DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch098

Indefinitely Scalable Computing = Artificial Life Engineering

David H. Ackley[1], Trent R. Small[1]

[1]University of New Mexico, Albuquerque, NM 87131
ackley@cs.unm.edu, tsmall1@unm.edu

Abstract

The traditional CPU/RAM computer architecture is increasingly unscalable, presenting a challenge for the industry—and is too fragile to be securable even at its current scale, presenting a challenge for society as well. This paper argues that new architectures and computational models, designed around software-based artificial life, can offer radical solutions to both problems. The challenge for the soft alife research community is to harness the dynamics of life and complexity in service of robust, scalable computations—and in many ways, we can keep doing what we are doing, if we use *indefinitely scalable* computational models to do so. This paper reviews the argument for robustness in scalability, delivers that challenge to the soft alife community, and summarizes recent progress in architecture and program design for indefinitely scalable computing via artificial life engineering.

The future of software artificial life

Focusing particularly on models that span traditionally separate representational levels, the 'soft alife' (Bedau, 2003) research community has used digital computers to investigate everything from artificial physics and chemistry to artificial biology and ecology. Collectively, the community has sharpened the understanding of life-like systems, deepened the understanding of their range, and offered illuminating examples of their complexities. Alife models and techniques have found applications in contexts like film production (Bajec and Heppner, 2009; Reynolds, 1987) and video games (Grand, 2003, e.g.).

The productivity of the community has been admirable, particularly given its modest size, but we believe a far greater destiny for it lies in store: Software-based artificial life is to be the architectural foundation of truly scalable and robust digital computing. Because, fundamentally, the mechanisms required for robust scalability—to switch energy and perform work, to adapt to local conditions or maintain invariants despite them, to increase parallel processing to suit available resources—are precisely *what life does*.

Future computer programs will be less like frozen entities chained to the static memory locations where they were loaded, and more like yeasty clusters of digital cells, moving and growing, healing and reproducing, cooperating and competing for computing resources on a vast digital landscape—that will itself be growing and changing, as we build and upgrade it even while it's operating. Any program instance will be finite, but the substrate architecture itself will be *indefinitely scalable*, defined as supporting open-ended computational growth without requiring substantial re-engineering (Ackley and Cannon, 2011). An indefinitely scalable machine will ultimately provide an aggregate computational power to dwarf our current visions for even high-performance computing.

To get there from here, we need to reveal, remove, and reimplement design elements that block indefinite scalability. Traditional models of computation, as well as important areas of soft alife research, embody several such assumptions. We need to raise awareness of the costs of such designs, and advocate for existing alternatives, as well as develop new ones.

To get there from here, ultimately a significant societal investment will be required, to back an expanding software artificial life community as it fleshes out a body of scientific and engineering knowledge around robust artificial life for computer architecture. There is much to be invented and discovered, but the payoff will be immense: The development of a tough and savvy computing base of great capability—not a system promising freedom from all risk or fault, but a system for which risk management and fault tolerance have always been inescapable parts of life.

To get there from here, we also need to get going. Recently we have presented a framework called *bespoke physics* to ground the effort (Ackley, 2013a); in addition, we have made the case to communities focusing on operating systems (Ackley and Cannon, 2011), spatial computing (Ackley et al., 2013), and general computing (Ackley, 2013b). Here our purpose is to sound a vigorous call to arms and place a challenge before the soft alife community, and to provide an update on our own recent progress in indefinitely scalable computing via artificial life engineering.

In the rest of this section we expand on the twin challenges—scalability and security—now facing tradi-

tional computer architecture. Section 'Beyond serial determinism' proposes an alternative and grounds it briefly in history, then Section 'Challenges to the soft alife community' highlights common architectural assumptions—such as synchronous updating and perfect reliability—that can yield evocative models in the small, but lead to engineering dead ends in the large. Section 'Programming the Movable Feast Machine' reports progress on our tool-building efforts for indefinitely scalable architecture, along with first benchmarks on (finitely) parallel hardware, and then finally, Section 'A call to action' touches on our next steps and appeals to the soft artificial life community for help.

Computer architecture at a crossroads

The rise of digital computing over the last seventy years has been a stunning feat of technological research and development, with revolutionary economic and societal impacts. But recently the growth rate of traditional serial deterministic computing has plateaued, as further clock speed increases consumed exorbitant resources for diminishing returns. Now 'multicore' machines exchange strict serial determinism for a modicum of parallelism, creating some uncertainty about the exact sequencing of operations, while preserving overall input-output determinism for well-formed programs. But even there, the requirement for *cache coherence*—so the architecture presents a unified Random Access Memory to all processors—is demanding increasingly heroic engineering (Xu et al., 2011, e.g.) even when only considering scalability within a single chip.

At the same time, given recent high-profile computer penetrations and security failures (Harding, 2014; Perlroth, 2013, and many others reported and not), there is underappreciated irony in the computing industry's determined preservation of CPU and RAM architectures, which—by fundamental design—are all but impossible to keep secure. Because programs and data can be placed anywhere in RAM, storage location provides precious little clue to the identity or provenance of its content. And *central* processing means that the *same* tiny spots of silicon run everything—whether code of the long-trusted servant or the drive-by scum of the internet.

Undeniably, serial deterministic computing with CPU and RAM has great strengths: It is flexible and efficient, and its behavior can be predicted accurately by chaining simple logical inferences—which programmers do routinely as they imagine execution of their code. But that predictability exists only so long as hardware and software and user all act as anticipated. If anything amiss is detected, *fail-stop* error handling—that is, halting the machine—is the traditional response. It's game over; no further predictions are required. Fail-stop is efficient to implement and tolerable assuming the only unexpected events are rare faults due to random cosmic rays or other blind physical processes, resulting in nothing more than the occasional system crash without lasting damage or too much lost work.

However, this only applies to small and isolated systems. By contrast, as the high-performance computing (HPC) community contemplates the move to exascale computers, the cost of using fail-stop to preserve program determinism is increasingly seen as untenable (Cappello et al., 2009). And for today's networked CPU/RAM computers, the unexpected is typically neither rare nor random. As app installs and updates bring ever more software bugs, and ever more value at risk attracts ever more malicious actors, the only safe prediction is that *the first flaw loses the machine* to an attacker's control.[1] A horror movie nightmare, where hearing one senseless incantation causes immediate and enduring loss of volition, is quite literally true in our digital environments.

We are now building millions of computers per week according to that staggeringly fragile blueprint. Hardware switches are packed millions to the square millimeter and controlled by a software house of cards—a rickety skyscraper of cards, lashed together by a single thread of execution. It's a devil's bargain that we have accepted, it seems, because its efficiency and flexibility strengths were immediate while its security and scalability weaknesses have overwhelmed us gradually. Now we're so deeply invested in the architecture that we blame only the imperfect tenants, never the doomed buildings: We blame the programmers with their buggy code and the harried managers shipping it, and the miscreants with their malware and the clueless users clicking it. We have accepted this devil's bargain, it seems, because we thought there was no fundamental alternative, or that any alternative would involve unaffordable exotic hardware and space shuttle-grade software.

Beyond serial determinism

There is another approach to building digital computers, a direction suggested decades ago but still mostly unexplored today, that leads to *robustness* instead of fragility. It is built not on static correctness but on dynamic stability, aimed not at efficient completion but at continuous creation, acting not via local changes to a frozen ocean but via collective stabilization of restless seas. It is neither free from faults nor paralyzed by them, but born to them, expecting and accomodating them—even exploiting them.

The proposal is to extract large quantities of robust, useful computation from vast ecosystems of engineered, software artificial life, running on a bulk digital hardware substrate consisting of interchangeable commodity processing tiles suitably (re)architected for the purpose.

This may sound like outrageous science fiction, but—at least in the architecture discussed below—it depends only on conventional electronics manufacturing and requires no

[1] Or, possibly, the *second* flaw, if there's an active 'user' vs 'administrator' distinction.

breakthroughs in materials science or unconventional computing media. Or it may sound outrageously inefficient, but for truly scalable computation, esteem for efficiency must be tempered with respect for robustness. There *will be* undetected faults in memory and processing, and hardware failures too big to mask, and substantial resources going online and off without notice. The system as a whole simply will not have a global start or halt or reset state.

To prosper in such systems, large and long-lived computations will employ strategic redundancy throughout. They will have capabilities for environmental and internal monitoring, as well as for regulation, repair, migration, and opportunistic replication for robustness and performance.

In short: Large computations will become artificial life.

In fundamental ways, the soft alife research community has been exploring in miniature the sorts of issues that the computing industry is destined to encounter in the large—indeed, that it is starting to encounter now. Soft alife should *own* that connection, and if we embrace indefinite scalability in our models, we can.

The proposal is frankly ambitious, but at the same time, in broad strokes it is far from new. Over sixty years ago, von Neumann (1951) predicted that serial deterministic computing—his namesake approach—was destined to change:

> Thus the logic of automata will differ from the present system of formal logic in two relevant aspects:
>
> 1. The actual length of "chains of reasoning," that is, of the chains of operations, will have to be considered.
> 2. The operations of logic . . . will all have to be treated by procedures which allow exceptions (malfunctions) with low but non-zero probabilities.

In other words, von Neumann argued, both the open-endedness of the 'serial' and the perfection of the 'determinism' would have to go. He used a reliability argument and explicitly contrasted fail-stop with the 'hide and heal' error handling of living systems.

At the time, von Neumann guessed it unlikely that computers would grow beyond perhaps tens of thousands of "switching organs"—we would say "logic gates"—using serial determinism. But today's microprocessors—still deterministic, if no longer entirely serial—often sport upwards of a billion gates, putting von Neumann's prediction in the wrong by some six orders of magnitude. Advances in materials, manufacturing, and electronics design have increased gate reliability and reduced costs immensely, and judicious touches of hardware redundancy where needed—for example in error-correcting memories—have, so far, preserved determinism as seen by the application programmer. But we believe application determinism in the large is a lost cause, as suggested by the HPC reference cited above, and it is high time to begin fleshing out alternatives.

The economics and risks of programmability

The universally programmable machine is certainly among the greatest theoretical ideas in all of computer science. Within their physical limits, manufactured digital computers are, in some sense, the most flexible machines possible; that is their glory—and their Achille's heel. There is an inherent coupling between *inflexibility* and security: If some machine simply cannot do something, as a matter of physics, no mode of failure or successful attack can change that.

More flexibility means more accessible actions and thus more risk. For trustworthy performance on high-consequence tasks—automobile engine control, for example—first principles would favor a rigid, special-purpose machine. But the automotive industry—like many others—is moving just the other way, despite the risks, because the economics of programmability is a tidal wave.

A multibillion dollar investment in a new chip fabrication factory today can be attractive because programmability allows one machine design to be used for myriads of purposes—that can be chosen after the hardware is built or sold. Any architecture hoping to supplant CPU and RAM will have to be significantly programmable, or able to generate high volume unit sales some other way.

Though today it is mostly a gleam in the eye, an indefinitely scalable computing architecture, that combines useful programmability with life-like savvy toughness, stands to offer both, and that is our goal.

The Movable Feast Machine

The specific architecture we are developing—called the *Movable Feast Machine* (MFM)—is designed to combine indefinite scalability with robust programmability, and be implementable in mature and cost-effective technologies. Here we describe it only briefly, to ground the discussion in a concrete example, and highlight the challenges those design goals can present for traditional soft alife modeling. Ackley (2013a) offers additional discussion of indefinite scalability, while details of MFM operations are described in (Ackley et al., 2013).

Consider Figure 1, the canonical MFM architecture overview diagram. To a soft alifer's eye, the MFM will most likely read as an artificial chemistry on a cellular automaton, and that is a fair assessment, although the details of both are unusual. The computational state is embodied by 'atoms', which are fixed-length bitvectors existing one per site like a many-valued CA symbol.

An MFM program consists of a finite set of 'elements' and a finite 'Garden of Eden' state. An element definition is analogous to a class definition in an object-oriented system, with the atoms representing object instances. The element definition specifies the interpretation of the bitvector of its atoms—its 'data members'—and provides a 'behavior' method specifying how type of atom reacts with its neighborhood when an asynchronous state transition (an 'event

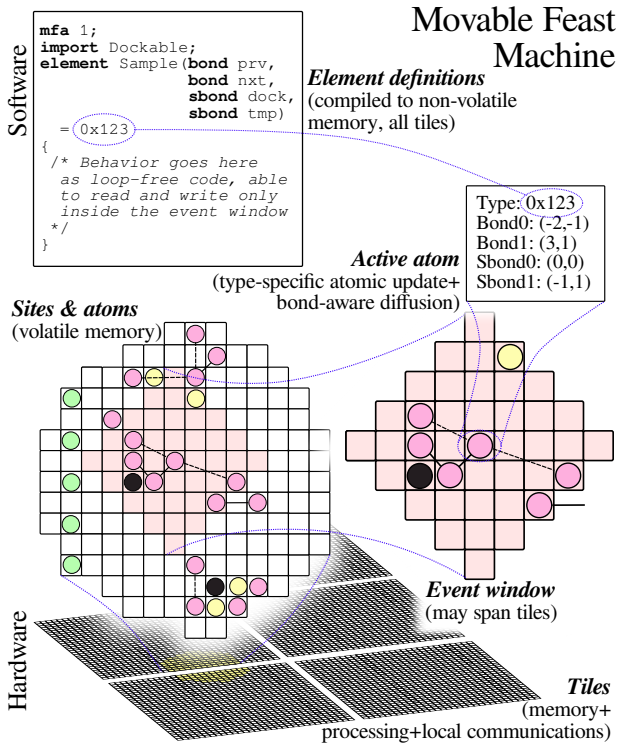


Figure 1: MFM architectural overview.

window' in Figure 1) occurs.

The MFM is unusual as a CA for being asynchronous and employing a relatively huge neighborhood, and its chemistry is entirely discrete, with its reaction rules defined not by transition tables or simple mathematical expressions, but a full-fledged though stylized programming language (originally based on Java, now C++). To enhance robustness, the MFM employs a strict Harvard architecture, placing executable element code in an entirely separate address space from the atoms, so *nothing* that happens during atomic level computations can affect the 'laws of physics' determined by the elements. A separate communications channel, inaccessible to the atoms, is presumed if it is necessary to perform 'magic'—to change or update the compiled element codes (see Ackley et al. (2013) for more).

Challenges to the soft alife community

Several of the MFM architectural decisions—such as the asynchronous updates, and the limited state per atom—are undeniably aggravating for programmers, compared to the heady freedom of serial determinism on CPU and RAM, so it is important to highlight the costs of succumbing to the latter, and the benefits of soldiering on with the MFM. Software-based artificial life, broadly speaking, can be constructed and construed either as 'weak alife'—as life-like computations intended for pedagogy or entertainment or scientific modeling—or as 'strong alife'—as forms of *living technology* (Bedau et al., 2010) deployable to advantage in computer systems other than just the experimenter's.

On the one hand, 'soft weak alife' can employ whatever architectural assumptions its designer wishes, since the ultimate worth of the computation depends only on properties of its output—being instructive, say, or enjoyable or predictive—and not on the computation's internal structures. But on the other hand, once a computational model exists, it is almost impossible to resist imagining it 'writ large': as a scaled-out system capable of actually *doing something*. In other words, 'soft weak alife' contains an implicit affordance to be interpreted as 'soft strong alife'—and that's when its architectural assumptions become crucial.

The game of Life (Gardner, 1970), as likely the best-known instance of a synchronous, faultless cellular automata, provides a good example of the issues and risks. Explicitly positioned as a game, both fun and instructive, its alluringly simple rules and complex resulting behaviors have inspired generations and spawned an entire subculture of obsessed aficionados. And that is vastly more impact than most simple models can claim, but despite its name and its mathematical and metaphorical appeal, the game of Life is a poor architecture for soft strong alife. Concrete evidence for this can be seen in a recent masterful analysis of the game of Life glider, in which Beer (2014) reports that, of the 2^{24} possible states of the sites surrounding a glider, an occurence of any of more than 99.44% of those states destroys the glider. Now that's fragile.

Robust systems and actual living systems have *structural degeneracy*—meaning multiple possible states that can perform the same function, allowing them to ride out many environmental perturbations. As alife researchers, we should want to emphasize architectures and models offering massive degeneracy—and yet, as Beer (2014) notes dryly, "Of course, the structural degeneracy of a glider is quite mild, but this degeneracy can be astronomical in more complicated entities." But in such a synchronous, faultless, deterministic model, everything is predictable in principle, so who needs degeneracy? It's inefficient! We could simulate a bigger model instance if we ditched it! Our architectural assumptions shape our models, more often and more deeply than we may recognize.

Here are five architectural properties—of which the first three, at least, are common in soft alife—that are incompatible with indefinite scalability, often for overlapping reasons:

1. *Global synchronization.* Architectures based on a global clock, or presuming global simultaneous updating, are not indefinitely scalable, requiring either globally flawless execution, faster than light communication, or both. Note the oft-rediscovered trick for overlaying a synchronous CA atop an asynchronous one (Nakamura, 1974; Nehaniv, 2004) presumes perfect reliability—if there's one 'stuck' asynchronous site anywhere, the entire synchronous universe grinds to a halt.

2. *Perfect reliability.* Architectures based on globally flawless execution are not indefinitely scalable, as discussed earlier. Our soft alife models, ideally, should be self-stabilizing (Dijkstra, 1974; Schneider, 1993), but at the very least, some non-zero level of random bit corruptions should be tolerable indefinitely.
3. *Free communication.* Architectures based on fixed or sub-linear latency in global communications are not indefinitely scalable. Physical machinery occupies space and the speed of light is finite, so global communications latency can be no better than linear in the machine diameter. This also rules out tree-based logarithmic communications latencies, and, perhaps less obviously, any local use of (finitely-old) global data—such as conditioning local reproduction decisions on the current global population size, for example.
4. *Excess dimensionality.* Architectures postulating more than three scalable dimensions are not indefinitely scalable, though any number of *finite* additional dimensions is implementable.
5. *Periodic boundary conditions in three dimensions*. Architectures postulating 'wrap around' edges in three scalable dimensions are not indefinitely scalable. In general, edge cases cannot be completely avoided—they can occur due to localized hardware failures, for example—but periodic boundary conditions can be tolerated in models with at most two scalable dimensions.

Again, it is important to stress that by evaluating a soft weak alife model for adequacy as a soft strong alife model, we are going beyond what most designers are prepared to claim for their models—so the blame for any shortcomings, strictly speaking, is on us. But that is not a complete excuse for the model designer, because fragile and unscalable soft weak alife models can tend to mislead our intuitions about the scope and complexity of the rules we should *actually* be considering to make substantial progress. And fundamentally, if one seeks potential designs for soft strong alife, where else should one turn?

There has been an asymmetry in the triumvirate of 'hard', 'soft', and 'wet' alife (Bedau, 2003). The hard alife folk, working in robotics, are obviously constrained by the physical—everything from backlash to friction to irreproducible results. The wet alife folk are likewise constrained by the physical—everything from toxicity to contamination to irreproducible results. Only the soft alife folk were apparently unfettered, but our claim is that, in reality, they too are constrained by the physical, and that indefinite scalability is a suitably abstract but effective way to recognize that.

Programming the Movable Feast Machine

The MFM is an indefinitely scalable architecture designed for research and exploration, and to be readily implementable on low-cost hardware. Our hope is that implementors of artificial chemistries (Dittrich et al., 2001, is a survey) as well as other alife researchers and interested programmers in general, ultimately, can be enticed to work with it. For that to happen, we need to make it readily accessible to all, and that is the goal of our current development work,

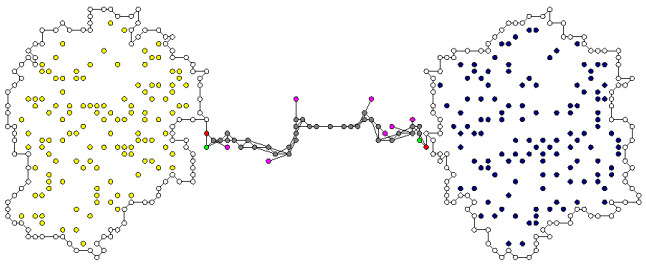


Figure 2: 'Cells' exchanging messages via double-bonded self-healing wire. From (Ackley and Cannon, 2011).

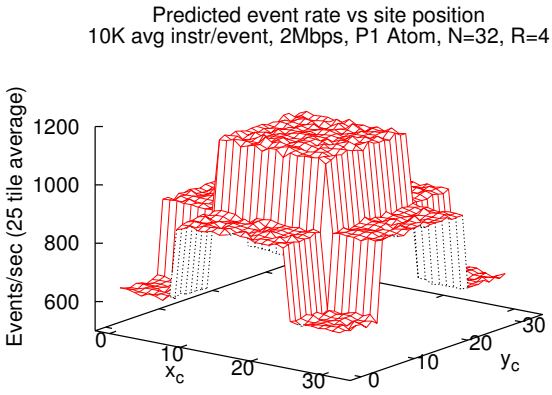


Figure 3: Estimated indefinitely scalable event rate by site, from (Ackley et al., 2013). Compare to Figure 4.

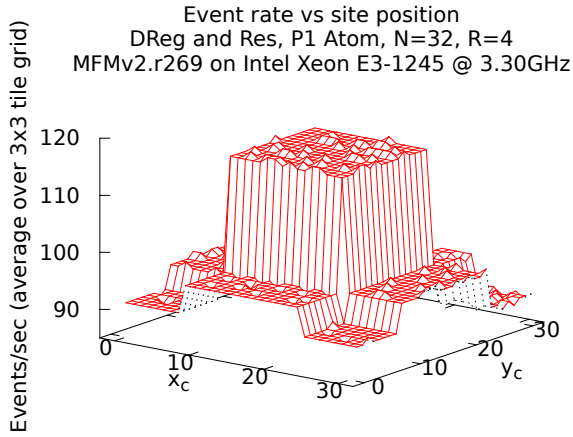


Figure 4: Measured (non-scalable) average event rate by site on `DReg`. Nine tile simulation running on a 4 core/8 hyper-thread processor. Compare to Figure 3 and Figure 5.

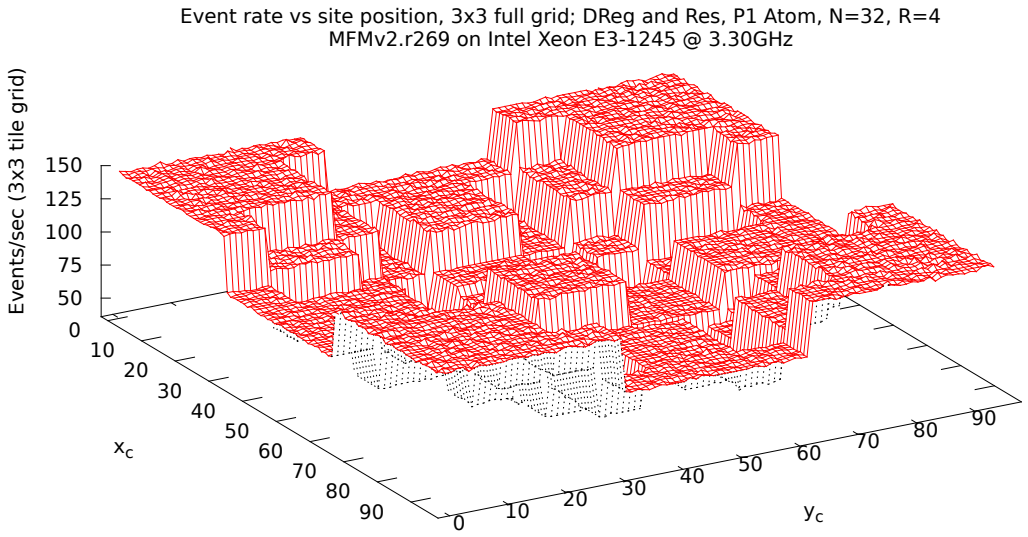


Figure 5: Measured event rates over all sites in a 3×3 grid. Same simulation as in Figure 4.

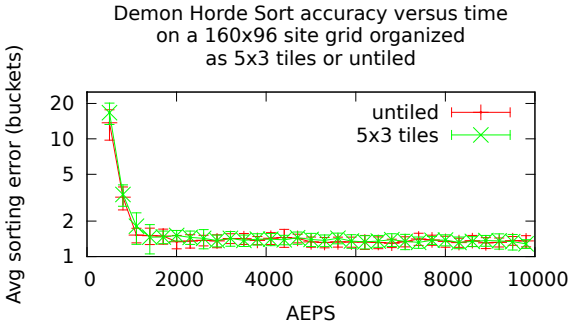


Figure 6: Average sorting accuracy of Demon Horde Sort in a monolithic space vs a 5×3 grid of tiles. See text.

which we describe briefly here.

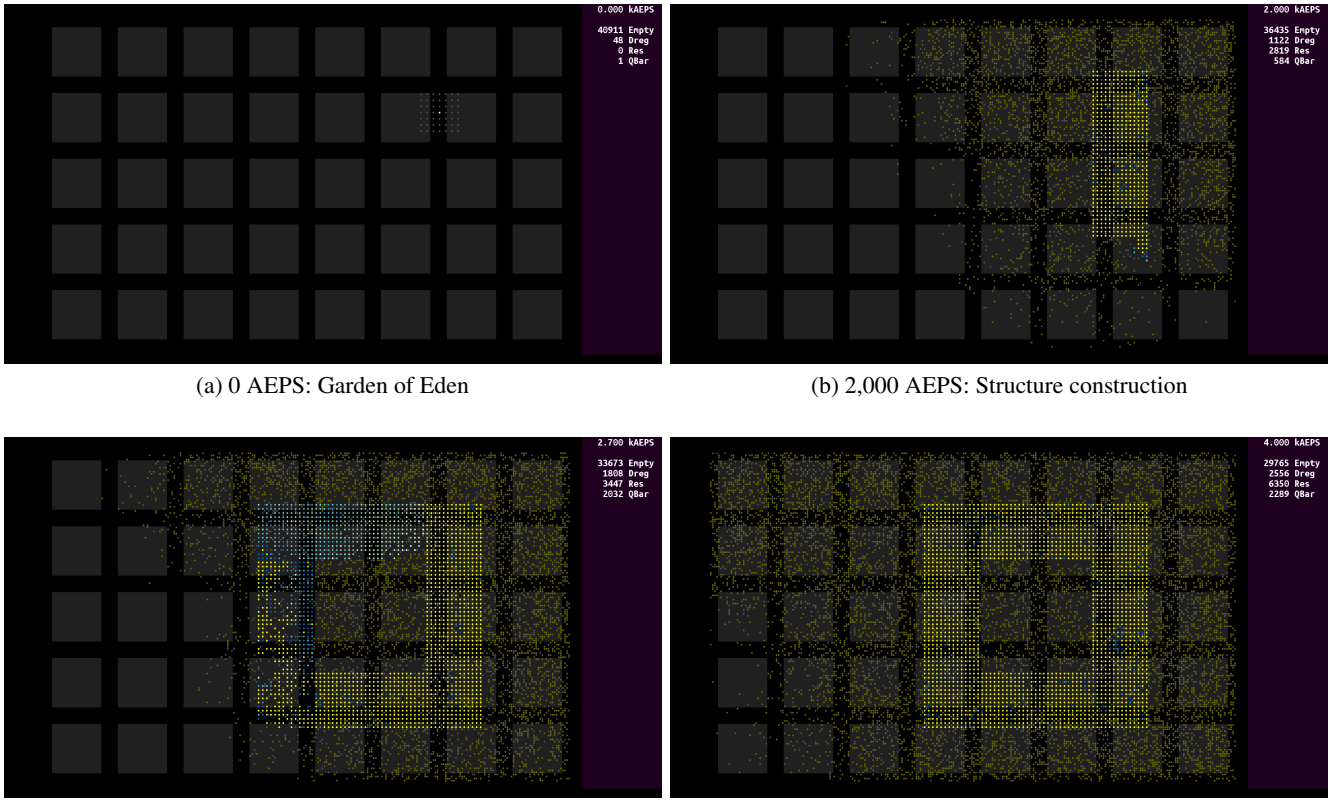
An MFM simulator and operating system

Most MFM simulation results reported in Ackley et al. (2013) and prior—such as the 'communicating cells' model in Figure 2—were obtained with a Java simulator that acted like one big tile. We are now developing a C++ 'MFMv2' codebase, aimed at using the same core event processing code both in simulators running on traditional machines—using one thread per simulated tile—and eventually, in the operating system for actual indefinitely scalable hardware tiles. With this approach, a model built on the finitely-scalable threaded simulator will likely require only a re-compilation to run very similarly on indefinitely scalable hardware—and while in simulation, we have easy global visibility into all tiles for debugging and data gathering.

To aid programmability, the MFM architecture employs intertile locking to make single event execution effectively serial deterministic even if the event window spans tiles. The studies in Ackley et al. (2013) predicted such locking would depress event rates at the edges and corners of tiles, as depicted in Figure 3. In our early experiments with the new multithreaded simulator, we have indeed observed those effects, as seen in Figure 4, which illustrates the average events per second at each site of a tile, averaging over a 3×3 grid of tiles. On average, we observe that events happen about 30% more often in the central 'hidden' regions requiring no communication with surrounding tiles, compared to the edges and especially the corners. However, Figure 5 shows that this 'average tile' is a mixture of two very different situations—central and edge—and the edge tiles actually see significantly higher event rates due to decreased locking.

Such variability in event rate may seem positively debilitating, coming from traditional models in which we are expected to orchestrate the motion of every bit, but indefinite scalability requires the software as well as the hardware to be robust. For example, the `DReg` physics used in the benchmark above (and as a part of the '`QBar`' simulation below), causes random atoms to be deleted every so often, so any computation sharing space with `DReg` must be robust to that. It is possible to consider mechanisms to level the event rate across tile sites, but they will incur a possibly significant performance penalty, and we may find that this hardware-induced event rate variability is better taken as just another factor to which the software stack will adapt.

For example, we have performed a number of studies on a robust sorting algorithm we call the 'Demon Horde Sort', in which `Sorter` atoms move `Datum` atoms across the grid while sorting them vertically by value. This algorithm is described in greater detail in Ackley et al. (2013), but its reported behavior there was based on single-threaded simu-



(a) 0 AEPS: Garden of Eden

(b) 2,000 AEPS: Structure construction

(c) 2,700 AEPS: Structure completion

(d) 4,000 AEPS: Structure maintenance

Figure 7: Stages in building a large structure, showing recycling atoms (`Dreg`, *grey*), resource atoms (`Res`, *brown*), and structure atoms (`QBar`) both immature (from *white* to *blue*) and mature (*yellow*). Assembly begins slowly *(a)*, *(b)* but completes rapidly *(c)* and is maintained indefinitely *(d)* as the system equilibrates. The tile structure is visible in the background (*grey*).

lations. Considering the wide variation of site event rates in these tiles, compared to a single-threaded model, we wondered how much this would affect the Demon Horde's performance. Figure 6 shows early results.[2]

We ran two simulations of the same dimensions and total sites: One on a untiled grid, *'untiled'*, and another on a grid of 5×3 tiles, *'tiled'*. The untiled simulation sees a uniform average event rate at all sites, while the tiled simulation experiences the variable event rates illustrated in Figure 4—and yet there are no obvious differences in sorting accuracy between them. Much more experimentation is needed, but this does suggest it is possible to design useful computations despite significant variabilities in the *neutral dynamics* of the architecture—the spontaneous underlying system behavior independent of any particular program.

Higher order structures

The MFM's explicit programmability makes it relatively easy to create a range of specific collective behaviors, even when the resulting structures are much larger than one atomic neighborhood (the *event window* in Figure 1). As an example, Figure 7 displays screenshots from a simulation that builds a four-sided structure of '`QBar`' atoms, measuring over 100 sites on a side, spread over 20 tiles. Simulated time is measured in *average events per site* (AEPS).

The `QBar` atom uses bits of its atomic state to represent:

- The overall $width$ (5 bits) and $height$ (7 bits) of the 'bar' it is part of,
- Its x_r (5 bits) and y_r (7 bits) relative position in its bar,
- Its $symmetry$ (2 bits) in terms of $90°$ bar rotations, and
- Its $maturity$ represented as a 3 bit stochastic counter.

The `QBar` transition rule performs several actions, including consistency checks involving neighboring `QBar`'s, and seeking out available resources (`Res` atoms) to transmute into further `QBar`'s, if a gap is found in the structure. Also, `QBar` may decay itself back into a `Res`, if it finds itself highly inconsistent with its neighborhood, and as a special case, when a `QBar` atom matures in the maximum position (i.e., when $x_r = width - 1$, $y_r = height - 1$, and

[2]Note the buckets are somewhat coarser in this case compared to prior reports, so the average bucket errors are not directly comparable across implementations.

$maturity = 7$), it seeks to create a minimum position `QBar` atom 'around the corner' using the next rotational symmetry.

The resulting `QBar` 'box' is not particularly useful on its own; it is shown here as an illustration of the kinds of structures that are relatively easy to build in the MFM. Extensions to `QBar` can be readily imagined, such as using the empty sites in its deliberately porous structure to actively transport other selected elements across the bars, generating distinct inner and outer chemical environments. In general, as experience grows with the MFM, we hope to build up libraries of elements and their components—*quarks*—to make bespoke physics design and construction easier and more modular.

A call to action

A major goal of the MFMv2 second generation simulator is to make exploring robust computations with the MFM—for alifers and programmers at least—as painless as possible, to help stimulate the growth of a community and a body of science and engineering knowledge around indefinitely scalable computing via artificial life engineering.

As we finalize this paper for the proceedings, the simulator is still under heavy development, but by the date of the conference we are hoping to have a version 1.0 release, complete with documentation, tutorials and Ubuntu packaging so that installation can be as simple as `apt-get install mfmv2` from a public personal package archive.

But it is not necessary to use our software, or even the MFM architecture itself, to contribute to the larger effort to develop robust-first computing and flesh out alternatives to traditional serial deterministic computing. The power of indefinite scalability, if we define and refine it properly, is that any indefinitely scalable architecture will be transformable into any other of equal or greater scalable dimensionality, at only plausible cost. Or, in particular, transformed into actual hardware at potentially massive scales.

For too long, soft alife models have been separated from each other by their unique laws of physics. But there can be strength in unity. Join us.

Acknowledgments

This work was supported in part by a Google Faculty Research Award, and in part by grant VSUNM201401 from VanDyke Software, both to the first author. Lance R. Williams, Thomas P. Jones, G. Matthew Fricke, and the Robust-First Computing Group at UNM contributed valuable ideas and a place for them to grow.

References

Ackley, D. H. (2013a). Bespoke physics for living technology. *Artificial Life*, 19(3_4):347–364.

Ackley, D. H. (2013b). Beyond efficiency. *Commun. ACM*, 56(10):38–40. Author preprint: `http://nm8.us/1`.

Ackley, D. H. and Cannon, D. C. (2011). Pursue robust indefinite scalability. In *Proc. HotOS XIII*, Napa Valley, California, USA. USENIX Association.

Ackley, D. H., Cannon, D. C., and Williams, L. R. (2013). A movable architecture for robust spatial computing. *The Computer Journal*, 56(12):1450–1468.

Bajec, I. L. and Heppner, F. H. (2009). Organized flight in birds. *Animal Behaviour*, 78(4):777 – 789.

Bedau, M. A. (2003). Artificial life: organization, adaptation and complexity from the bottom up. *Trends in Cognitive Sciences*, 7(11):505 – 512.

Bedau, M. A., McCaskill, J. S., Packard, N. H., and Rasmussen, S. (2010). Living technology: Exploiting life's principles in technology. *Artificial Life*, 16(1):89–97.

Beer, R. D. (2014). The cognitive domain of a glider in the game of life. *Artificial Life*, 20(2):183–206.

Cappello, F., Geist, A., Gropp, B., Kal, L. V., Kramer, B., and Snir, M. (2009). Toward exascale resilience. *IJHPCA*, 23(4):374–388.

Dijkstra, E. W. (1974). Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistries: A review. *Artificial Life*, 7:225–275.

Gardner, M. (1970). The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123.

Grand, S. (2003). *Creation: Life and How to Make It*. Harvard University Press.

Harding, L. (2014). *The Snowden Files: The Inside Story of the World's Most Wanted Man*. Vintage. Knopf Doubleday Publishing Group.

Nakamura, K. (1974). Asynchronous cellular automata and their computational ability. *Systems, Computers, Controls*, 5(5):58–66.

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

Perlroth, N. (2013). Target struck in the cat-and-mouse game of credit theft. *The New York Times*.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA. ACM.

Schneider, M. (1993). Self-stabilization. *ACM Comput. Surv.*, 25(1):45–67.

von Neumann, J. (1951). The general and logical theory of automata. In Jeffress, L. A., editor, *Cerebral Mechanisms in Behaviour: the Hixon Symposium (1948)*. Wiley.

Xu, Y., Du, Y., Zhang, Y., and Yang, J. (2011). A composite and scalable cache coherence protocol for large scale CMPs. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 285–294, New York, NY, USA. ACM.