Increased Complexity and Fitness of Artificial Cells that Reproduce Using Spatially Distributed Asynchronous Parallel Processes

Lance R. Williams[1]

[1]Department of Computer Science, University of New Mexico, Albuquerque, NM 87131
williams@cs.unm.edu

Abstract

Replication time is among the most important components of a bacterial cell's reproductive fitness. Paradoxically, larger cells replicate in less time than smaller cells despite the fact that assembling a larger cell requires collecting and combining increased quantities of raw materials. This feat is accomplished through the prodigious use of parallel processing, chiefly, the translation of mRNA into protein by tens of thousands of ribosomes acting in parallel. The massive over expression of ribosomes permits protein synthesis, the limiting step in replication, to occur at a rate and scale that would be otherwise impossible. In computer science, spatial parallelism is the distribution of work across the nodes of a distributed-memory multicomputer system. Despite the fact that a non-negligible fraction of artificial life research is grounded in formulations based on spatially parallel substrates, there have been no examples of artificial organisms that use spatial parallelism to replicate in less time than smaller organisms. This paper describes artificial cells defined using a combinator-based artificial chemistry that replicate in less time than smaller cells. This is achieved by employing extra copies of programs implementing the limiting steps in the process used by the cells to synthesize their component parts. Significant speedup is demonstrated, despite the increased complexity of control and export processes necessitated by the use of a parallel replication strategy.

Introduction

On Earth, the origin of life and its evolution into organisms of increasing complexity was facilitated by our universe's rich physics, the early Earth's bountiful solar, geothermal and chemical energy resources, the massive volume of its prehistoric marine environment, and the vast scale of geologic time. The field of artificial life is premised on the idea that artificial systems can be constructed which, through simplifications and other economies of design, are capable of analogous displays of emergence and open-ended evolution, but on vastly shorter time scales within the memories of digital computers.

Any attempt to demonstrate the evolution of organisms of increasing complexity in an artificial system should begin with an attempt to define complexity itself. Lloyd (2001) gives a list of complexity measures which he divides into three categories: 1) difficulty of description; 2) difficulty of creation; and 3) degree of organization. Unfortunately, for our purposes, measures based on difficulty of description like Kolmogorov (1965) suffer from the drawback that random objects have longer descriptions than non-random objects. This is a problem because the constraints on structure imposed by function ensure that, although complex in a sense we are attempting to define, living things are also far from random.

In Loyd's taxonomy, degree of organization functions as a catch all for measures that attempt to quantify the extent of an object's self-similarity. Bennett (1988) also reviewed measures of complexity, including measures based on self-similarity such as multivariate mutual information, and discussed their suitability for quantifying the complexity of living things. Ultimately, citing the existence of trivial non-living examples, Bennett discounts measures based on self-similarity and proposes a measure based on difficulty of creation. Informally, the *logical depth* of an object is the minimum time required for a *sequential* process to construct the object from its shortest description. Unfortunately, logical depth can be difficult to estimate in practice, since knowing that a process constructs an object from some description in a given number of steps only establishes an *upperbound* on its logical depth. Perhaps more importantly, the size of the object gives a *lowerbound,* since no sequential process can construct an object composed of n primitives in less than n steps. However, any object of size n can be constructed in n steps from a description of length n by a trivial copying process, and objects that can only be constructed this way are effectively *random*.

Proposed measures of complexity based on difficulty of creation and description are not specific to self-replicating entities but apply to objects of all kinds. Moreover, although Bennett's formulation of logical depth is compelling, there is no implied threshold that, if exceeded, would indicate an object's possession of non-trivial complexity. These considerations suggest two qualities specific to self-replicating entities and characteristic of all biological life, yet few (if any) artificial organisms. The first is that living things reproduce by copying and translating compressed self-descriptions. The

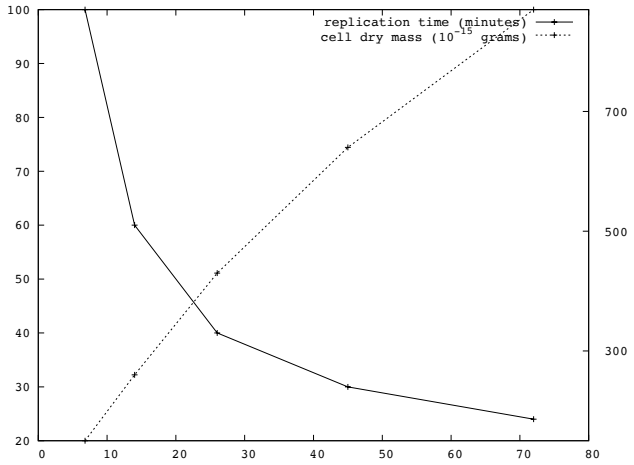

Figure 1: *E. coli* replication time in minutes (left scale) and dry mass in 10^{-15} grams (right scale) as functions of number of ribosomes ($\times 10^3$). All numbers are from Milo and Phillips (2015).

second is that living things reproduce in less time than a sequential process could construct them from an uncompressed description. Although both qualities characterize biological life, and therefore merit investigation, it is the second which is the topic of the present paper.

Replication time is among the most important components of a bacterial cell's reproductive fitness. The faster a bacterium replicates, the more likely it is to outcompete others for space and nutrients. It is for this reason that defining complexity as logical depth seems incompatible with the goal of increasing complexity through natural selection, since it it seems to imply that more complex cells must (by definition) reproduce more slowly. But bacterial cells are constructed using *parallel* not sequential processes. Using parallel construction processes, it is possible for a bacterial cell to both be larger and require less time to reproduce. Indeed, this is precisely what is observed in nature; the growth rate of populations of bacterial cells increases with cell size (Bremer and Dennis, 1996). This would not be possible if bacterial cells did not use parallelism at a massive scale.

Ribosomes are macromolecular complexes made of rRNA which translate mRNA into protein, the most fundamental construction process employed by the living cell. A bacterium's replication time decreases as the fraction of its dry mass consisting of rRNA increases. In the limit, a single *E. coli* can contain 7.2×10^4 ribosomes and rRNA can account for fully 37% of its dry mass, achieving a replication time as short as 24 minutes (Milo and Phillips, 2015). See Figure 1.

All living cells have non-trivial algorithmic complexity since they are all many orders of magnitude more massive than their genomes, which function as compressed self-descriptions. Decompression is accomplished by means of gene expression networks, which permit genes to be expressed at variable rates and in numbers far exceeding their representation in absolute numbers in the genome. Although ribosomes are made of *rRNA* not protein, they are similarly over expressed relative to their representation in the genome. The *E. coli* genome does not have 7.2×10^4 copies of the gene encoding its rRNA; it only has seven (Milo and Phillips, 2015).

Parallelism is not limited to translation of mRNA into protein by ribosomes. It is also used to accelerate duplication of the bacterial genome itself. This process is implemented by a pair of replication forks that copy the circular DNA molecule forming the chromosome of the mother cell, resulting in a $2\times$ speedup. However, Youngren et al. (2014) showed that additional replication forks can begin construction of the granddaughter cell's chromosomes within daughter cells prior to binary fission and before duplication of the mother cell's chromosome has completed, resulting in additional parallel speedup.

Eukaryotic cells exploit parallelism to an even greater extent, by incorporation of complex self-replicating components like mitochondria that reproduce in parallel, and by distribution of their genomes across multiple chromosomes, which are copied and exported in parallel. However, it is indisputable that, with the advent of multicellular organisms, self-replication by means of parallel processes achieved a breakthrough to a qualitatively different level. The existence of organisms as large as elephants and whales would not be remotely possible were it not for the fact that cell division, growth and differentiation are inherently parallel processes.[1]

In summary, we believe that the study of self-replication by parallel processes is critical to achieving the primary goal of the field of artificial life, the demonstration of open-ended evolution of artificial organisms of increasing complexity. Despite this, and despite a long history of research on artificial self-replicating systems (Sipper, 1998; Freitas and Merkle, 2004), there has been relatively little study of systems that replicate using parallel processes. The prior work relevant to parallel self-replication can be divided into five categories:

1. Artificial organisms that replicate using sequential processes implemented on top of synchronous (von Neumann, 1966; Langton, 1984) or asynchronous parallel substrates (Nehaniv, 2004).

2. Artificial organisms implemented on top of asynchronous parallel substrates that replicate using sequential processes but also make limited use of parallelism (Laing, 1977; Hutton, 2004).

[1] One might consider *eusociality* as practiced by social insects (ants) and some mammals (mole rats) to be a continuation of this pattern.

3. Simulations of spatially distributed evolving populations of self-replicating sequential programs (Ray, 1994; Adami et al., 1994).

4. Autocatalytic sets defined in artificial chemistries that are not complete organisms because they do not isolate their reactants and products and/or lack the complexity of complete organisms (Farmer et al., 1986; Nakamura, 2010).

5. Self-replicating parallel programs in shared-memory multiprocessors (Thearling and Ray, 1996).

Nakamura (2010) showed how programs for a universal computer can be encoded as a parallel production system hosted on an asynchronous parallel substrate. He proved important properties of his encoding scheme including freedom from race conditions. It follows that programs which halt yield unique solutions. Finally, he described a parallel asynchronous self-replicating system based on a self-replicating program encoded using his scheme that replicates in time logarithmic in program size. Unfortunately, because no experimental results were described, this speedup can only be considered a theoretical upperbound and an implementation on real parallel hardware would be subject to overhead like non-zero mixing times of reactants and other factors not addressed in the paper.

Thearling and Ray (1996) described evolution of parallel self-replicating programs in a version of Tierra which included a *split* instruction that caused a process to fork into a pair of processes on a shared-memory multiprocessor. Unlike spatial parallelism on a distributed-memory multicomputer, these processes execute in parallel without duplication and distribution of the programs defining the processes. The instruction also modified an index register in a way specific to each daughter process so that the address range was divided between them. The ancestral program was not a *quine*, which replicate using dual *translation* and *copying* processes, but instead copied itself directly by exploiting its residency in the random access shared-memory. Nor was it sequential; it contained a single split instruction that reduced execution time by $2\times$ relative to a non-parallel copying program. Although the evolution of programs with reduced execution times was demonstrated, it should be noted that adding n splits would decrease the execution time by $2^n\times$ of *any* program copying a block of memory of size 2^n on this architecture, and the programs which evolved simply added splits while padding with non-operational instructions so that program length remained evenly divisible by 2^n.

Concurrency and Parallelism

As conceived by von Neumann (1966), a self-replicating system consists of five components

$$A+B+C+D+\phi(A+B+C+D)$$

where A is a *translator*, B is a *copier*, C is a *controller*, D is *payload* and $\phi(A+B+C+D)$ is a *description* of A, B, C and D. Although von Neumann's self-replicating system was formalized as a synchronous cellular automaton, the above schema applies equally well to an asynchronous system of five *actors* which can be viewed as an autocatalytic set in an artificial chemistry governed by two rules:

$$a : A+\phi(A+B+C+D) \;\rightarrow\; 2A+\phi(A+B+C+D)+B+C+D$$
$$b : B+\phi(A+B+C+D) \;\rightarrow\; B+2\phi(A+B+C+D).$$

These rules have reactants on the left side of the arrow and products on the right. The first rule translates the description into a set of *programs* while the second rule copies the description. Each rule describes a *subproblem* which must be solved to achieve self-replication. The reactants and products of subproblems c and d are omitted for the time being, but might (for example) recognize when the a and b subproblems have completed, export products into the daughter cell, and effect fission of an enclosing membrane. For the moment, we assume that the times required to solve c and d are small and, unlike a and b, independent of the length of the description, $\phi(A+B+C+D)$. Nevertheless, $\phi(C)$ and $\phi(D)$ are likely as long as $\phi(A)$ and $\phi(B)$ and (like them) must be both translated and copied.

In computer science, a *concurrent* system consists of subprocesses that can be executed in different orders, constrained only by *data dependencies.* A data dependency exists when one subprocess provides the input to a second, in which case the first process must complete before the second can begin. *Concurrency* is a necessary precondition for *parallelism*, which is the simultaneous execution of subprocesses on different processors. Parallelism imposes additional constraints on execution order since a resource being used by one subprocess cannot be simultaneously used by another. The shared resource in the self-replication problem is the description, $\phi(A+B+C+D)$. It follows that subproblems a and b cannot be solved in parallel, which we abbreviate as a/b. Since there are no data dependencies, the execution order of the a and b subproblems is otherwise unconstrained, so that $a \rightarrow b$ and $b \rightarrow a$ are both possible.

The asynchronous system can be reformulated so that parallel solution of subproblems is possible by splitting the description into four pieces:

$$A+B+C+D+\phi(A)+\phi(B)+\phi(C)+\phi(D).$$

The system is still governed by two rules

$$\begin{aligned} ax : A+\phi(x) &\;\rightarrow\; A+\phi(x)+x \\ bx : B+\phi(x) &\;\rightarrow\; B+2\,\phi(x) \end{aligned}$$

where fx is program f and description x. However, there are now eight subproblems, which we refer to as aa, ab, ac, ad, ba, bb, bc and bd. Because there are no data dependencies between the subproblems, they can be solved sequentially in any of 8! possible orders, including

$$aa \rightarrow ab \rightarrow ac \rightarrow ad \rightarrow ba \rightarrow bb \rightarrow bc \rightarrow bd.$$

To understand the potential for parallel solution of subproblems, the constraints on simultaneous execution imposed by shared resources must first be identified. Execution of a process solving subproblem fx precludes the simultaneous execution of processes solving subproblems fy and gx since there are single instances of program f and description x. This imposes ten constraints on simultaneous execution which we abbreviate as fx/fy and fx/gx. Nevertheless, even with these constraints, many strategies for parallel solution of subproblems remain, including

$$(aa \,|\, ba) \rightarrow (ab \,|\, bb) \rightarrow (ac \,|\, bc) \rightarrow (ad \,|\, bd)$$

which yields a $2\times$ speedup relative to the sequential strategy. Distributing the description across multiple actors increased parallelism by reducing contention for a shared resource. However, it also created new opportunities for parallelism that can be exploited by the addition of extra copies of A, B, $\phi(A)$ and $\phi(B)$. Let n be the total number of copies of A, B, $\phi(A)$ and $\phi(B)$ in an asynchronous self-replicating system:

$$n(A+\phi(A)+B+\phi(B))+C+\phi(C)+D+\phi(D).$$

The system is governed by two rules:

$$\begin{aligned} a_i x_j : A_i + \phi(x_j) &\rightarrow A_i + \phi(x_j) + x_j \\ b_i x_j : B_i + \phi(x_j) &\rightarrow B_i + 2\,\phi(x_j). \end{aligned}$$

It requires the solution of $4(n+1)$ subproblems $a_i x_j$ and $b_i x_j$ to achieve self-replication. The same constraints on simultaneous execution of the increased number of subproblems apply: fx/fy and fx/gx. When $n=2$, these constraints allow the following strategy (and many others of equivalent or lesser efficiency) for parallel solution:

$$(a_0a_0 \,|\, a_1a_1 \,|\, b_0b_0 \,|\, b_1b_1) \rightarrow$$

$$(a_0b_0 \,|\, a_1b_1 \,|\, b_0a_0 \,|\, b_1a_1) \rightarrow (a_0c \,|\, b_0c \,|\, a_1d \,|\, b_1d).$$

Because this strategy contains only three steps, the speedup is $\frac{4}{3}\times$ relative to the non-redundant parallel strategy and $\frac{8}{3}\times$ relative to the sequential strategy. This system is the first in a series of systems where additional instances of the actors which translate and copy can offset the replication cost of control and payload actors leading to decreased self-replication time. Speedup occurs because the work of translating and copying the descriptions of the control and payload actors, $\phi(C)$ and $\phi(D)$, is divided among all instances of the translate and copy actors, A and B. A hypothetical system containing only translate and copy actors

$$n(A+\phi(A)+B+\phi(B))$$

would not benefit from additional A and B instances, since minimum length strategies for all n require two steps:

$$(a_0a_0 \,|\, b_0b_0 \,|\, \ldots \,|\, a_{n-1}a_{n-1} \,|\, b_{n-1}b_{n-1}) \rightarrow$$

$$(a_0b_0 \,|\, b_0a_0 \,|\, \ldots \,|\, a_{n-1}b_{n-1} \,|\, b_{n-1}a_{n-1}).$$

Although the example of the asynchronous von Neumann replicator is compelling, its lack of data dependencies makes it unusually amenable to speedup via parallelism. Indeed, it falls into a category of problems often termed, *embarrassingly parallel.* Many problems cannot be decomposed so easily. Although data dependencies do not necessarily preclude parallel speedup, they sometimes allow only a more limited form of parallelism termed *pipeline parallelism.* Consider an asynchronous self-replicating system governed by the following two rules:

$$\begin{aligned} a_i x_j : A_i + \phi(x_j) &\rightarrow A_i + 2\,\tilde{\phi}(x_j) \\ b_i x_j : B_i + 2\,\tilde{\phi}(x_j) &\rightarrow B_i + x_j + 2\,\phi(x_j). \end{aligned}$$

The first rule consumes a description $\phi(x_j)$ and produces a pair of *reversed* descriptions $2\,\tilde{\phi}(x_j)$. The second rule consumes a pair of reversed descriptions and produces a program x_j and two descriptions $2\,\phi(x_j)$. In effect, the second rule simultaneously translates and copies. Although this process might seem contrived, it is very natural when programs and descriptions are represented as stacks that can be copied in a two stage process using a pair of stacks as an intermediate representation. Far from being contrived, it is the process used by the artificial cell described in this paper.

A self-replicating system using this process for translation and copying contains four data dependencies not present in the embarrassingly parallel system. These can be abbreviated as $bx > ax$, since for all x subproblem bx cannot be solved before subproblem ax has completed. Despite these constraints, for $n=2$, the following strategy is possible

$$(a_0a_0 \,|\, a_1a_1) \rightarrow (a_0b_0 \,|\, a_1b_1 \,|\, b_0a_0 \,|\, b_1a_1) \rightarrow$$

$$(b_0b_0 \,|\, b_1b_1 \,|\, a_0c \,|\, a_1d) \rightarrow (b_0c \,|\, b_1d).$$

This strategy yields a $2\times$ speedup relative to the sequential strategy. Although not quite as large as the $\frac{8}{3}\times$ speedup observed in the embarrassingly parallel system, the speedup in the parallel pipelined system is still significant.

Artificial Chemistry of Program Fragments

An artificial chemistry is a system of constructible objects (Fontana and Buss, 1999). Elemental objects called *atoms* are combined to form more complex objects by reaction rules. When objects are embedded in a two-dimensional space, and are subject to diffusion, the artificial chemistry is a *bespoke physics* (Ackley, 2013). A bespoke physics serves as an abstract interface to a substrate comprised of asynchronous cellular automata (Priese, 1978) that can be simulated in parallel on a distributed-memory multicomputer. Realism can be increased by assuming that atoms cannot be created or destroyed, so that mass is conserved (di Fenizio, 2000). In a typical artificial chemistry, objects are symbols, strings or graphs and the rules which construct them are immutable. However, when objects can represent *programs*

and the immutable rules implement an *interpreter*, an artificial chemistry can (in effect) construct and apply new rules at runtime (Fontana and Buss, 1999; Tomita et al., 2007; Hickinbotham et al., 2011; Buliga and Kauffman, 2014). These artificial chemistries have increased expressive power.

In functional programming, *monads* are an abstract data type representing *program fragments* (Moggi, 1991). The monad interface allows program fragments to be composed and applied to values. By making control idioms implicit in data types, monads make it possible to build simple programs exhibiting complex behaviors, *e.g.,* backtracking. This makes them a powerful tool for defining highly expressive artificial chemistries. In prior work Williams (2016) described an artificial chemistry where atoms are monadic combinators and programs implementing non-deterministic rules are constructed objects. The atoms, which cannot be created or destroyed, are embedded in a two-dimensional space and subject to diffusion. Programs composed of combinators can construct both spatially distributed objects, and non-distributed objects of increased mass. Programs themselves are objects of this second type. Objects of increased mass diffuse more slowly, reflecting the real cost of data transport in the asynchronous parallel substrate. Upon completion, programs report the number of operations they executed. Because this number is a sum over all paths of execution, the real cost of simulating non-deterministic rules on the substrate is paid for. The asynchronous updates of local state needed to simulate both the execution of spatially distributed processes running for different amounts of time and the diffusion of objects of unequal mass are uniformly managed using the Gillespie algorithm (Gillespie, 1977).

Parallel Export

Artificial organisms are more complex than self-replicating systems since, in addition to making their own parts, they must also segregate their parts from the parts of other organisms (McMullin, 2004). Given a device which implements this segregation, the system can become a full-fledged organism by adding $(m \geq 1)$ instances of a program E that moves the products of the mother's translation and copying processes from mother to daughter:

$$n(A+\phi(A)+B+\phi(B))+C+\phi(C)+D+\phi(D)+m(E+\phi(E)).$$

The organism requires an additional rule governing export:

$$e_i\,x_j : E_i + x_j + \phi(x_j) \quad \rightarrow \quad E_i + x'_j + \phi(x_j)'.$$

where x'_j and $\phi(x_j)'$ are a program and its description after export to the daughter organism. Because export of products can only happen after products are completed, two additional constraints on execution order are needed, $e\,x > a\,x$ and $e\,x > b\,x$. Since E exports both a program and its description, the number of steps required to solve subproblem $e\,x$ is twice the number required to solve $a\,x$ or $b\,x$. However, these steps are of unequal size, and although the export process is sequential when $m = 1$, the value of m needed to avoid an export bottleneck depends on the relative cost of primitive synthesis and export operations.

Segregation of an organism's component parts from those of other organisms can be accomplished in different ways. If organisms are defined solely as patterns of states inside compact, connected regions of a parallel substrate, then distance alone can serve this purpose. A daughter organism can be constructed at some offset relative to its mother using the device of a construction arm (von Neumann, 1966; Langton, 1984; Nehaniv, 2004). Unfortunately, the use of this device eliminates the possibility of parallel speedup since it acts as a sequential bottleneck. Irrespective of the fact that it is spatially distributed and uses pipeline parallelism in its transmission of signals, the von Neumann automaton, considered as a whole, is sequential since the daughter automaton is constructed one site at a time at the tip of the moving construction arm.

Moveable organisms require a different solution. The most straightforward is to represent them as the connected components of a graph embedded in the substrate. The work of Hutton (2004) serves as an excellent example. Although this groundbreaking work has influenced the thinking of this author and many others, Hutton's "cells" are probably more accurately described as "molecules" since they consist of a few dozen atoms linked by bonds, lack membranes and genomes, and are copied by the reaction rules of a complex artificial chemistry designed solely for this purpose. The reaction rules implement a sequential and deterministic process that results in two half-sized daughters, which are then completed by means of a growth process that is concurrent and spatially parallel.

In living cells, segregation is accomplished using membranes. Viewed abstractly, membranes are just topological spheres that partition space into two disjoint volumes called *inside* and *outside*. The computational problems associated with the use of membranes as a segregation device were discussed at length in Williams (2019). Although space considerations preclude a detailed reprise of that discussion here, its conclusion was that using membranes to partition space is deceptively complicated, yet also completely unnecessary since simpler methods for representing membership in compact, spatially embedded sets of objects exist.

Williams (2019) describes one such method in detail. The footprint of a *roving pile* is a connected component of a two-dimensional lattice graph. In the combinator-based artificial chemistry described in Williams (2016), actors can form groups which diffuse as a single unit. Groups in the footprint serve as bases for stacks of groups which form the contents of the pile. Base groups with one or more missing edges in the lattice graph form the footprint's *boundary.* Programs in the same stack execute concurrently but not in parallel; they compete for a shared processor resource in zero sum fashion. However, programs in different stacks in the same pile exe-

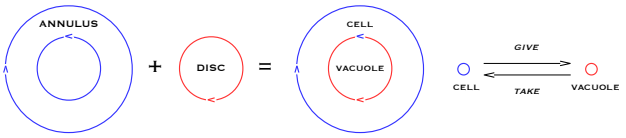

Figure 2: Topologically identifying the inner boundary of an *annulus* and the boundary of a *disc* creates the footprint of a roving pile with two compartments called *cell* and *vacuole*. The boundary between compartments is *selectively permeable.* Transport is mediated by a pair of combinators that act as a *transport interface* (right). These combinators modify *group state.* Groups in the *give* state can move (by diffusion) from cell to vacuole while groups in the *take* state can move in the opposite direction.

cute in parallel. So that piles can move and grow, and so that objects within piles can freely mix, groups in piles undergo diffusion. The footprint's shape evolves because groups on the boundary can move inwards or outwards subject only to the constraint that such movement cannot disconnect the pile's footprint. Although implemented as connected components of a lattice graph, the footprints of a roving pile can be visualized as planar immersions of two-dimensional surfaces with boundaries, and they are depicted this way in the figures of this paper. Observation of a working implementation shows that roving piles remain flat (low average stack height) and connected. Smaller piles constantly evolve in shape while rapidly moving around the space on random walks. Larger piles extend and retract pseudopod-like extensions but are less mobile in aggregate. In summary, roving piles look a lot like living cells.

To achieve the results reported in the present paper, the functionality of roving piles was significantly enhanced beyond that described in Williams (2019). The artificial cells described in this paper are hosted in piles with multiple compartments separated by selectively permeable boundaries; see Figure 2. Using this device, it is possible to implement a parallel export process that resembles binary fission in living cells. See Figure 3 and Figure 4.

Parallel Speedup in Artificial Cells

As McMullin (2004) noted, an artificial cell must accomplish two tasks. First, it must reproduce its own components. Second, the components and their duplicates must be spatially organized into separate cells. If the organization process is symmetrical, then the separate cells are both *daughters.* However, if the organization process is asymmetrical (as in yeast which reproduce by budding) then the two cells are distinguishable and can be called *mother* and *daughter.* We implemented an artificial cell that accomplishes the first of these tasks using a parallel pipeline consisting of six programs *A*-*F* that translate and copy program descriptions; see Figure 5. The second task is accomplished by programs *X*, *Y* and *Z*, which manage the organization of components and duplicates into mother and daughter cells. *X* consists

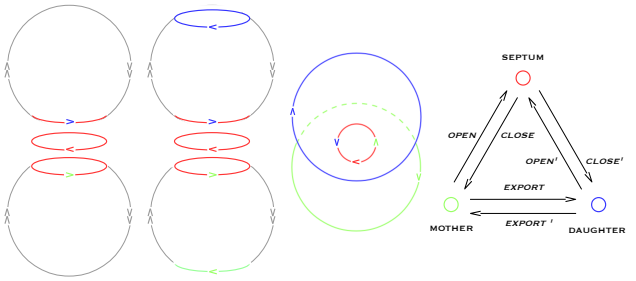

Figure 3: Exploded view in three dimensions of a cell in the process of binary fission showing a pair of circular apertures that pierce daughter cell membranes and the *septum* that divides the cell volume (left). The pair of apertures and the boundary of the septum are topologically identified along the course of the *contractile ring* (red). Adding a second circular aperture to each membrane at the end opposite the septum (blue and green) creates a *topological complex* that can be flattened, *i.e., immersed* in the plane. This complex is the footprint of a roving pile with three compartments called *mother*, *daughter* and *septum*. The boundary shared by the three compartments (red) is selectively permeable. Export from mother to daughter is mediated by a set of combinators that act as an *export interface* (right). *Open* (or *close*) change group states to *open* (or *close*), conferring permission to diffuse from mother to septum (or *vice versa*). This causes the area of the septum to increase (or decrease), which increases (or decreases) the length of the shared boundary, controlling the rate of diffusion from mother to daughter of groups in the *export* state.

of a short executable segment followed by a much longer non-executable segment. When it can do so without splitting the mother's pile, *X* creates a *bud* and moves itself to the newly created daughter's footprint; see Figure 4 (left). It then *smashes* itself within the daughter's footprint, creating the daughter's *cytosol* from the primitives comprising its non-executable segment. The cytosol contains the minimum set of primitive combinators necessary for self-replication. These are consumed by the *A*-*F* pipeline during synthesis and replenished by imports to the pile. *Z* closes the septum after verifying the daughter cell's viability and *Y* is a short non-executable program that is manufactured by the daughter just to prove its viability to *Z*. The lengths of the nine programs comprising the artificial cell are given in Table 1.

Table 1: Program lengths (combinators).

A	B	C	D	E	F	X	Y	Z
42	62	55	63	74	92	89	8	43

A *zipper* is an implementation of a data structure that can be traversed and modified without mutation (Huet, 1997). All zippers consist of three parts. The *front* and *back* represent the parts of the data structure that 1) have already been traversed; or 2) have yet to be traversed. The *focus* is an item between the front and back that can be inspected or replaced.

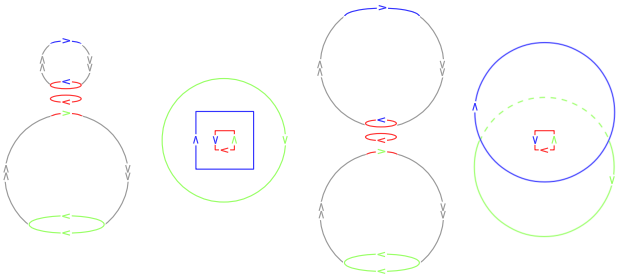

Figure 4: Exploded view of mother cell with *bud* and planar immersion of complex (left). The septum's footprint is a square of size 1×1, while the mother and daughter cell's footprints are annuli with square inner boundaries of size 1×1 (red). The outer boundary of the daughter cell's footprint is a square of size 3×3 (blue). Significantly, the footprints of the daughter and septum can be created using $O(1)$ operations in a single Moore neighborhood of the mother's footprint. Exploded view of mother and daughter cell immediately prior to fission and planar immersion of complex (right). The footprint of the septum is a square of size 1×1, while the footprints of the mother and daughter are annuli with square inner boundaries of size 1×1. The group comprising the septum's footprint can be expelled and the resulting holes in mother and daughter filled using $O(1)$ operations in the septum's Moore neighborhood.

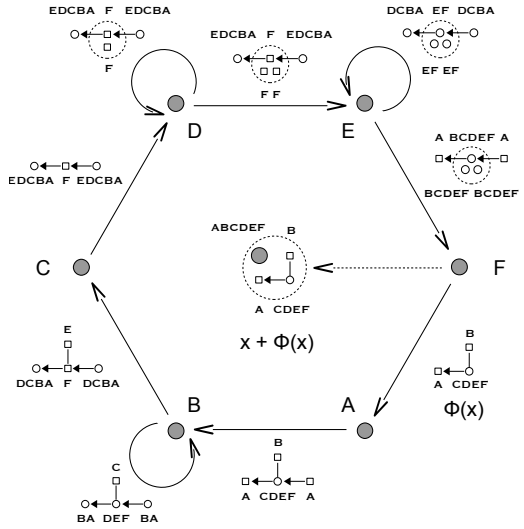

Figure 5: Six stage parallel pipeline for translating and copying zipper representation of description $\phi(x)$ showing changes to zipper conformation effected by programs *A*-*F*. Unlike the pipeline described in Williams (2019), this pipeline produces both program x and description $\phi(x)$ in a single cycle. This permits export to be implemented as a parallel process.

The input to the synthesis pipeline is a zipper representation of $\phi(x)$. A pair of reversed program descriptions $2\,\tilde{\phi}(x)$ can be constructed by traversing this zipper. The process begins when *A* copies the front of the input zipper. The front initially consists of a single combinator. In each step of the traversal, *B* pushes the focus onto the mother front and a primitive from the neighborhood (with matching type) onto the daughter front; the back is then popped to create a new focus. This is repeated until the back consists of a single combinator and the mother and daughter fronts each hold a reversed description $\tilde{\phi}(x)$. These are (in turn) reversed by a second traversal of the zipper in the opposite direction by *E* yielding $\phi(x)$ and $\phi(x)'$. Significantly, during this traversal, *E* also makes an instance of x'. The time required for synthesis by the *A*-*F* pipeline is dominated by stages *B* and *E* which require time proportional to the length of $\phi(x)$. All other stages require constant time.

The last stage in the synthesis pipeline is implemented by *F*, which constructs a group in the *export* state containing x' and $\phi(x)'$ and restores the mother zipper $\phi(x)$ to the conformation needed by *A*, which implements the first stage in the pipeline. However, *A* will not actually see it until after replication completes, since *F* also changes the state of the group containing the mother zipper $\phi(x)$ to *open*. This causes it to migrate to the septum which improves the efficiency of the synthesis process by reducing contention while also preventing the synthesis of unneeded instances of x' and $\phi(x)'$. Translation of $\phi(Y)'$ by A'-F' in the daughter cell causes Z' to migrate to the septum. Once in the septum, Z' returns the zippers $\phi(x)$ stashed there to the mother, where they reenter the *A*-*F* pipeline, restarting the mother's self-replication process. Of course, by this point, the daughter's self-replication process is already in progress; see Figure 6. When Z' is the last actor remaining in the septum, it expels itself from the pile by closing the septum, completing the final operation in the construction of the daughter cell; see Figure 4 (right).

It should be noted that this differs from the process described in Williams (2019) because the *E* stage of the pipeline in that process reversed but did not copy. The advantage of the new pipeline is that both x' and $\phi(x)'$ are produced in a single cycle. Significantly, it is this modification which permits export to be implemented as a parallel process. To appreciate this, recall that the cell described in Williams (2019) used a set of actors contained in a single group as a checklist to ensure that the daughter received the full complement of programs and descriptions. The fact that the checklist was non-distributed yet needed to be updated by any programs implementing the export process necessitated that the process be sequential. The fact that x' and $\phi(x)'$ are exported in a single group in the new cell means that the daughter can demonstrate that it has received the full complement of programs and descriptions simply by using its synthesis pipeline to translate and copy the description of a short dummy program $\phi(Y)'$. Because programs and descriptions are exported in pairs, its ability to do this demonstrates that it possesses not only A'-F', X' and $\phi(Y)'$, but also $\phi(A)'$-$\phi(F)'$, $\phi(X)'$ and Y'. Finally, because Z' closes the septum but migrates there from the daughter, fission cannot occur unless the daughter also possesses $\phi(Z)'$.

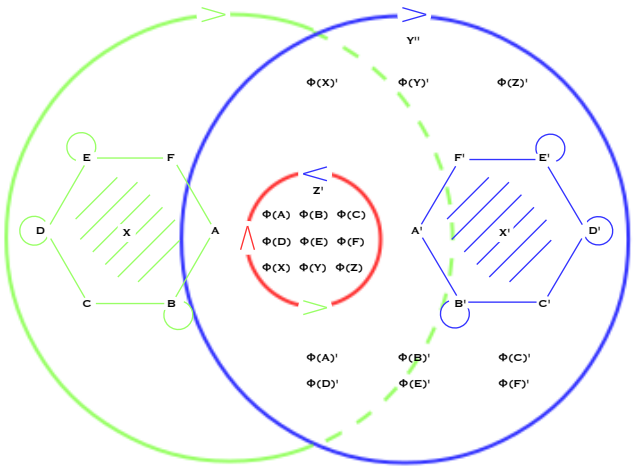

Figure 6: Schematic diagram of artificial cell. Mother (green) and daughter (blue) are depicted immediately prior to fission. After duplication by pipeline implemented by methods A-F, zippers $\phi(x)$ representing the genome migrate to the septum (red) to prevent further duplication. Synthesis of Y'' by A'-F' triggers Z', which returns zippers to the mother while closing the septum (red). Parallel speedup results from extra copies of B and E.

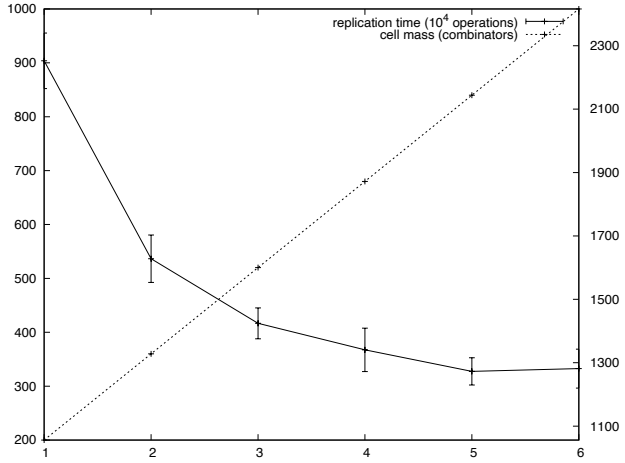

Figure 7: Artificial cell replication time in 10^4 operations (left scale) and mass in combinators (right scale) as functions of number of copies of B and E. Error bars show $\pm\sigma$.

The artificial cell's replication time as a function of the number of copies (n) of the B and E programs and descriptions was measured experimentally. All trials were repeated ten times. Significant decrease in replication time relative to $n = 1$ was observed for $1 < n < 7$, consistent with parallel speedup. This speedup occurred despite the increased size of the artificial cells; see Figure 7. Replication time continued to decrease with increasing n before reaching a minimum value at $n = 5$ followed by a slight increase at $n = 6$.

To ascertain the speedup relative to a sequential implementation of an otherwise identical process, an artificial cell was constructed that combined the A-F programs into a single method together with programs for construction of the daughter cell's cytosol and export. This method and its description formed a group that diffused as a unit within the roving pile. Because of its simplified control and export processes, this sequential artificial cell had a significantly reduced size (734 combinators). Despite its reduced size, the sequential cell's replication time (405×10^4 operations) exceeded the replication times of much larger parallel cells (> 1872 combinators) for $n > 3$. This demonstrates that spatial parallelism can pay for the increased complexity and runtime overhead associated with its use, resulting in artificial cells of increased complexity and fitness.

The export of redundant genes from mother to daughter is not completely reliable. Unlike Nakamura (2010), the parallel computation described in this paper is subject to race conditions that can potentially affect its result. Because they are unnecessary for synthesis of Y'', it sometimes happens that unfinished copies of extra B and E are in the mother when the septum closes. Since X is significantly longer than both B and E, this happens only rarely. However, when it does, the number of copies of the affected gene in the current daughter decreases by one and the number in the next daughter (its younger sister) increases by one. It follows that, in a large population, there will be some variation in n_B and n_E, the number of copies of the B and E genes. Given sufficient time, natural selection would presumably cause the means to converge to values maximizing the fitness tradeoff between increased size (bad) and increased parallelism (good). Characterizing the distribution of n_B and n_E in a large population, and potentially observing their convergence over time to optimum values, is a subject for future work.

Conclusion

Reproduction by spatially distributed asynchronous parallel processes is essential to the long term goal of demonstrating open-ended evolution of artificial organisms of increasing complexity. Despite this fact and the extensive literature on self-replicating systems, there has been little research on systems that replicate using parallel processes. This paper described artificial cells that replicate using spatially distributed asynchronous parallel processes defined using a combinator-based artificial chemistry. Larger cells that reproduce in less time than smaller cells were demonstrated. This was achieved by adding extra copies of programs implementing the limiting steps in the parallel pipelined process used by the cells to synthesize their component parts. Significant speedup was observed, despite the increased complexity of control and export processes necessitated by the use of a parallel replication strategy. The similarity between the parallel speedup observed in artificial cells with increased numbers of programs implementing limiting steps in their component synthesis pipelines and that of bacterial cells with increased numbers of ribosomes is noteworthy.

References

Ackley, D. (2013). Bespoke physics for living technology. *Artificial Life*, 19(3-4):347–364.

Adami, C., Brown, C. T., and Kellogg, W. (1994). Evolutionary learning in the 2D artificial life system "Avida". In *Artificial Life IV*, pages 377–381. MIT Press.

Bennett, C. H. (1988). Logical depth and physical complexity. In *A Half-century Survey on The Universal Turing Machine*, pages 227–257. Oxford University Press.

Bremer, H. and Dennis, P. (1996). Modulation of chemical composition and other parameters of the cell by growth rate. *E Coli Salmonella Cell Mol Biol*, 2.

Buliga, M. and Kauffman, L. (2014). Chemlambda, universality and self-multiplication. In *Proc. of the 14th Intl. Conf. on the Simulation and Synthesis of Living Systems (ALIFE)*, pages 490–497.

di Fenizio, P. S. (2000). A less abstract artificial chemistry. In *Proc. of the 7th Intl. Conf. on the Simulation and Synthesis of Living Systems (ALIFE)*, pages 49–53.

Farmer, J., Kauffman, S. A., and Packard, N. H. (1986). Autocatalytic replication of polymers. *Physica D: Nonlinear Phenomena*, 22(1):50 – 67.

Fontana, W. and Buss, L. W. (1999). What would be conserved if the tape were played twice? In Cowan, G. A., Pines, D., and Meltzer, D., editors, *Complexity*, pages 223–244. Perseus Books, Cambridge, MA, USA.

Freitas, R. A. and Merkle, R. C. (2004). *Kinematic self-replicating machines*. Landes Bioscience.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361.

Hickinbotham, S., Clark, E., Stepney, S., Clarke, T., Nellis, A., Pay, M., and Young, P. (2011). Molecular microprograms. In *European Conference on Artificial Life (ECAL)*, pages 297–304.

Huet, G. (1997). Functional pearl: The zipper. *Journal of Functional Programming*, 7(5):549–554.

Hutton, T. J. (2004). A functional self-reproducing cell in a two-dimensional artificial chemistry. In *Proc. of the 9th Intl. Conf. on the Simulation and Synthesis of Living Systems (ALIFE)*, pages 444–449.

Kolmogorov, A. (1965). Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7.

Laing, R. A. (1977). Automaton models of reproduction by self-inspection. *Journal of Theoretical Biology*, 66(1):437–456.

Langton, C. G. (1984). Self-reproduction in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):135–144.

Lloyd, S. (2001). Measures of complexity: a nonexhaustive list. *IEEE Control Systems Magazine*, 21(4):7–8.

McMullin, B. (2004). Thirty years of computational autopoiesis: A review. *Artificial life*, 10:277–95.

Milo, R. and Phillips, R. (2015). *Cell biology by the numbers*. CRC Press.

Moggi, E. (1991). Notions of computation and monads. *Information and Computation*, 93(1):55–92.

Nakamura, K. (2010). Asynchronous parallel self-replication based on logic molecular model. In *Proc. of the 12th Intl. Conf. on the Simulation and Synthesis of Living Systems (ALIFE)*, pages 403–410.

Nehaniv, C. L. (2004). Asynchronous automata networks can emulate any synchronous automata network. *IJAC*, 14(5-6):719–739.

Priese, L. (1978). A note on asynchronous cellular automata. *Journal of Computer System Sciences*, 17:237–258.

Ray, T. S. (1994). An evolutionary approach to synthetic biology, Zen and the art of creating life. *Artificial Life*, 1:179–209.

Sipper, M. (1998). Fifty years of research on self-replication: An overview. *Artificial Life*, 4(3):237–257.

Thearling, K. H. and Ray, T. S. (1996). Evolving parallel computation. *Complex Systems*, 10(3):229–237.

Tomita, K., Murata, S., and Kurokawa, H. (2007). Self-description for construction and computation on graph-rewriting automata. *Artificial Life*, 13(4):383–396.

von Neumann, J. (1966). Theory of self-replicating automata. *Urbana: University of Illinois Press*.

Williams, L. R. (2016). Programs as polypeptides. *Artificial Life*, 22(4):451–482.

Williams, L. R. (2019). Towards complex artificial life. In *Conference on Artificial Life (ALIFE)*, pages 292–299.

Youngren, B., Nielsen, H. J., Jun, S., and Austin, S. (2014). The multifork Escherichia coli chromosome is a self-duplicating and self-segregating thermodynamic ring polymer. *Genes and Development*, 28(1):71–84.