and p_2 after the writes conclude. The main formal objective in this section is to prove Theorem 2, stated below.

Definition 4. *Tile assembly system $\mathcal{T}$* simulates *distributed system $\mathcal{M}$ if:*

1. *There is a 1-1 mapping h from configurations of $\mathcal{M}$ to stable tile configurations of $\mathcal{T}$.*
2. *If $C_0, \phi_0, C_1, \phi_1 \cdots C_i, \phi_i$ is a legal execution segment of $\mathcal{M}$, then $h(C_0) \longrightarrow h(C_1) \longrightarrow \cdots \longrightarrow h(C_i)$ is a legal tile assembly sequence in $\mathcal{T}$.*
3. *If there is no legal execution segment from C_0 to C_1 in $\mathcal{M}$, then there is no legal tile assembly sequence in $\mathcal{T}$ such that $h(C_0) \longrightarrow h(C_1)$.*
4. *Let C be a configuration of $\mathcal{M}$ and $\mathcal{C}$ be a set of configurations of $\mathcal{M}$. If $\mathcal{M}$ is such that, upon achieving configuration C, it must eventually achieve some configuration $C' \in \mathcal{C}$ unless a process crashes, then $\mathcal{T}$ is such that, if it ever reaches $h(C)$ then it must achieve $h(C')$ for some $C' \in \mathcal{C}$ unless there is a tile blockage. (Note that $\mathcal{C}$ may be an infinite set.)*
5. *If C_0, ϕ_0, C_1, ϕ_1 is a legal execution segment in $\mathcal{M}$, and the event ϕ_0 is the crash failure of a previously correct process, then $h(C_1)$ contains one more tile blockage binding error than $h(C_0)$ contains.*
6. *If in configuration C of $\mathcal{M}$ all processes have halted, then in tile configuration $h(C)$ of $\mathcal{T}$ there are no further locations to which tiles can bind stably,* i.e.*, $h(C)$ is terminal.*

Theorem 2. *Let $\mathcal{M}$ be an asynchronous message passing model of distributed computing with two processes and one 2-consensus object O, such that p_1 and p_2 send no messages to each other, and such that each process invokes O at most once. Then there is a tile assembly system $\mathcal{T}$ that simulates $\mathcal{M}$.*

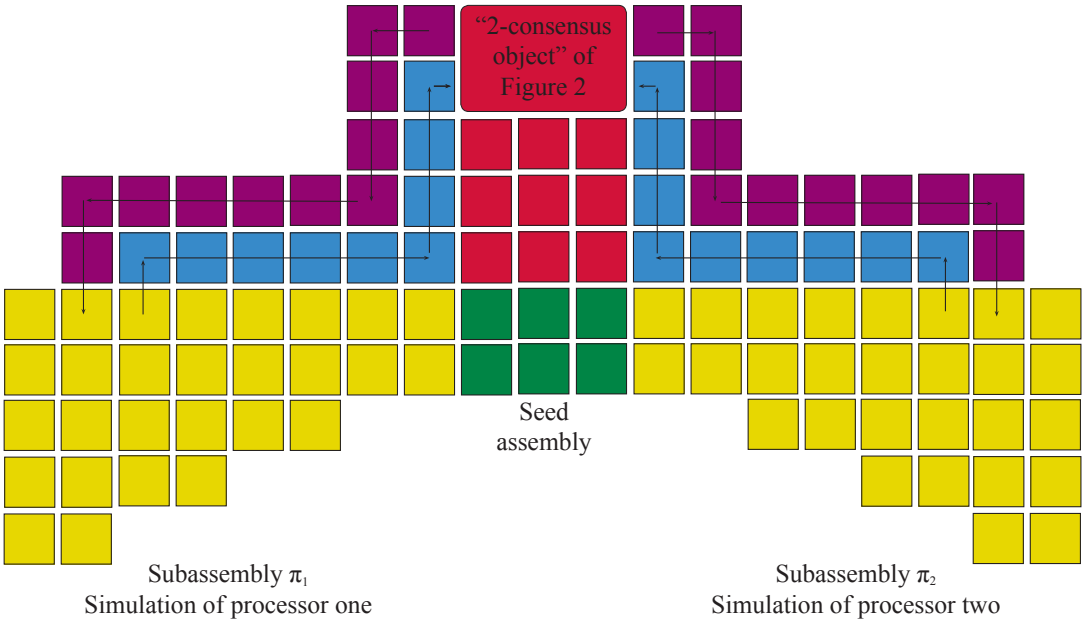

Fig. 1. Two (yellow) subconfigurations π_1 and π_2 growing from a common (green) seed assembly. (The arrows indicate the order in which tiles bind to the assembly.) They communicate with the configuration shown in Figure 2, by means of (blue) "message" tiles they send, and (purple) "ack" tiles they receive back. Intuitively, π_1 and π_2 simulate two processes in a distributed system, communicating with a 2-consensus object.

To prove Theorem 2 we first exhibit a tile configuration that can simulate a 2-consensus object.

Lemma 1. *Fix V any finite set. There is a tile configuration ρ that contains a binding location l with the following properties: (1) the only tiles that bind at l have nonzero glue strenghs at either the south, north and west sides, or the south, north and east sides; (2) any tile that binds at l will have a glue name taken from V on either its east or west side; (3) if glue name $v \in V$ is on the tile that binds at l, then the name of the tile's north glue will be "Ackv," and ρ will build a ray transmitting the glue name Ackv to its west and east.*

We omit the proof of Lemma 1; it essentially describes the construction shown in Figure 2.

Proof (Theorem 2). Fix $\mathcal{M}$, a system of distributed computing per the theorem statement, let V be the set of possible input values to the consensus problem that O might be invoked to solve, and let $k = |V|$. We will design $\mathcal{T}$ so it builds the structure shown in Figure 1.

Using the machinery from the proof of Theorem 6 in [18], let $\mathcal{T}_1$ be a tile assembly system that simulates p_1, $\mathcal{T}_2$ a tile assembly system that simulates p_2, and $\mathcal{T}_3$ a tile assembly system that constructs ρ, a simulator of a 2-consensus object, per Lemma 1. Without loss of generality, we can require $\mathcal{T}_1$ to build in a direction opposite to $\mathcal{T}_2$ (west and east), and for each to build in such a way that they simulate the growth of the Turing machine tape by extending one additional tile to the south at every other column of the construction. We will build $\mathcal{T}$ by extending $\mathcal{T}^*$ to include the simulation of communication between those two processes and the consensus object.

Extend $\mathcal{T}^*$ to $\mathcal{T}$ by adding, for each of the k possible decision values, unique tiles of the form shown in Figure 2(i), so π_1 and π_2 can send rays to ρ, and receive acks as shown in Figure 2(iv). (The

Fig. 2. Diagram of the simulator of a 2-consensus object used to prove Theorem 2. At stage (i), rays from subassemblies π_1 and π_2 approach location l. At stage (ii), a tile binds at l, in this case deciding in favor of the input value of subassembly π_1. At stage (iii), the simulator sends an ack to π_1. At stage (iv), the simulator sends an ack to π_2. "Stage" (v) demonstrates the alternative: a decision tile has bound in favor of the initial value of π_2, and the simulator has acked to π_2.

diagram shows a simulation of binary consensus, with only glue names "0" and "1." We replace those glue names with a total of k distinct names.) We also include in $\mathcal{T}$ the tiles needed so that a subconfiguration that simulates a process can send, and receive, one tile's worth of information. (See [18] for details on how to construct the sending of tiles of information. To receive information back, we use the construction in [18] and ensure the glues on one side are of the form that they accept an ack back, as shown in Figure 2.)

A straightforward induction argument shows that $\mathcal{T}$ does indeed simulate $\mathcal{M}$.

4 Simulating distributed systems with three or more processes

One critical difference between "classical" distributed systems and tile assembly systems is that sending messages—and writing to shared memory locations—in a distributed system does not affect the future computation resources available to processes; whereas in a tile assembly system, the tiles placed on the plane to simulate such operations may "box in" other subassemblies, so they cannot grow beyond some point, due to tile blockages. Put another way, systems of distributed processes have multidimensional resources: each process computes using its own set of resources, and message-passing takes place via a different set of resources. By contrast, every tile operation self-assembles using the same shared resource: the surface. It is, therefore, not surprising that to simulate this resource-independence of distributed systems, tile assembly systems require multiple surfaces, that is to say, three spatial dimensions. Formalizing that is the main objective of this section.

Definition 5. *Let $\mathcal{M}$ be a system of distributed processes such that all correctly-operating processes will run forever. A solution to the* Consensus Subroutine Problem *for $\mathcal{M}$ is an algorithm A that solves the wait-free consensus problem for $\mathcal{M}$ in such a way that no process increases its likelihood of crash failure by calling A as a subroutine.*

If $\mathcal{M}$ has only two processes, then a tile assembly system of the form in Figure 1 will solve the Consensus Subroutine Problem for $\mathcal{M}$, because there is enough space on the surface for π_1, π_2 and the messages to and from ρ to assemble without interfering. For $\mathcal{M}$ with three or more processes, the situation is different. If we attempt to simulate processes that run forever in two dimensions, the tile configurations that simulate the processes must grow without bound. This will cause destructive collision with tiles attempting to communicate information with an n-consensus object.

Theorem 3. *Let $\mathcal{M}$ be a system of n distributed processes ($n \geq 3$) and one n-consensus object, such that all correctly-working processes will run forever. Then no two-dimensional tile assembly system can simulate a solution of the Consensus Subroutine Problem for $\mathcal{M}$.*

We omit the proof from this version of the paper, and instead refer the reader to Figure 3, which presents the key idea pictorially.

Theorem 4. *Let $\mathcal{M}$ be a system of n distributed processes ($n \geq 3$) and one n-consensus object O, such that the processes do not send one another messages, each process invokes O at most once, and all correctly-working processes will run forever. There is a three-dimensional tile assembly system that simulates solution of the Consensus Subroutine Problem for $\mathcal{M}$.*

Proof (Sketch). The construction of Figure 1 was successful because the communication between π_1 and ρ, and π_2 and ρ, did not block the growth of either π_1 or π_2. We duplicate that effect by placing the tile configuration that simulates n-consensus object O *above* the seed assembly (in the

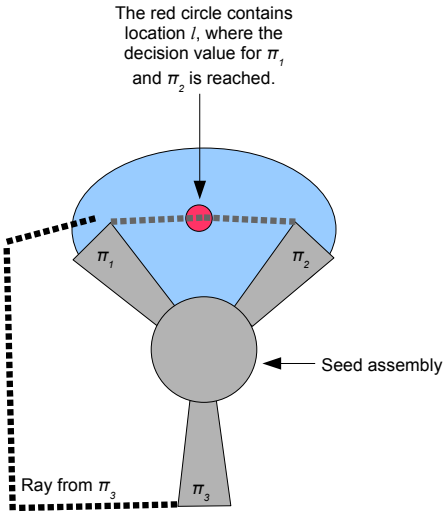

Fig. 3. For π_1 and π_2 to agree on a decision value, they must agree on a value at some location l. If the information at l is tiled outside the blue area before π_1 and π_2 incorporate it into their simulation of p_1 and p_2, it will block further progress of (at least) one of π_1, π_2. So to simulate an execution where p_1 and p_2 take no further steps until p_3 has decided, the decision value must lie inside the blue area. But if π_3 sends a ray into the blue area to obtain the decision value, it permanently blocks growth of π_1 or π_2 (the figure shows π_3 blocking π_1). However, π_3 has no other way to make progress if π_1 and π_2 agree on a value, and then no more tiles bind to π_1 or π_2 until π_3 has committed to a decision. (The point is information-theoretic. If the ray from π_3 connects to a ray from the red circle before a ray from the red circle connects to π_1 or π_2, then π_1 or π_2 will be forever unable to send information *through* that tile configuration.)

z-direction), and requiring that the tiles that simulate processes have glue names on their "tops" (their upward faces in the z-direction) that create paths for transmission of proposed decision values, and reception of acks of those transmissions. The construction of three-dimensional cubes from two-dimensional tiles is a known theoretical technique [8]. We omit the explanation of three-dimensional tiling configurations that simulate n-consensus objects for $n \geq 3$ from this version of the paper. The key idea, however, is that such a construction contains a "decision point," and the message that reaches the decision point first floods as the ack to all processes. So there exists a three-dimensional tile assembly system $\mathcal{T}$ that simulates the Consensus Subroutine Problem for $\mathcal{M}$.

5 Conclusion

We have shown how two-dimensional tile assembly systems can simulate solution to the consensus problem for some two-process distributed systems, and how three-dimensional tile assembly systems can simulate a strengthening of the consensus problem for some n-process distributed systems, for any n. One way to extend our current results would be to consider what types of communication among processes could be simulated by tile assembly.

In this paper, we assumed that the only way a tile simulation could fail was via a blockage that immediately caused the entire subassembly to stop growing. A more extreme failure assumption (analogous to "Byzantine" failures in the language of distributed computing) would be that a subassembly might grow in a malformed, haywire fashion. It would be interesting to see what

application research into Byzantine failures might have to issues of fault-tolerance in tile self-assembly.

Acknowledgements

I am grateful to Jim Lathrop, Jack Lutz and Scott Summers for helpful discussions on earlier versions of this paper. I am especially grateful to Soma Chaudhuri for many helpful discussions.

References

1. S. Arora, A. Blum, L. Schulman, A. Sinclair, V. Vazirani. The computational worldview and the sciences: a report on two workshops. NSF Report, October 2007.
2. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics, second edition.* Wiley Series on Parallel and Distributed Computing, 2004.
3. H-L. Chen, A. Goel. Error free self-assembly using error prone tiles. In: Ferretti C, Mauri G, Zandron C (eds) DNA Computing 10, LNCS 3384. Springer-Verlag, Berlin, pp 111 (2005)
4. M. Fischer, N. Lynch, M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM, 32(2):374-382, April 1985.
5. K. Fujibayashi, D. Y. Zhang, E. Winfree, S. Murata. Error suppression mechanisms for DNA tile self-assembly and their simulation. Natural Computing, published online July 9, 2008.
6. M. Herlihy. Wait-free synchronization. ACM Transactions on Programmng Languages and Systems, 13(1):124-149, January 1991.
7. M. Herlihy, N. Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann 2008.
8. M.-Y. Kao, V. Ramachandran. DNA self-assembly for constructing 3D boxes. LNCS 2223, Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC 2001), P. Eades, T. Takaoka (Eds.), pp. 429-441, 2001.
9. J. Lathrop, J. Lutz, M. Patitz, S. Summers: Computability and complexity in self-assembly. In: Logic and Theory of Algorithms: Proceedings of the Fourth Conference on Computability in Europe, to appear (2008)
10. J. Lathrop, J. Lutz, S. Summers. Strict self-assembly of discrete Sierpinski triangles. In: Computation and Logic in the Real World: Proceedings of the Third Conference on Computability in Europe, pp. 455-464. Springer, Heidelberg (2007)
11. M. Patitz, S. Summers. Self-assembly of decidable sets. Proceedings of the Seventh International Conference on Unconventional Computation (Vienna, Austria, August 25-28, 2008), Springer-Verlag (2008)
12. P. W. K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly.* Ph.D. thesis, University of Southern California, Los Angeles (2001)
13. P. Rothemund, E. Winfree. The program-size complexity of self-assembled squares. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp. 459-468 (2000)
14. D. Soloveichik, M. Cook, E. Winfree. Combining Self-Healing and Proofreading in Self-Assembly. Natural Computing 7(2), pp. 203-218, June 2008.
15. D. Soloveichik, E. Winfree. Complexity of self-assembled shapes. SIAM Journal of Computing, 36(6), pp. 1544-1569 (2007)
16. A. Sterling. A limit to the power of multiple nucleation in self-assembly. LNCS 5218, Proceedings of the International Symposium on Distributed Computing 2008, G. Taubenfeld (Ed.), pp. 451-465, 2008.
17. A. Sterling. A limit to the power of multiple nucleation in self-assembly (full version). Submitted. http://arxiv.org/abs/0902.2422v1
18. A. Sterling. Self-assembly as graph grammar as distributed system. Submitted. http://arxiv.org/abs/0902.2420v1
19. E. Winfree. *Algorithmic Self-Assembly of DNA.* Ph.D. thesis, California Institute of Technology, Pasadena, 1998.