Simple + Parallel + Local = Cellular Computing

Moshe Sipper

Logic Systems Laboratory, Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland. E-mail: Moshe.Sipper@di.epfl.ch, Web: lslwww.epfl.ch/∼moshes.

Abstract. In recent years we are witness to a growing number of researchers who are interested in novel computational systems based on principles that are entirely different than those of classical computers. Though coming from disparate domains, their work shares a common computational philosophy, which I call *cellular computing*. Basically, cellular computing is a vastly parallel, highly local computational paradigm, with simple cells as the basic units of computation. It aims at providing new means for doing computation in a more efficient manner than other approaches (in terms of speed, cost, power dissipation, information storage, quality of solutions), while potentially addressing much larger problem instances than was possible before—at least for some application domains. This paper provides a qualitative exposition of the cellular computing paradigm, including sample applications and a discussion of some of the research issues involved.

1   What is cellular computing?

The reigning computing technology of the past fifty years, often referred to as the von Neumann architecture, is all but ubiquitous nowadays. Having proliferated into every aspect of our daily lives, the basic principle can be summed up as follows: one complex processor that sequentially performs a single complex task (at a given moment). In recent years we are witness to a growing number of researchers who are interested in novel computational systems based on entirely different principles. Though coming from disparate domains, their work shares a common computational philosophy, which I call *cellular computing*.

At the heart of this paradigm lie three principles:

1. Simple processors, referred to as cells.
2. A vast number of cells operating in parallel.
3. Local connections between cells.

Cellular computing is thus a vastly parallel, highly local computational paradigm, with simple cells as the basic units of computation.

Let us take a closer look at what is meant by these three principles. Firstly, the basic processor used as the fundamental unit of cellular computing—the cell—is simple. By this I mean that while a current-day, general-purpose processor is capable of performing quite complicated tasks, the cell can do very little in and of itself. Formally, this notion can be captured, say, by the difference between a universal Turing machine and a finite state machine. In practice, our

experience of fifty years in building computing machines has left us with a good notion of what is meant by "simple."

The second principle is vast parallelism. Though parallel computers have been built and operated, they usually contain no more than a few dozen processors. In the parallel computing domain, "massively parallel" is a term usually reserved for those few machines that comprise a few thousand (or at most tens of thousands) processors. Cellular computing involves parallelism on a whole different scale, with the number of cells measured at times by evoking the exponential notation, 10^x. To distinguish this huge number of processors from that involved in classical parallel computing, I shall use the term *vast* parallelism (as opposed to "mere" massive parallelism). This quantitative difference leads, as I shall argue, to novel qualitative properties, as nicely captured by the title of a 1972 paper by Philip Anderson "More is Different" [2]. (Note that while not all works presented to date necessarily involve vast parallelism, partly due to current technological limitations, this underlying principle still stands firm).

The third and final distinguishing property of cellular computing concerns the local connectivity pattern between cells. This means that any interactions taking place are on a purely local basis—a cell can only communicate with a small number of other cells, most of which (if not all) are physically close by. Furthermore, the connection lines usually carry only a small amount of information. One implication of this principle is that no one cell has a global view of the entire system—there is no central controller.

Combining these three principles results in the equation *cellular computing* = *simplicity + vast parallelism + locality*. It is important to note that changing any single one of these terms in the equation results in a totally different paradigm; thus, these three axioms represent necessary conditions of cellular computing (Figure 1).

Cellular computing is at heart a paradigm that aims at providing new means for doing computation in a more efficient manner than other approaches (in terms of speed, cost, power dissipation, information storage, quality of solutions), while potentially addressing much larger problem instances than was possible before—at least for some application domains. This paper describes the essence of cellular computing; I provide a *qualitative* exposition, my goal being to convince the reader of the viability of this emerging paradigm. Toward this end I first present in the next section four representative examples of cellular computing, noting that in spite of their differences they all share in common the above three principles. Then, in Section 3 I shall discuss some general issues, followed by concluding remarks in Section 4. (In the full version, I expound upon many of the issues underlying cellular computing, such as properties of the different models, system characteristics, and more [23].)

2   Four examples of cellular computing

To get an idea of what is meant by cellular computing I shall set forth four examples in this section. Though differing in many ways, e.g., the underlying
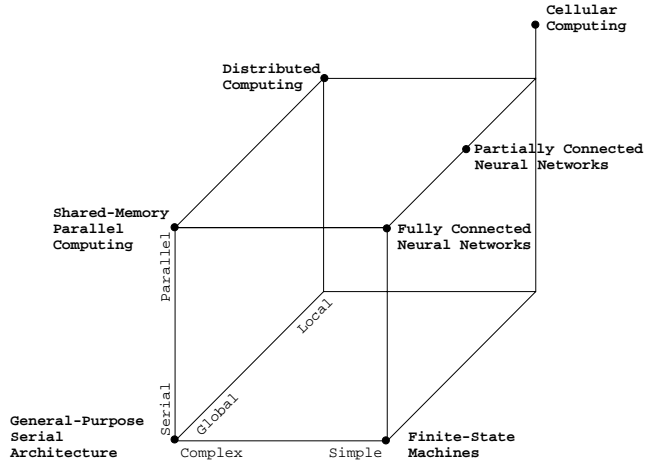
Fig. 1. *Simple + Parallel + Local = Cellular Computing.* Changing any single one of these terms in the equation results in a totally different paradigm, as shown by the above "computing cube." Notes: (1) Cellular computing has been placed further along the parallelism axis to emphasize the "vastness" aspect (see text). (2) Artificial neural networks can be divided into two classes (for our purposes): fully connected architectures, where no connectivity constraints are enforced (e.g., Hopfield networks, Boltzmann machines), and partially connected networks (e.g., the Kohonen network, which exhibits local connectivity between the neurons in the feature map, though each of these is still connected to all input neurons).

model, the problems addressed, input and output encoding, and more, I argue that they all sprout from the common cellular-computing trunk. (To facilitate their referencing each example is given a three-letter mnemonic.)

1. A cellular adder (ADD). Cellular automata are perhaps the quintessential example of cellular computing, as well as the first to historically appear on the scene. Conceived in the late 1940s by Ulam and von Neumann, the model is that of a dynamical system in which space and time are discrete [28,29]. A cellular automaton consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete time steps, according to a local, identical interaction rule. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. This transition is usually specified in the form of a rule table, delineating the cell's next state for each possible neighborhood configuration. The cellular array (grid) is n-dimensional, where $n = 1, 2, 3$ is used in practice.

In a recent work, Benjamin and Johnson [3] presented a cellular automaton that can perform binary addition: given two binary numbers encoded as the initial configuration of cellular states, the grid converges in time towards a final configuration which is their sum. The interesting point concerning this work is

the outline given therein of a possible wireless nanometer-scale realization of the adding cellular automaton using coupled quantum dots. (As pointed out in [3], the device is a nanometer-scale classical computer rather than a true quantum computer. It does not need to maintain wave function coherence, and is therefore far less delicate than a quantum computer).

2. Solving the contour extraction problem with cellular neural networks (CNN). A cellular neural network can be regarded as a cellular automaton where cellular states are analog rather than discrete, and the time dynamics are either discrete or continuous [5–7]. Since their inception, almost a decade ago, they have been studied quite extensively, resulting in a lore of both theory and practice. As for the latter, one major application area is that of image processing. For example, in the contour extraction problem the network is presented with a gray-scale image and extracts contours which resemble edges (resulting from large changes in gray-level intensities). This operation, oft-used as a preprocessing stage in pattern recognition, is but one example of the many problems in the domain of image processing solved by cellular neural networks.

3. Solving the directed Hamiltonian path problem by DNA computing (DNA). The idea of using natural or artificial molecules as basic computational elements (i.e., cells) has been around for quite some time now (e.g., [9, 10]). The decisive proof-of-concept was recently given by Adleman [1] who used molecular biology tools to solve an instance of the directed Hamiltonian path problem: given an arbitrary directed graph the object is to find whether there exists a path between two given vertices that passes through each vertex exactly once. Adleman used the following (nondeterministic) algorithm to solve this hard (NP-complete) problem:

Step 1: Generate random paths through the graph.

Step 2: Keep only those paths that begin with the start vertex and terminate with the end vertex.

Step 3: If the graph has n vertices, then keep only those paths that enter exactly n vertices.

Step 4: Keep only those paths that enter all of the vertices of the graph at least once.

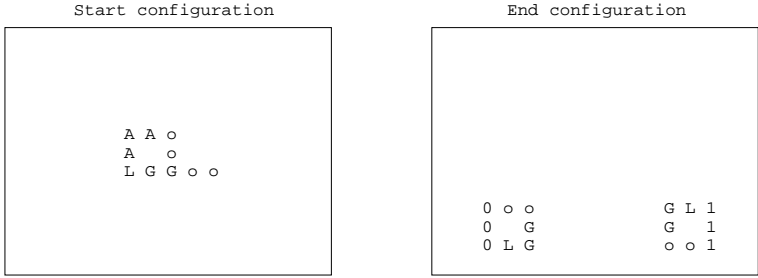*Step 5:* If any paths remain, say "Yes"; otherwise, say "No."

The key point about Adleman's work is the use of DNA material along with molecular biology tools to implement the above algorithm. Vertices and edges of the graph were encoded by oligonucleotides (short chains of usually up to 20 nucleotides). In the test tube these would then randomly link up with each other, forming paths through the graph (step 1), to be then subjected to a series of molecular "sieves" that essentially carried out the remaining steps. The extremely small cell size in this form of cellular computing (a DNA molecule) gives rise to vast parallelism on an entirely new scale. Adleman estimated that such molecular computers could be exceedingly faster, more energy efficient, and able to store much more information than current-day supercomputers (at least with respect to certain classes of problems).

It could be argued that DNA molecules violate the simplicity-of-cells principle. However, while such molecules may exhibit complex behavior from the biologist's standpoint, they can be treated as simple elements from the computational point of view. As put forward by Lipton [17]: "Our model of how DNA behaves is simple and idealized. It ignores many complex known effects but is an excellent first-order approximation." Indeed, it seems that in DNA computing the basic cell (DNA molecule) is typically treated as a simple elemental unit, on which a small number of basic operations can be performed in the test tube [17]. This is similar to several other instances in computer science where irrelevant low-level details are abstracted away; for example, the transistor is usually regarded as a simple switch, with the complex physical phenomena taking place at the atomic and sub-atomic levels being immaterial.

4. Solving the satisfiability problem with self-replicating loops (SAT).

In his seminal work, von Neumann showed that self-replication, previously thought to exist only in nature, can be obtained by machines [28]. Toward this end he embedded within a two-dimensional cellular-automaton "universe" a machine known as a universal constructor-computer, able both to construct any other machine upon given its blueprint (universal construction) and also to compute any computable function (universal computation). Here, the term "machine" refers to a configuration of cellular-automaton states; indeed, the ability to formally describe such structures served as a major motivation for von Neumann's choice of the cellular-automaton model. Self-replication is obtained as a special case of universal construction, when the machine is given its own blueprint, i.e., instructions to build a universal constructor. This latter's complexity prohibited its implementation, and only partial simulations have been carried out to date. Langton [16] showed that if one renounces universal construction, stipulating but self-replication, a much simpler, and entirely realizable structure can be obtained. His so-called self-replicating loop does nothing but replicate. More recently, researchers have shown that one can embed a program within the loop, thus having the structure replicate as well as execute a program [20,27]. The motivation for such programmed replicators is the possibility of obtaining a vastly parallel, cellular computing environment.

Chou and Reggia [4] have recently shown that self-replicating loops can be used to solve the NP-complete problem known as satisfiability (SAT). Given a Boolean predicate like $(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3)$, the problem is to find the assignment of Boolean values to the binary variables x_1, x_2, and x_3 that satisfies the predicate, i.e., makes it evaluate to True (if such an assignment exists). In [20, 27] the program embedded in each loop is copied unchanged from parent to child so that all replicated loops carry out the same program. Chou and Reggia took a different approach in which each replicant receives a distinct partial solution that is modified during replication. Under a form of artificial selection, replicants with promising solutions proliferate while those with failed solutions are lost. The process is demonstrated in Figure 2. This work is interesting in that it can be considered a form of DNA computing in a cellular automaton, using self-replicating loops in a vastly parallel fashion. A

molecular implementation of this approach might be had by using synthetic self-replicators, like those described, e.g., by Rebek, Jr. [21]. Lipton [17] presented a DNA-computing solution to the SAT problem, similar to Adleman's method discussed above. He noted that "biological computations could potentially have vastly more parallelism than conventional ones." [17]

```
        Start configuration                    End configuration
  ┌─────────────────────────────┐       ┌─────────────────────────────┐
  │                             │       │                             │
  │                             │       │                             │
  │                             │       │                             │
  │       A A o                 │       │                             │
  │       A   o                 │       │                             │
  │       L G G o o             │       │                             │
  │                             │       │   0 o o           G L 1      │
  │                             │       │   0   G           G   1      │
  │                             │       │   0 L G           o o 1      │
  │                             │       │                             │
  └─────────────────────────────┘       └─────────────────────────────┘
```

Fig. 2. Solving the satisfiability (SAT) problem with self-replicating loops. Shown above for a three-variable problem: $(\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$. The initial configuration of the two-dimensional cellular automaton contains a single loop with three embedded binary bits (marked by As). This loop self-replicates in the cellular space, with each daughter loop differing by one bit from the parent, thus resulting in a parallel enumeration process. This is coupled with artificial selection that culls unfit solutions, by eliminating the loops that represent them (each loop represents one possible SAT solution). In the end only two loops remain, containing the two truth assignments for the predicate in question: $x_1, x_2, x_3 = 0, 0, 0$ or $1, 1, 1$.

3   Discussion

In this section I shall discuss a number of issues related to cellular computing, ending with a presentation of some possible avenues for future research.

Cellular computing and parallel computing. It could be claimed that the concept of cellular computing is not new at all, but is simply a synonym for the longstanding domain of parallel computing. In fact, the two domains are quite disparate in terms of the models and the issues studied. Parallel computing traditionally dealt with a small number of powerful processors, studying issues such as scheduling, concurrency, message passing, synchronization, and more. This clearly differs from cellular computing, whose underlying philosophy is quite distinct from that of parallel computing (Figure 1). The only area of intersection may be the few so-called "massively parallel machines" that have been built and studied by parallel computing practitioners (e.g., [14]). As noted in Section 1, cellular computing has the potential of exhibiting vastly parallel computation, giving rise to an entirely new phenomenology. (Interestingly, models

such as cellular automata were usually regarded by hard-core parallel computing practitioners as being "embarrassingly parallel" and therefore uninteresting.)

Considering the domain of parallel computing one can observe that decades of research have not produced the expected results—parallel machines are not ubiquitous and most programmers continue to use sequential programming languages. I believe that one of the major problems involves the domain's ambitious goal (at least at the outset) of supplanting the serial computing paradigm. The parallel computing lesson for cellular computing practitioners might thus be that they should not aim for an all-encompassing, general-purpose paradigm, which will replace the sequential one (at least not at present...); rather, one should find those niches where such models could excel. There are already a number of clear proofs-of-concept, demonstrating that cellular computing can efficiently solve difficult problems.

Next, I wish to discuss what I call the *slow fast train*. Consider a 300-kmh fast train arriving at its destination, with passengers allowed to disembark through but a single port. This is clearly a waste of the train's parallel exit system, consisting of multiple ports dispersed throughout the train. This metaphor, dubbed the slow fast train, illustrates an important point about parallel systems, namely, their potential (ill-)use in a highly sequential manner. Note that for most cellular computing models, it is not too difficult to prove computation universality by embedding some form of serial universal machine. This proves that, *in theory*, the model is at least as powerful as any other universal system. However, *in practice*, such a construction defeats the purpose of cellular computing by completely degenerating the parallelism aspect. Thus, on the whole, one wants to avoid slowing the fast train.

Cellular computing and complex systems. In recent years there is a rapidly growing interest in the field of complex systems [11, 15, 19]. While there are evident links between complex systems and cellular computing it should be noted that the two are not identical, the former being a scientific discipline, the latter primarily an engineering domain. As is the time-honored tradition of science and engineering, fruitful cross-fertilization between the two is manifest.

Research themes. Finally, I wish to outline a number of themes that present several possible avenues for future research.

– As noted in Section 1, cellular computing is a computational paradigm that underlies a number of somewhat disparate domains. In this respect, we wish to gain a deeper understanding of the commonalities and differences between the different approaches. Among the important questions are: What classes of computational tasks are most suitable for cellular computing? Can these be formally defined? Can informal guidelines be given? Can we relate specific properties and behaviors of a certain model with the class of problems it is most suitable for? And, vice versa, for a given problem (or class of problems), how do we choose the most appropriate cellular model?
– Adaptive programming methodologies for cellular computing, including evolutionary algorithms [13, 22] and neural-network learning [8].
– Most real-world problems involve some degree of global computation. Thus,

    understanding how local interactions give rise to global ("emergent") computational capabilities is a central research theme [12]. Furthermore, it is important to explore the application of adaptive methods to the programming of such behavior [18, 22].

– What are the major application areas of cellular computing? Research to date has raised a number of possibilities, including: image processing, fast solutions to some NP-complete problems, generating long sequences of high-quality random numbers [24,25]—an important application in many domains (e.g., computational physics and computational chemistry), and, finally, the ability to perform arithmetic operations (e.g., ADD example) raises the possibility of implementing rapid calculating machines on a very small (nano) scale.

– The scalability issue. Cellular computing potentially offers a paradigm that is more scalable than classical ones. This has to do with the fact that connectivity is local, and there is no central processor that must communicate with every single cell; furthermore, these latter are simple. Thus, adding cells should not pose any major problem, on the surface. However, in reality this issue is not trivial both at the model as well as the implementation level. As noted by Sipper [22], *simple* scaling, involving a straightforward augmentation of resources (e.g., cells, connections), does not necessarily bring about *task* scaling, i.e., maintaining of (at least) the same performance level. Thus, more research is needed on the issue of scalability.

– Fault tolerance. Lipton [17] noted that: "The main open question is, of course, if one can actually build DNA computers based on the methods described here. The key issue is errors. The operations are not perfect." This motivated, e.g., Deaton *et al.* [13] to apply evolutionary techniques to search for better DNA encodings, thus reducing the errors during the DNA computation. Sipper, Tomassini, and Beuret [26] studied the effects of random faults on the behavior of some evolved cellular automata, showing that they exhibit graceful degradation in performance, able to tolerate a certain level of faults (see also references therein to other works on faults and damage in cellular models).

– Novel implementation platforms, such as reconfigurable processors (digital and analog), molecular devices, and nanomachines.

4   Concluding remarks

Cellular computing is a vastly parallel, highly local computational paradigm, with simple cells as the basic units of computation. This computational paradigm has been attracting a growing number of researchers in the past few years, producing exciting results that hold prospects for a bright future. Though several open questions yet remain, it is always encouraging to consider the ultimate proof-of-concept: nature.

Acknowledgments

References

1. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 1994.
2. P. W. Anderson. More is different. *Science*, 177(4047):393–396, August 1972.
3. S. C. Benjamin and N. F. Johnson. A possible nanometer-scale computing device based on an adding cellular automaton. *Applied Physics Letters*, 70(17):2321–2323, April 1997.
4. H.-H. Chou and J. A. Reggia. Problem solving during artificial selection of self-replicating loops. *Physica D*, 115(3-4):293–312, May 1998.
5. L. O. Chua and T. Roska. The CNN paradigm. *IEEE Transactions on Circuits and Systems*, 40(3):147–156, March 1993.
6. L. O. Chua and L. Yang. Cellular neural networks: Applications. *IEEE Transactions on Circuits and Systems*, 35(10):1272–1290, October 1988.
7. L. O. Chua and L. Yang. Cellular neural networks: Theory. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1271, October 1988.
8. V. Cimagalli and M. Balsi. Cellular neural networks: A review. In E. Caianiello, editor, *Proceedings of Sixth Italian Workshop on Parallel Architectures and Neural Networks*, Singapore, 1994. World Scientific.
9. M. Conrad. On design principles for a molecular computer. *Communications of the ACM*, 28(5):464–480, May 1985.
10. M. Conrad and E. A. Liberman. Molecular computing as a link between biological and physical theory. *Journal of Theoretical Biology*, 98:239–252, 1982.
11. P. Coveney and R. Highfield. *Frontiers of Complexity: The Search for Order in a Chaotic World*. Faber and Faber, London, 1995.
12. J. P. Crutchfield and M. Mitchell. The evolution of emergent computation. *Proceedings of the National Academy of Sciences USA*, 92(23):10742–10746, 1995.
13. R. Deaton, R. C. Murphy, J. A. Rose, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr. A DNA based implementation of an evolutionary search for good encodings for DNA computation. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 267–271, 1997.
14. W. D. Hillis. *The Connection Machine*. The MIT Press, Cambridge, Massachusetts, 1985.
15. K. Kaneko, I. Tsuda, and T. Ikegami, editors. *Constructive Complexity and Artificial Reality, Proceedings of the Oji International Seminar on Complex Systems-from Complex Dynamical Systems to Sciences of Artificial Reality*, volume 75, Nos. 1-3 of *Physica D*, August 1994.
16. C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10:135–144, 1984.
17. R. J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, April 1995.
18. M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

19. H. R. Pagels. *The Dreams of Reason: The Computer and the Rise of the Sciences of Complexity.* Bantam Books, New York, 1989.
20. J.-Y. Perrier, M. Sipper, and J. Zahnd. Toward a viable, self-reproducing universal computer. *Physica D*, 97:335–352, 1996.
21. J. Rebek, Jr. Synthetic self-replicating molecules. *Scientific American*, 271(1):48–55, July 1994.
22. M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach.* Springer-Verlag, Heidelberg, 1997.
23. M. Sipper. Cellular computing. 1998. (Submitted).
24. M. Sipper and M. Tomassini. Co-evolving parallel random number generators. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 950–959. Springer-Verlag, Heidelberg, 1996.
25. M. Sipper and M. Tomassini. Generating parallel random number generators by cellular programming. *International Journal of Modern Physics C*, 7(2):181–190, 1996.
26. M. Sipper, M. Tomassini, and O. Beuret. Studying probabilistic faults in evolved non-uniform cellular automata. *International Journal of Modern Physics C*, 7(6):923–939, 1996.
27. G. Tempesti. A new self-reproducing cellular automaton capable of construction and computation. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *ECAL'95: Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 555–563, Heidelberg, 1995. Springer-Verlag.
28. J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Illinois, 1966. Edited and completed by A. W. Burks.
29. S. Wolfram. *Cellular Automata and Complexity.* Addison-Wesley, Reading, MA, 1994.

This article was processed using the LaTeX macro package with LLNCS style