LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-289

THE CAM-7 MULTIPROCESSOR: A CELLULAR AUTOMATA MACHINE

TOMMASO TOFFOLI
NORMAN MARGOLUS

DECEMBER 1985

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Massachusetts Institute of Technology
Laboratory for Computer Science

THE CAM-7 MULTIPROCESSOR: A CELLULAR AUTOMATA MACHINE

Tommaso Toffoli
Norman Margolus

December 1985

Contents

1 Introduction **1**

2 Motivation and background **3**

2.1 A new approach to the computer modeling of distributed systems 3

2.2 Cellular automata 4

2.3 Modeling with cellular automata 6

2.3.1 Preview 6

2.3.2 An introductory physical example 7

2.3.3 Microscopic reversibility 8

2.3.4 Image analysis and synthesis 9

2.3.5 Digital circuitry 10

3 A cellular automata facility **12**

3.1 General considerations 12

3.1.1 Data reduction 13

3.1.2 Configuration and control; interactivity 15

3.2 Alternative architectures 16

3.3 Basic architecture of CAM-7 18

3.3.1 The module 18

3.3.2 Larger arrays: edge glueing 19

3.3.3 More states per cell: sheet ganging 20

3.3.4 More dimensions: layer stacking 20

3.4 Advantages of the proposed architecture 20

3.4.1 Optimal use of resources 20

3.4.2 Display and analysis 21

3.4.3 Modularity and expandability 23

4 Technical issues **24**
4.1 System control 24
4.2 Neighborhood options 25
4.3 Initialization and read-out; short-term back-up 27
4.4 Stepping 28
4.5 Display 29
4.6 Error handling 30

5 Plan of work **31**

A Reference material **34**
A.1 The CAM family of cellular automata machines 34
A.2 Case study: Fully-parallel array 36

Chapter 1

Introduction

This paper describes CAM-7, a large, high-performance cellular automata machine that we plan to develop and construct soon. CAM-7 will be suitable for the emulation and study of fine-grained computational processes characterized by spatially adjacent interactions.

The functional architecture of CAM-7 is that of synchronous cellular automata, with discrete space and time and a small state set at each site or *cell*. From an applications viewpoint, this machine may be visualized as a volume of simulated "programmable matter," in which a large variety of experiments can be performed rapidly and conveniently. Examples of this kind of use include a "silicon wind-tunnel" and simulations of physical phenomena such as diffusion.

In addition, CAM-7 will constitute a powerful computer for many information-processing applications dealing with fine-grained structures having a high degree of regularity in at least two dimensions, e.g., a "silicon retina" with real-time performance.

CAM-7 is modularly expandable. The initial realization will handle 134 million sites, which can be arranged as a three-dimensional array of 512x512x512 or a two-dimensional array of 8Kx16K, with two bits per site and an effective throughput of about 8 Giga-EPS ("Events Per Second"—where an *event* is the updating of an individual site.[1])

[1] A cellular-automaton *event* handles a smaller number of bits that a *floating-point operation* (i.e., 10–20 vs. 16–64 bits); on the other hand, the manipulation performed on these bits is of a more general nature.

The design includes facilities for continuous monitoring of overall system parameters and detailed probing of the activity in selected areas of the array. Concurrently with simulation, the machine can also be used as a programmable parallel analyzer, to provide preprocessing of the current system state for subsequent analysis by more conventional means.

Both in terms of its architecture and intended applications, CAM-7 is the next generation in a line of cellular automata machines developed at the MIT Laboratory for Computer Science and used by many investigators. This activity is documented in a number of recent papers and books [3,7,8,11,12,13,14,15,16,17,18].

Cellular automata provide, as an alternative to differential equations, new, powerful methods and techniques for studying a class of large systems[2][6][14][3]. These methods exploit the resources of digital computing in a more direct way than the methods of calculus, and are expected to lead to models that can be formulated more naturally and treated more efficiently.

In addition to providing an alternative modeling tool for the study of systems that are traditionally represented by continuous models, cellular automata offer a natural approach to the modeling of systems that are intrinsically discrete, and whose complexity arises from the presence of a large number of structural elements (parameters and state-variables)[12][10].

The structure of cellular automata directly reflects those ultimate physical contraints to large-scale computation (e.g., topology and connectivity of physical space, finite speed of propagation of signals) that are already felt to some extent by current computers, and which are going to be more and more significant in the design of larger computers. In this context, the proposed machine will offer an advanced testing ground for developing modeling and programming concepts suitable for high-speed, large-scale computation.

Chapter 2

Motivation and background

2.1 A new approach to the computer modeling of distributed systems

To study the properties of a given "target" system we make extensive use of *models*, i.e., substitute systems that reproduce significant aspects of the target but can be handled and examined much more conveniently (in terms of scale, cost, safety, repeatability, accessibility, etc.). In a *simulation* model, the system we construct is one that can be made to "run," i.e., to evolve in time mimicking the evolution of the target. In this context, computation is the process that continually updates every element—or *state-variable*—of the system. As we increase the spatial resolution of the model, as well as its resolution in time and in the representation of the state-variables, the model demands larger and larger amounts of computation.

The architecture of conventional computers imposes a strong bias on the nature of the models that one can efficiently handle. These computers greatly emphasize resolution in the domain of the individual state-variables, i.e., they are optimized to handle fixed- or floating-point variables of substantial range. Processors of the kind required to update these variables to their full resolution (i.e., arithmetic/logic or floating-point processors) are intrinsically complex and expensive. As a consequence, the computer is typically equipped with a single processor, or at most a few as in some high-performance machines, which must be time-shared between all of the variables that make up the model. Since this complex processor is relatively

slow and entails substantial overhead to be time-shared, in a large model state-variables are updated at a low rate.

What we propose here is a new kind of modeling facility, based on an approach that is complementary to that of conventional computers. In this approach, space and time in the target system are sampled at a higher resolution; on the other hand, the state of each element of the model is a *symbol* chosen from a small state-alphabet—individual state-variables make no attempt to emulate the precision of a real number. A processor that can handle such variables is much simpler and faster, and its cost is therefore comparable to that of storage for the variable itself. Thus, one can seriously consider implementations where each variable (or small group of variables) is served by its own processor.[1] In this case, processing power grows in proportion to the number of variables, and large systems can be simulated at reasonable speed.

2.2 Cellular automata

Cellular automata are discrete dynamical systems whose behavior is completely specified in terms of a local relation, much as is the case for a large class of continuous dynamical systems defined by partial differential equations. In this sense, *cellular automata are the computer scientist's counterpart to the physicist's concept of "field."*

A cellular automaton can be thought of as a stylized universe. It consists of a uniform grid, with each site or cell containing a few bits of data; time advances in discrete steps; and the laws of the universe are represented by a single rule—e.g., a small look-up table—through which at each time-step each cell computes its new state from that of its nearest neighbors. As a consequence, the system's laws are *local* and *uniform*. Given suitable rules, such a simple mechanism is sufficient to support a whole hierarchy of structures, phenomena, and properties[12][20][10]. Cellular automata provide eminently usable models for many investigations in natural science,

[1] While the processor for a given model might consist of a fixed arrangement of a few logic gates, in practice a general-purpose machine useful for exploring this approach requires a somewhat larger, more flexible processor. In CAM-7, the problem is solved by using a programmable look-up table; look-up is fast, and the table can be shared by a group of variables.

combinatorial mathematics, and computer science; in particular, they represent a natural way of studying the evolution of large physical systems. They also constitute a general paradigm for parallel computation, much as Turing machines do for serial computation.

The generality and flexibility of the cellular-automaton approach are achieved at a cost. Instead of having relatively few lumped variables that interact in an arbitrarily assigned way, a cellular automaton uses many variables (i.e., one per cell) that interact only locally and uniformly. In order to synthesize structures of significant complexity it is necessary to use a large number of cells, and in order for these structures to interact with one another and evolve to a significant extent it is necessary to let the automaton run for a large number of time-steps. For elementary applications, a satisfactory experimental run may require the computation of billions of events (an *event* is the updating of a single cell); for more substantial applications, a thousand or a million times this value may be desirable (i.e., 10^{12}–10^{15} events): the limits are really set by technology rather than by the applications, which by virtue of their matter-like nature can occupy an arbitrarily large number of cells.

In such applications, von-Neumann-architectures are of little use: when the simulation of a cellular automaton is carried out on a sequential computer, an event may require some thirty machine operations each involving a few machine cycles—i.e., ≈ 10 μsec on a fast machine. To compute $\approx 10^{13}$ events under such an approach one would need several years.

On the other hand, a cellular automaton is the ideal mathematical structure for a machine having a high degree of parallelism and local and uniform interconnections.[2] In Chapters 3 and 4 we discuss in more concrete terms the design of this machine.

[2]The term "non-von Neumann architecture" is often used to stress the difference between a parallel computer of this kind and more conventional sequential computers. However, it should be noted that the theory of cellular automata originated with von Neumann himself.

2.3 Modeling with cellular automata

2.3.1 Preview

Cellular automata are most naturally used, and cellular automata machines most efficiently exploited, in modeling systems whose geometric and causal relations are close to those of ordinary space-time. These include:

1. Problems of a direct physical nature: statistical mechanics, fluid dynamics, microscopic mechanics, etc.

2. Problems governed by semi-empirical laws which are in principle reducible to the laws of physics, such as those encountered in chemistry, biology, and geology: reaction dynamics, materials science, and land erosion.

3. Problems analogous to the above, but entailing a higher level of aggregation: population dynamics, epidemiology, and communications networks.

4. Issues of self-organization, stability, resistance to noise, self-reproduction, and evolution; chaotic behavior arising from simple laws and simple initial conditions; and, in general, nonlinear dynamics in complex systems.

5. Problems satisfying realistic space-time constraints, but whose laws are arbitrarily assigned: simulation of large digital circuits; distributed computation and control; inventory and shipping, flows in networks; certain questions in graph theory, dynamic programming, operations research; certain board games; and, in general, simulation of universes provided with an *ad hoc* set of laws.

6. Miscellaneous problems having a strong geometric component: image processing, analysis, and generation; parallel encryption and decryption; certain problems of pattern recognition.

There are two additional contexts in which cellular automata machines of the kind described here may prove useful:

1. Miscellaneous combinatorial problems that can be decomposed into a large number of small independent subproblems. Here, the individual subproblems are parceled out to different portions of the cellular automata array and processed in parallel. This approach might be useful, for instance, in factoring.

2. As we shall explain in Chapter 4, the specific implementation of the cellular automata architecture that we propose for CAM-7 provides in a natural way additional interconnection and processing features that go beyond those required for strictly local and uniform processing. In particular, the 512 processors can each run a *different* program; and schemes are available for achieving communication that is faster than linear propagation (e.g., instant communication between selected sites, or log-distance communication between arbitrary sites). With these additional features, many problems that transcend the local and uniform paradigm can be treated by CAM-7 with little, if any, degradation of performance.

In the remainder of this chapter we shall highlight some aspects of modeling with cellular automata.

2.3.2 An introductory physical example

Let us consider the Navier-Stokes equation—the "master equation" of hydrodynamics. The conventional way to study the evolution of a system governed by this equation is to discretize space and time, rewrite the equation in terms of a finite-difference scheme acting on *real variables* attached to the points of the space-time grid, code this scheme as a FORTRAN program, and run the program on a general-purpose computer. On the other hand, by using CAM, in which a large number of *binary variables* are interconnected locally and uniformly (i.e., in a way that directly reflects the structure of the physical space-time in which the hydrodynamical system resides), all one has to do is give a small look-up table that specifies the behavior of an individual binary variable as a function of the state of its neighbors. This table, consisting of a dozen binary entries, bears no resemblance to the Navier-Stokes equation; yet on a macroscopic scale the resulting model

behaves *isomorphically* to the original physical system[3][6,14,9].

What has happened is that the "matter" of CAM has been programmed to be a *fluid*. The fluid's individual particles carry on their normal routine, which is simple and certainly not *governed* by the Navier-Stokes equation; rather, the latter is just a macroscopic consequence of the former. With the CAM approach, we directly capture the particles' activity, and we are able to model it so efficiently that we can effectively derive macroscopic features of the fluid.

Using this approach on smaller cellular automata machines such as CAM-5, we have modeled a jet of gas flowing past an obstacle. With CAM-7, experiments of a more realistic size—of the nature of a wind tunnel—will be possible.

The above approach can readily be extended to problems of material science, thermophysics, and acoustics, by making direct use of known microscopic interactions; and to problems of land erosion, ecology, and epidemiology, by introducing interactions of a more macroscopic nature. Finally, one can set up *phenomenological* investigations of problems entailing empirical interactions between a large number of entities, such as are studied in traffic, economics, voting theory, political science.

2.3.3 Microscopic reversibility

One isssue that is of fundamental importance in physics is that of *microscopic reversibility*—or *invertibility*, to use the standard mathematical term.

As far as we know, reversibility is a universal characteristic of physical laws. In particular, it is a necessary prerequisite for the second law of thermodynamics to hold,[4] and is a sufficient condition for the existence of conserved quantities.[5]

---

[3]This model is also capable of dealing with *compressible* fluids; however, its behavior at Mach numbers close to unity has not yet been analyzed in detail.

[4]For systems having a *discrete* state set, such as cellular automata, reversibility directly yields Liouville's principle ("incompressibility of the phase space"), since each distinct initial state leads to a distinct final state.

[5]In physics, a reversible system having n degrees of freedom possesses $2n-1$ conserved quantities, some of which (e.g., energy, momentum, etc.) are of special significance because of their connection with fundamental symmetries of the physical laws. The arguments that

Outside of scientific computing, reversibility considerations are relevant when dealing with the integrity of data in concurrent transactions, and in this context they provide tools for insuring proper synchronization and interlocking.

The above considerations suggest that *invertible* cellular automata may play an important role in the study of reversible distributed dynamics. In fact, it is possible to construct cellular automata that are *exactly* invertible (in this respect, they suffer none of the approximations that are so common in conventional numerical simulations). Thus, one can arrive at models that, thought drastically stylized in other respects, make no compromises in the representation of certain fundamental aspects of a physical process or of a data transaction.

For this and other reasons, invertible cellular automata show promise of becoming useful modeling tools in many areas of investigation.

The theory of invertible cellular automata[16] has many open problems; in particular, no general decision procedure is known for determining whether a given rule has an inverse (and this question may well be undecidable). Until recently, no examples at all were known of invertible cellular automata having better than trivial computational capabilities.

We have developed theoretical methods and practical techniques for constructing invertible cellular automata having a variety of desirable computational capabilities. Some of these techniques (second-order cellular automata, Margolus neighborhood, guarded context[15]) are suitable for implementation on cellular automata machines, and are directly incorporated in the design of CAM-7.

2.3.4 Image analysis and synthesis

The task of image analysis is substantially one of extracting specific features or correlations out of a picture. The representation of high-level correlations usually requires dealing with an abstract space having arbitrary interconnections, and is better handled by AI techniques; on the other hand, the

lead to these conservation laws can be generalized to cellular automata: the key idea is that a given state "encodes" all of the information necessary to identify the particular dynamical trajectory it lies on, and, if the system is reversible, none of this information is lost in the course of its evolution.

processing of low-level correlations (e.g., classification of texture, outline, light levels, shape and size of small objects) entails operations of an essentially local and uniform nature that can be carried out in parallel by cellular automata techniques. Of the several stages of computation involved in image analysis, front-end processing involves the largest amount of raw data and provides the largest factor of data compression—and thus is the most obvious candidate for parallel computing techniques.

Some simple image-analysis techniques based on cellular automata are already in use and have been incorporated in the design of special-purpose processors aimed, for instance, at robotic "vision" (cf. [19]). More sophisticated analysis techniques require correlating data on areas wider than a few pixels and realizing convolutions having a substantial amount of local memory. CAM-7 will provide ample resources suitable for this purpose. In particular, each of CAM-7's two-dimensional bit-planes can be assigned a different transition function; in this way, complex sequences of image-processing steps can be pipelined, leading to real-time processing of a steady stream of visual data. Moreover, the structure and the size of CAM-7 will make possible the exploration of processing techniques for *three-dimensional* images of realistic size.

Besides image analysis, CAM-7 will permit the exploration of new methods of image synthesis. A cellular automata machine can be though of as a universe synthesizer, and thus can naturally be used in a variety of contexts as a synthesizer of images or image sequences. Since the spatial and temporal textures of these images reflect definite microscopic laws, a high degree of internal consistency can be achieved by using fewer parameters than with conventional computer graphics methods (cf. Knuth's METAFONT approach to typographical font generation).

2.3.5 Digital circuitry

A cellular automaton can be thought of as an array of uniformly interconnected logic elements. It is possible to choose a cellular-automaton rule such that each cell can act—depending on its state—as a gate, a section of wire, or a memory element. Thus, the generic array can be turned into an arbitrary digital circuit by downloading an initial configuration representing the schematics of the circuit itself. "Soft circuitry" of this kind naturally

extends to parallel computation the well-established role that "software" plays in sequential computation.

To make the design of such gate arrays possible, one must simulate their behavior at a reasonable speed. A simulation on an ordinary sequential computer would run perhaps a trillion times slower than the target array (cf. Appendix A.2). A general-purpose cellular automata machine of the kind proposed here would run only a few million times slower than the target, and at the same time would provide the flexibility and the interactivity required for experimentation and design iteration. (Unclocked, asynchronous cellular automata might run orders of magnitude faster—these also can be studied on CAM-7 using random-number generators to control whether a given site gets updated during a given time-step.)

More generally, CAM-7 will represent an ideal development system for designing target system based on cellular automata. Once a certain cellular automaton rule has been found useful in a particular application of wide practical interest (viz., hydrodynamics, image processing), it then becomes practical to build a VLSI chip that implements that rule in a fully-parallel way (cf. Appendix A.2), running thousands of times faster than CAM-7.

Chapter 3

A cellular automata facility

3.1 General considerations

In designing a large cellular automata machine, it is important for us to anticipate the range of applications through which it will be put by the end user. We must therefore visualize it as the nucleus of a dedicated computational facility. Here, it may help to think of other experimental facilities built around a substantial piece of apparatus, such as a large telescope, a particle accelerator, or a wind tunnel. In such facilities, a number of experiments may be in progress at any given time, some directly using the core machine and monitoring it by real-time analyzers, event detectors, etc.; others processing off-line data from previous runs, setting up filters and detectors or preparing control programs for the next run, and so forth.

The amount of resources invested in the facility demands careful planning of design and operation; on the other hand, it is in the nature of applications and experiments to run their course and be replaced by new ones, often suggested by the current results and thus hard to anticipate. Thus, our principal responsibility at the design stage is to insure versatility and robustness over a wide range of application types, rather than optimize for a known set of applications.

In the facility we propose, the major resource will be represented by CAM-7—the cellular automata machine. As we have already remarked, one can visualize CAM-7 as a volume of simulated programmable matter.

By assigning the transition function, the user specifies what kind of laws will be in force within this volume; by assigning its initial configuration, one can place in this volume structures made out of different materials and surrounded by different media. By operating the machine, the volume of matter is made to run through time; structures and media will interact and evolve, and each experiment will realize the dynamic behavior of a particular world.

3.1.1 Data reduction

This volume of matter can be programmed to model a large variety of phenomena over a wide space- and time-scale (cf. 2.3.1). The machine's basic architecture (cf. 3.3) guarantees that, independently of what is being modeled, the state variables will be updated at great speed and in the specified way. However, the user's requirements for access, interpretation, and further treatment of this information are strongly dependent on the nature of the model and on the particular experiment being conducted. Accordingly, one of the major design tasks will be to complement the machine's basic architecture with a number of built-in, analysis-oriented features that will help make this information *useable*. How much happens in an experiment is of value only in terms of how much the investigator can see, organize, and digest.

We must stress here that while in most computations one is interested only in the final result, in a simulation the entire course of intermediate results is of potential interest to the investigator. Just as in many large-scale physical experiments, one is confronted with a serious problem of data reduction. The bigger and faster the machine, the more important it is to equip it with appropriate data-reduction resources.

The major channels for monitoring and analyzing the machine's throughput will include at least the following.

1. Direct, real-time display on color monitors of any portions of the simulation volume (no frame "grabbers" or frame buffers are needed for this purpose, since data sequencing and formatting within each module are directly compatible with raster-scan monitors).

   This will provide a number of windows on the ongoing process, and will allow one to bring in the sophisticated real-time processing ca-

pabilities of the human eye and brain. We have found this feature also invaluable as an aid for the timely detection of software bugs and hardware malfunctions.

2. The machine itself can be turned into a programmable parallel processor for analyzing the configurations successively reached in the course of a simulation. This analysis channel is particularly important, since its capabilities grow in direct proportion to the size of the machine.

3. In parallel with the updating of the bits and with no speed penalty CAM is capable of computing—via look-up tables quite similar to those used for the updating function—another local function of comparable complexity. The results of this function represent preprocessed data that can be fed to built-in global counters (this is useful for event detection and counting, as needed for example in correlation experiments) or made available to external analyzers. Thus, a substantial amount of real-time data analysis and/or display processing can be performed "piggy-back" on the simulation.

4. CAM-7's architecture provides *free and continuous access to the full data bandwidth (≈16 Gbits/sec)* of the simulation. This "flywheel" I/O bus is a unique feature of the CAM approach. At each step, every bit of the model is made available to be examined and possibly changed by external devices, thus making it possible to couple the system to specialized forcing functions and/or analyzers.

5. A small volume within the overall simulation volume can be equipped with a probe, i.e., a small cellular automata machine constructed out of the same kind of components but dedicated to extensive on-line pattern recognition and/or pattern generation—perhaps with a high-bandwith connection to a conventional general-purpose computer.

6. A general-purpose computer, the *control host*, will be intimately connected with the cellular automata machine, and will be able to sense and set special hardware registers in each of the machine's modules. This computer will provide the operating system for the overall monitoring and control of the simulation and the supervison of bulk input/output transfers.

7. In addition to the flywheel bus, a high-speed bus of conventional structure will be available for rapid data transfers (bulk I/O) between CAM-7 and storage or analysis devices.

Having designed in the machine adequate flexibility and power for the first level of data access, reduction, and analysis, we need not discuss in detail at the design stage further levels of analysis, which will take place externally to CAM-7. They can be handled by conventional general-purpose computers or, when necessary, by special-purpose configurations provided by the individual experimenters. In this context, the LCS Multiprocessing Emulation Facility (MEF) may offer sufficient power and flexibility for the analysis of representative CAM-7 experiments, for the evaluation of analysis strategies, for emulating a variety of analysis configurations, and for evaluating the peformance of the high-speed bus under different load patterns.

3.1.2 Configuration and control; interactivity

Two important features of our approach to systems modeling are:

1. The high level of *interactivity* that can be achieved (and which is enhanced by the high-quality real-time display).

2. The *flexibility* in composing, concurrently or sequentially, different kinds of dynamical and analysis steps and different kinds of input/output interventions (e.g., assignment of forcing terms in the boundary conditions, and detection of events for the purpose of conditionally altering the course of a simulation).

In order to insure efficient real-time support of these features, appropriate provisions must be made in the hardware and in the interface with the host, mostly in the form of control channels and of software-reconfigurable hardware ("configuration" options). Since the number of potential options in this context is enormous, here too it is necessary for us to anticipate at the design stage a range of realistic simulation circumstances, and try to reach a reasonable compromise between generality and flexibility on one hand, and hardware complexity on the other. A more detailed discussion of proposed configuration and control features is given in Chapter 4.

3.2 Alternative architectures

Most of the architectural aspects of CAM-7 have already been realized and verified in earlier machines of the CAM family (cf. A.1). Keeping this in mind, we shall briefly review some alternative approaches. This should help make it clear why the proposed solution is appropriate, and why the other alternatives are not viable for the class of applications that interest us.

The first alternative we will consider is, of course, the use of an ordinary general-pupose computer for the simulation of cellular automata. For greater processing power, one could connect several such computers in parallel, each handling a part of the simulation space. However, to achieve a performance comparable to the proposed design one would need from *thousands* to *millions* of such machines. Even if we were to disregard cost, the total bulk and power requirements would be prohibitive; interfacing and interconnection would demand enormous resources; and overall reliability would be nearly impossible to guarantee.

There are *array* or *vector* processors on the market which execute the same machine instruction in parallel on a number of independent register sets; they can be programmed to run cellular automata one or two orders of magnitude faster than an ordinary computer. However, in these machines the emphasis is on general-purpose numerical computation; their control and arithmetic/logic machinery is more complex and less flexible than is desirable for computing cellular automata events. This approach entails a lot of unnecessary overhead, and is neither practical nor economical if such resources have to be multiplied by a thousand or a hundred thousand to achieve the desired performance.

Other machines, which emphasize graphic display processing, have dedicated resources for performing logical operations in parallel on bit-plane representations of pictures. Taking advantage of this feature, certain simple cellular automata rules can be made to run quite fast; however, this technique is of little use in the more general case, which may require thousands of such parallel bit operations for a single update of the whole array.

Two experimental architectures, namely those of the Yorktown Simulation Engine (YSE) and the Connection Machine (CM), specifically address the modeling of large systems in which the individual state-variables represent discrete logical quantities. Both architectures stress generality of interconnection and generality of functions performed at each site. These

features are invaluable for the intended applications (namely, high-speed simulation of logic circuitry, for YSE, and handling of data structures typical of AI work, for CM). However, the same features tie up a very large amount of resources, slow operations down, prevent the possibility of modular expansion, and overall make both machines poorly suited to the task of modeling large systems with the cellular automata approach. In particular, YSE is much too small (roughly, by a factor of 1000) and CM much too slow (by a similar factor) for our purposes.

Finally, certain research laboratories have developed special-purpose machines for computing statistical properties of magnetic crystals (Ising spin models). Optimal design criteria for this application come close to those for a cellular automaton; however, owing to the limited scope of the application, these machines are more specialized than the one we propose. They must be regarded substantially as custom tools, of little use for general-purpose cellular automata applications.

From a conceptual viewpoint, the most natural architecture for a cellular automata machine is a fully-parallel one. That is, one can

1. Design a circuit that implements a single cell,

2. Fit as many copies as possible of this circuit on a large VLSI module, and finally

3. Produce and interconnect as many of these modules as is technically and economically feasible.

Since cells are small and all inteconnections are local, one can easily visualize an array of millions of such cells running in parallel at only a few nanoseconds per step.

Attractive as it may appear at first, such a fully-parallel architecture raises serious difficulties, especially for a large, general-purpose machine. In particular, one encounters problems of *flexibility* and *computing power* of the individual cell, *interconnection density*, and *data accessibility* (see case study in Appendix A.2). This architecture may be suitable for specialized applications, or for use with new technologies (e.g., molecular computers).

3.3 Basic architecture of CAM-7

The architecture of CAM-7 maintains the basic conceptual approach of the fully-parallel machine discussed above, but with certain variants that lead to a more practical and economical realization and a better utilization of current technological resources. With reference to the fully-parallel approach, the CAM-7 architecture is still based on *modules*, an arbitrary number of which can be connected in parallel; and each module still spans a large number of sites. However, the individual module is a *pipelined* rather than a *parallel* processor. As we shall see, several features of this approach reinforce one another synergistically, making it more attractive than the competing alternatives.

3.3.1 The module

For the sake of the present discussion we shall restrict our attention to two-dimensional cellular automata containing one bit of data at each site. More dimensions and larger state-sets are discussed in the following sections.

The whole array is partitioned into rectangular portions of identical size called *sectors*, and a separate hardware module is assigned to each sector. Each module consists of three main sections—*state-variable storage*, *data routing*, and *transition function*.

The storage section contains the state variables of the corresponding sector. In order to perform one updating step on this area, the current values of the state variables are read once, sequentially, and injected into the routing section. The corresponding new values, determined by table look-up, are returned by this section in the same sequential order and written back onto the storage section.

From the above sequential stream of data, the routing section extracts with the appropriate timing the nine values corresponding at each moment to the nine neighbor positions of a site: the site itself, or *Center*; its four nearest neighbors, *North*, *South*, *East*, and *West*; and its four next-nearest neighbors *N.East*, *N.West*, *S.East*, and *S.West*. This section also provides appropriate buffering to make the updating of sites appear *synchronous* even though realized in a sequential manner, and to achieve correct vertical and horizontal wrap-around.

Any desired subset of the above nine signals, possibly augmented by

signals coming from other modules (cf. 3.3.3, 3.3.4), are submitted in parallel as arguments to the transition-function section, which uses a look-up table to compute the corresponding new value for the center cell. After a brief journey through the routing section, the new value is handed to the storage section, where it replaces the current value of that cell. Note that all ancillary tasks such as argument gathering are performed by the routing section; thus the look-up table, which is the most critical resource in the simulation, is exploited to its full bandwidth. Moreover, since each table is shared by a large number of cells (256K in the proposed design), it becomes practical to employ a very large look-up table (16K–64K entries), thus compressing a substantial amount of computation into a single step.

3.3.2 Larger arrays: edge glueing

An arbitrarily large two-dimensional array can be obtained by *glueing* sectors edge-to-edge, i.e., by exchanging between the pipelines of two adjacent sectors data about those sites that are contained in one module but are neighbors of sites in the other module. The proposed size for a module is 512x512 sites. In the fully-parallel architecture, this would entail a module with thousands of external terminations (cf. A.2); in the pipelined architecture, instead, exchange of information at the edges is serial, and four bidirectional lines, corresponding to the four adjacent sectors, are sufficient.[1]

By glueing sectors in this way, one obtains an arbitrarily large *sheet*; typically, this sheet will be wrapped-around, i.e., the top edge will be joined to the bottom edge and the left to the right; thus, the overall topology of a sheet will be that of the surface of a torus. The same glueing technique is used both for array-expansion purposes and for boundary elimination by wraparound.

Observe that the glueing of modules is done once at the routing stage. In this way, both from a logical and a physical viewpoint the transition-function section is *completely decoupled* from a number of implementation

[1] This sector-joining technique relies on the fact that the cell memory of the individual modules is logically wrapped around—a cell at the physical edge of the sector sees cells on both that edge and the far edge as neighbors. Since the scanning pattern for cell updates is the same for all modules, all modules have the appropriate edge neighbors available simultaneously to be exchanged. By exchanging pipelines rather than neighbors, we get the same effect with one connection to each immediately adjacent sector *independently* of the size of the neighborhood.

details, namely, (a) the fact that a sheet consists of modules glued together, (b) that storage and routing are done on a two-dimensional basis, and independently for each bit plane, and (c) that operations are pipelined.

3.3.3 More states per cell: sheet ganging

Once sheets of the desired size have been assembled, further hardware configuring of the cellular automata machine is done by selecting suitable signals as arguments to the transition-function. In particular, in order to have a larger state-set for the automaton's cell it is sufficient to *gang* a set of sheets, i.e., connect as inputs to the look-up table of each sheet a selection of neighbor outputs from the other sheets of the group. Such a ganged set will then constitute a *layer* of the cellular automaton, containing a complete cell at each site.

In practice, the CAM-7 modules will already handle four sheets each, so that further ganging will be required only in special situations. By combining several sheets in the same module, and thus making certain data paths *internal* to a module, it is possible to feed the look-up tables with particularly useful, wide-scoped neighbor configurations that would otherwise impose too large an interconnection burden.

3.3.4 More dimensions: layer stacking

Finally, layers can be *stacked* on top of one another, by connecting as inputs to the transition function of each layer a selection of neighbor outputs from the layers immediately above and below. This is possible because all modules will be updating corresponding cells at the same time. In this way we can configure CAM-7 into a three-dimensional cellular automaton. This construction can be further iterated in order to obtain cellular automata in four or more dimensions.

3.4 Advantages of the proposed architecture

3.4.1 Optimal use of resources

No matter how much ingenuity is put into the design of a cellular automata machine (or, for that matter, of any computer), an upper bound to its

performance is set by two essential resources, namely, the *storage* available for representing the state variables, and the amount of *processing power* available for computing the transition function. The architecture of CAM-7 allows us to make use of the least-expensive form of these two resources available on the market, and to exploit them to their fullest extent; it thus comes close to the theoretical optimum in terms of price/performance.

Specifically, the three main sections of a module (cf. 3.3.1) are quite naturally realized as three distinct integrated-circuit chips, each one performing a function that is almost ideally catered to by the current state of silicon technology.

Namely, the *storage* section can be realized directly as a large dynamic-RAM chip—and thus take advantage of the investments made in developing this standard commodity. In the proposed design (see Chapter 4), the size and access time of the storage section are matched to the popular 256K DRAMs.

Similarly, the *transition function* section, which in our design is a look-up table, can be realized directly as a medium-size, fast static-RAM chip. This chip has also become a standard commodity, owing to heavy demand for it in caches, video refresh buffers, and microprogram control stores.

Finally, the glue that ties all these circuits together, that is, the *routing* section, can be realized by a semi-custom VLSI chip. In fact, the speed, number, and kind of functions performed by this section can be accomodated by routine fabrication techniques, and the design of the chip can take advantage of standard macro-cell libraries.

3.4.2 Display and analysis

Each module of CAM-7 generates new data at a rate of ≈20 Mbits/sec. If one had to do any substantial reformatting of this information for display purposes, one would need resources of the same order of magnitude as those used for producing it.

In the pipelined architecture, scanning of the array is sequential; with an appropriate choice of scanning parameters this information can be made to appear in the correct framing format for display on a raster-scan device. In the CAM-7 module, the number of array rows and columns spanned by the module, the scanning sequence, and the timing are such that a tap on

the pipeline can directly feed a conventional CRT monitor running at 60 Hz with 525 non-interlaced lines. Of course, the outputs from a set of ganged modules (cf. 3.3.3), which collectively represent the value of a multi-bit state variable, can be combined into an RGB signal and displayed on a single *color* monitor.[2]

The advantages of this set-up are not limited to raw display. On the module, all the neighbor information that is potentially available to the transition function is conveniently accessible, and can be fed to an additional look-up table. In this way, one can compute and send to the display an arbitrary *output* function, instead of just the value of the current center cell. This allows one to do on-the-fly a substantial amount of graphic preprocessing (this approach remainds us of the "staining" techniques used in microscopy for enhancing selected features of the tissue under examination).

Further, the stream of values supplied by such an output function can be sorted into a histogram, accumulated and compared with set threshold values, and in general used for real-time processing and control of the system's dynamics. In particular, one can locate and count occurrences of any specified local pattern.

Finally, since at each updating step all the data on each module are streamed through the pipeline, a single bidirectional tap on this pipeline is sufficient to provide any external device with read and write access to the totality of the data. The collection of these taps, one per module, constitutes an extremely high-speed bus (with an overall word width of 1024 bits and a synchronous word rate of 40 nsec) through which the entire state of the simulated system is continually made accessible to the experimenter while the simulation is in progress—and without slowing it down. This "flywheel bus" (cf. Section 3.1.1) is unique to the CAM architecture.

In conclusion, a pipeline fed according to a well-chosen sequencing format and provided with a few well-placed taps constitutes a general-purpose bus on which one can hook up not only the transition function, but also a great variety of display, analysis, and control functions—without any overhead on the simulation process. As in a physical experiment, any portion of

[2]When display is not required, CAM's clock can be decoupled from the video rate, to allow—for example—more frequent updating of a smaller array, say at a few thousand frames per second.

the system is potentially accessible to on-line stimulation and measurement.

3.4.3 Modularity and expandability

The flexibility of the proposed module, both in terms of internal architecture and external interface, makes possible its use it in a variety of machine configurations and sizes. Moreover, out of the same modules one can construct not only general-purpose simulators of distributed dynamical systems, but also distributed analyzers that are matched to the simulators in data-handling format and processing power. Thus, the *observation power* available to the experimenter can be made to grow hand-in-hand with the *simulation power* of the experiment.

Initially, a typical CAM-7 realization will consist of 1024 modules, configurable, for instance, as a $512 \times 512 \times 512$ cube with two bits per site. However, unlike other current schemes for parallel computation, the CAM-7 architecture is truly scale-independent, and a much larger cellular automata machine can be built simply by connecting together an appropriate number of modules. The limits are set by economic constraints rather than by electrical problems or issues of logic design. Since there are no "addresses" in a traditional sense, the data space is not limited by the size of an address word. The only timing signal that is distributed to each block of modules is the clock; and since signals are reclocked within each block, the system can cope with a timing slack between blocks comparable to the width of the clock pulse ($\approx$40 nsec).

Finally, we note that the single modules are by themselves useful experimental tools. A miniature copy of the proposed cellular automata machine, consisting of a few modules, will be used as a prototype. Such reduced-size machines could be made available for evaluation, training, experiment preparation, and software testing. In this way, full-scale experiments would be ported to a major facility in a ready-to-run state.

Chapter 4

Technical issues

The design of CAM-7 draws much on the experience gained from designing and operating smaller machines—such as CAM-5 and CAM-6 (Appendix A.1). In particular, the proposed architectural solution concerning the simulation process (see Section 3.3) will require only a refinement of techniques that are known to us and are well tested. On the other hand, the size of CAM-7 poses a number of original problems (mostly concerning what in a smaller machine would be termed ancillary functions, such as configuration, control, monitoring, data-analysis, error detection and correction, and testing) whose optimal solution will require substantial additional work, in collaboration both with experts in various aspects of electronic design and with potential users of the cellular automata facility.

In this chapter we shall concentrate on those technical aspects which we believe will be of particular relevance to the operation of the machine as a flexible experimental tool, and on which we intend to do further study before finalizing design details.

4.1 System control

CAM-7 will be controlled by a dedicated host computer (*control host*) of adequate performance; this computer will also coordinate the activity of optional *analysis hosts*. In writing software for the control host, attention will be given to the interactive nature of modeling work.

Data transfers between the control host and the cellular automata ma-

chine will be through a high-speed control bus serving all modules. Certain data, such as configuration parameters and look-up tables, will be broadcast at the same time to all modules, or to large groups of them; other data will be written to or read from individual modules, sequentially or randomly.

The following is a representative list of control functions: selection of neighborhood options (cf. 4.2), sector glueing (3.3.2), phase specification and channel selection[13], selection of display function and display color map (3.4.2), stepping(4.4), downloading of transition and display functions, selection of internal pattern-initialization modes, setting and reading of event counters, configuration of the random-number generators, and error handling.

4.2 Neighborhood options

An important and difficult decision, since it will perforce entail compromise between the exigencies of different users, concerns the choice of signals or groups of signals to be made available to a module's transition-function look-up table (cf. Section 3.3.1), so as to configure the machine for a given cellular-automaton neighborhood format. Many questions concerning this issue have already been answered in the design of CAM-6; additional questions presented by CAM-7 arise from its large scale and its emphasis on three-dimensional simulations.

Let's consider a few typical configurations. In Conway's well-known game of "life"[5] one has a two-dimensional cellular automaton with one bit per cell and nine neighbors; thus, the look-up table requires 9 binary address lines, and contains $2^9 = 512$ 1-bit entries. In Codd's universal computer/constructor[1] there are 3 bits per cell and 5 neighbors, for a total of 15 address lines and $2^{15}(= 32,768)$ 3-bit entries. A three-dimensional, 1-bit cellular automaton in which each cell "sees" its 6 nearest neighbors in addition to itself requires 7 address lines for the look-up table (2^7 entries); if the cell contained 2 bits, one would need 14 lines.

Since the size of the table grows exponentially with the number of address lines, it is clear that a look-up table of practical size (say, 16 input lines, corresponding to 65,536 entries) cannot be permanently connected to the several dozen signals that at one time or another might be used as ar-

guments to the transition function; in order to configure the machine for a particular cellular-automaton format one will have to connect the relatively few inputs of the look-up table to a particular set of relevant neighbors. By and large, the selection of such a set will be done by software-controlled hardware multiplexing. The real question is which and how many sets should be preselected at the design stage as multiplexing options.

The above problem is made more complex by the following factors:

1. In addition to neighbors, there will be available many signals (or *pseudo-neighbors*), generated within the module or externally, that can have an important use as inputs to the transition function. These include space- or time-dependent parameters, such as boundary conditions, space or time "grids," or the output from a random number generator.

2. There are many cases in which the number of relevant signals is too large for direct look-up, but can be brought within a practical range by a small amount of preprocessing. For instance, several variations of the Margolus neigborhood[8][15], which has important application in the synthesis of microscopically reversible systems, compress 12–13 signals down to 4 signals (in two dimensions), or 30–32 signals down to 8 (in three dimensions). Techniques of this kind greatly expand the range of systems that can be realized with a given size of look-up table; but it is clear that one should arrive quite early in the design stage at a decision as to which combinations of preprocessing tools will be provided by the hardware.

3. Neighbors and pseudo-neighbors are used not only by the transition function but also by the output-function look-up tables (cf. 3.4.2). Here too one must resort to preselected sets of neighbors. Though generally one would use a set of neighbors quite similar to that supplied to the transition function, there are many occasions where variants or additions to this set would greatly contribute to the processing power of the output function. As above, to avoid a combinatorial explosion, reasonable compromises will have to be made.

4.3 Initialization and read-out; short-term back-up

In operating a simulator under control of a host computer, two functions will have to be performed habitually, i.e.:

1. Initializing the system's state at the beginning of an experiment.

2. Reading out the system's state at the end of the experiment, and possibly a number of times during the course of the experiment itself, for a permanent record or for off-line analysis.

Of course, read-out and subsequent re-initialization are also required when a simulation task is suspended to make room for a higher-priority task.

In addition, a simulation task may be interrupted and resumed several times on a short-term basis. Typical situations for this are the following.

1. The machine is being briefly switched from simulation mode to analysis mode (cf. Section 3.1.1)..

2. Some real-time analysis is done in a pipelined fashion, and thus its stream of results lags behind the simulation. If an interesting event is detected that demands closer scrutiny, one would like to back up the simulation to the moment the event occurred. With reversible cellular-automaton rules (cf. Section 2.3.3) it is possible to make the simulation retrace its steps. In general, however, one would have to make frequent *breakpoints*, where the current state is saved and held until the next breakpoint, so that the missed event can be reconstructed by re-running the simulation for a few steps starting from the last breakpoint.

All of these cases entail saving the current state in some form of back-up memory and eventually restoring the system to a previously-saved state.

Let us consider first the issue of data transfer. In a typical machine configuration (here, we are thinking of an initial realization, since the architecture is modularly expandable) the state-variables of CAM-7 will collectively amount to $\approx$ 32 Mbytes. On one hand, this amount of information is comparable to the primary-storage capacity (physical RAM) of a typical

analysis host, and to a significant fraction of its secondary-storage capacity (hard-disk); on the other hand, merely transfering this information in or out of the host at, say, 100 nsec/byte will take ≈ 3 seconds, that is, a time interval corresponding to 200 simulation steps. It is clear that, to avoid tying up valuable resources, one should strive to keep the number of such bulk serial transfers low.

Since short-term back-up may be frequently required, perhaps the most reasonable solution to this problem is to let each module have an amount of *shadow* memory equal to that of state-variable—or *object*—memory. Swapping object and shadow can be done instantly, and copying object to shadow or vice versa can be done in a single step time for the whole array. In other words, one would replace bulk serial transfer by parallel back-up *in situ*. The shadow memory represents a moderate additional investment; moreover, its use is not limited to back-up. There are many cases, both during simulation and analysis, where even a one-level stack for the state-variables would greatly enhance the machine's power and flexibility.

Secondly, initialization data must be *generated* by a program as well as transfered to the state-variables; in general, this may take much longer than the transfer itself. However, in many cases a large share of the initialization burden can be carried by the modules themselves, thus bypassing serial generation and transfer of data. In fact, distinguished, explicitly constructed initial patterns are usually specified only for a small portion of the array, while the rest is initialized with a "filler" of low information content (a "vacuum" of all zeroes, a uniformly random texture, a regular grid, etc.). We plan to provide the modules with a number of self-initialization modes covering the most common requirements; in particular, we may provide an adjustable-threshold random-number generator.

Finally, the need for read-out and off-line analysis of the system's state can be reduced by using a number of on-line analysis techniques.

4.4 Stepping

It should be kept in mind that, even though within a module operations are pipelined, from a functional viewpoint the module behaves as a *parallel* device synchronously updating all the cells of its area. For this reason, a

simulation or analysis step must be treated as an indivisible, uninterruptible operation; that is, the registers containing the configuration parameters governing that step cannot be modified during the step itself, and the registers containing status parameters accumulated during a step will not have valid data until the end of that step. Thus, a complete cycle entails three phases:

1. Write configuration registers.
2. Perform step.
3. Read status registers.

While each module contains a small amount of configuration and status information, the time needed to write or read this information for *all* modules will add up to a significant fraction of the time needed to perform a step. To avoid degrading the machine's performance, register data will be double-buffered in the module, and the three phases of the above cycle will be performed in a pipelined, overlapped fashion. That is, while time-step t is in progress—and thus has no use for the control bus—the host will collect through the bus status data from time-step $t-1$, perform routine analysis functions on this data, and then ship configuration data for time-step $t+1$. In this way, the full bandwidth of the control bus is exploited not only during bulk I/O transfers, when the simulation is stopped, but also during the simulation.

A well-known drawback of such a pipelined scheme is that "commands" for time step $t+1$ will have to be issued before the results from time step t have been analyzed. Occasionally, one will discover *a posteriori* that the command just issued is not the appropriate one; hence the need for backtracking discussed in Section 4.3.

4.5 Display

The basic display mode of CAM-7 will be through raster-scanned RGB monitors operating synchronously with the sequential scanning of a module. In this mode, the state of any plane or small group of bit-planes is color-coded by a display map and directly sent to a monitor. No frame buffering or frame grabbing will be necessary in this mode.

The most challenging display problems arise when the machine is configured for three dimensions. Given the large data rate of CAM-7, the reduction of a three-dimensional image to a two-dimensional projection by conventional ray tracing is out of the question, since adequate ray tracing would require computational resources orders of magnitude greater than those of the machine itself.

CAM-7's output is produced in a format that is in principle directly usable for the presentation in real-time of of a true three-dimensional image. Here, one problem is to identify a display device able to handle this information at the required data rate. Another problem is to preprocess the image—by highlighting and shading edges, surfaces, and other manifolds of low dimensionality—so that the human eye will be able to perceive it in an intelligible way. We have considered a number of approaches to the above problems, and we will refine and evaluate them in terms of effectiveness, practicality, and cost.

4.6 Error handling

We have given preliminary consideration to the problem of error-handling. The appropriate architectural and procedural solutions, such as error detection, error correction, self-test, and experiment verification by statistical means, will be selected after reviewing with the contractors the machine's tentative specifications, and discussing with them conceivable error modes and likely error rates.

Chapter 5

Plan of work

Final design and realization of CAM-7 will be structured as six major tasks, of which the first three will progress concurrently.

(A) *Identification of an appropriate set of neighborhood options, using an emulator.* One of the most important design issues concerns the choice of which neighbors or group of neighbors should be made available as arguments to the look-up tables, so as to configure the machine for a given cellular automaton "format." Many questions concerning this issue have already been answered in the design of CAM-6; additional questions presented by CAM-7 arise from its large-scale and its emphasis on three-dimensional simulations.

We will be able to address these questions immediately, by assembling out of a number of CAM-6 modules (which are already in commercial production and are quite inexpensive) a small three-dimensional machine on which it will be possible to emulate and evaluate different sets of options.

(B) *Identification of appropriate interface, control, and I/O protocols.* Another important design issue arises from the combination of size and speed of the proposed machine. While the internal mechanics of the modules and their logical interconnection are well understood and have been tested on previous machines, putting together a large number of modules may present particular problems in the areas of overall interfacing and control, with particular stress on bulk I/O that

will make the totality of the information contained in the modules accessible in a reasonable time by the host computer.

Issues of interface and control, as well as flexibility and expandability, will be substantially influenced by our choice—out of several workable schemes—of how to vary the size and shape of CAM's array.

The study of these problems will proceed concurrently with Points *(A)* and *(C)*, since the three tasks are to a great extent decoupled. At an appropriate moment, we plan to release a "Summary of CAM-7's architecture," and to invite selected research groups to present proposals for significant initial uses of the machine. Timely feedback from these proposals may influence some final architectural details.

(C) *Design of the VLSI chips.* The project will involve the design of one or two semi-custom VLSI chips, in addition to a few FPLA's (Field-Programmable Logic Arrays). For this, we plan to use tools and services that are readily available (computer aided design and simulation, gate and function libraries, etc.)

(D) *Assembly of a reduced-scale prototype* As a milestone, and a demonstration of the major functional elements of CAM-7, we plan to assemble a 1/16 size prototype. In this prototype, the storage and pipeline sections will be essentially in their final form and will utilize one of the VLSI chips, while the transition function section, which may undergo further refinement, may be limited to essential functions and be realized with discrete IC's.

(E) *Construction of the full machine.* The full machine will consist of an assembly of 1024 modules, coupled to a suitable host computer by a high-bandwidth control and data interface. Modules will be grouped into cards, each one provided with a module controller and a bus interface; in turn, cards will be hosted by a cage/backplane assembly. Given the size of the project, cards and backplane will entail a substantial amount of design, assembly, and testing work.

(F) *Evaluation of the machine; sample experimental runs.* We will finally start working on the machine as *users*, stressing sample experimental runs that will exercise the machine's features and evaluate its

performance and flexibility. At this stage we will emphasize developement of user-oriented software, and we may experiment with additional display and analysis devices ("transducers") and techniques. We may also implement additional hardware options to give higher performance in certain applications. These options require experimentation and evaluation; while provisions for their addition have been made in the initial design, we do not want them to delay the realization of the main phase of the project.

In the evaluation of the machine we plan to solicit significant test problem and collaboration from selected users in several research areas.

Appendix A

Reference material

A.1 The CAM family of cellular automata machines

Versions of CAM have been in use for several years, and have been extensively demonstrated. Charles Bennett of IBM Research still nurtures a vintage CAM-2, with which he and his colleagues have obtained many results. CAM-5, now existing in several copies, is our current workhorse, and has been used for extensive investigations by Norman Packard and Stephen Wolfram[10] and Gérard Vichniac[18], and for other studies of interest in theoretical physics (for instance, to implement models suggested by Pomeau[6] and Creutz[2]). Popular articles related to CAM-5 have appeared in *Scientific American*[7], *High Technology*[17], and *Discover*[11]. CAM-5 is on permanent exhibit at the Boston Computer Museum.

At the Cellular Automata workshop held in Los Alamos during March of 1983[3] we demonstrated CAM-3, a high-performance cellular automata machine. Many of the participants expressed interest in arranging to have copies of this machine produced so that they could use them in their work. This led us to look for an effective way to transfer this technology to the user community.

Last year we commissioned Systems Concepts, of San Francisco, CA, to produce CAM-6—with the explicit intention that, after fulfilling

MIT's internal needs, further output of the production line would be made available to the scientific community as inexpensively as possible.[1]

This machine is intended to serve as a laboratory for experimentation, a vehicle for communication of results, and a medium for real-time, interactive demonstrations of visual impact.

CAM-6 consists of a module that plugs into a single slot of the IBM-PC, -XT, or -AT, and of driving software operating under PC-DOS 2. While this readily-available host computer provides housing, shielding, power, and a standard operating environment, the real work of simulating cellular automata at a very high speed is all done by the module itself, with a performance comparable—for this specific application—to that of a CRAY-1.

The CAM-6 module consists of two 6-layer printed-circuit boards, piggy-backed on one another. Each module can simulate a two dimensional space of 64K cells arranged on a 256x256 grid with 4 bits of state per cell, or 128K cells configured as 256x512x2. Up to eight modules can be used together to provide more states at each site or bigger spaces (up to 2048x512x2). In all cases the machine can update the entire space *60 times per second*, with simultaneous display on a color (RGB) monitor.

Each module contains 256K bits of cell-state memory, eight look-up tables of 4K bits each for the transition function and related real-time computations, extensive multiplexing for source selection (which neighbors, time-dependent parameters, and external signals are used by the look-up tables[2]), a color-map table for controlling the display, and display multiplexing that allows CAM and PC to share the same color monitor, if desired.[3]

---

[1] The description of CAM-5 in [13], together with Norman Margolus's discussion on "partitioning" neighborhoods[8] may be used for the moment to get an idea of CAM-6's functionality—though the latter differs from CAM-5 in many respects.

[2] Besides the familiar Moore and von Neumann neighborhoods and combinations or variants thereof, there are hardwired provisions for partitioning neighborhoods[8], which are quite useful, for example, in creating rules with particle-conserving collisions.

[3] CAM drives directly an IBM-PC compatible RGB monitor. If CAM and the PC use separate monitors, the monitor and display controller for the PC can be either color or

For users interested in specialized applications—or for those who want to use the CAM module as an OEM building block—the relevant inputs and outputs are brought out to a connector. Source selection includes user modes, where user-provided, external circuitry can be made an integral part of the update loop.

CAM-6 fills a void in the computing machinery spectrum: for work on cellular automata, this machine has a performance vs price advantage of several orders of magnitude over machines not specifically designed for this application.

A.2 Case study: Fully-parallel array

Consider, as a simple but fairly representative case, a cellular automaton where each cells requires two bits of memory to encode its state and several dozen gates to realize its transition function; this corresponds to some 200 transistors. On a large VLSI chip, containing on the order of 1 million transistors, one could fit a patch of 64x64=4096 (4K) cells. In turn, a two-dimensional array of 64x64 chips would yield a total of about 16 million cells. At a clock rate of, say, 32 nsec, the array would be capable of *500,000 Giga-EPS!*

There are four major problems with this architecture. We shall briefly review them.

Interconnection density: Consider the interface between two adjacent chips in the array. There are 64 cells on the edge; each cell must send two bits of data to its neighbor across the edge, and receive two bits of data from it, for a total of 4 lines. Thus, aside from power, ground, and clock, each chips needs 256 contacts along each edge, to be mated with the corresponding contacts on the adjacent chip. For a one-inch chip, the spacing between contacts would be 100 microns—too close for direct surface-mounting on a substrate, with today's technology. The conventional technique of running a wire from a bonding pad on the chip to a lead of a much larger carrier

black-and-white.

would work, but we are contemplating now a chip with one-thousand discrete bonding pads, which would waste an emormous amount of silicon real-estate, and a carrier with many long rows of pins on either side—measuring at least 4 inches across. The whole array would cover a 20-ft square and even with a reasonable amount of folding it wouldn't be easy to reliably distribute a 32 nsec clock!

Thus, in the fully-parallel architecture, interconnection poses a major obstacle—even in the two-dimensional case. If chips were to be connected as a three-dimensional array, signals from all 4096 cells on the flat sides as well as the 64 cells on the edges would have to be brought out of the chip. We would then have a matrix of 16-thousand contacts on each side of the chip, to be mated somehow with the corresponding contacts of the chips above and below. In conclusion, the more parallelism one puts in a single-chip, the harder it becomes to connect several of these chips in parallel, and the attractive aspects of very-large-scale integration are lost.

Trade-offs between speed and size: A typical application of a large cellular automaton is to model a piece of a physical system in a direct way; that is, each cell of the automaton represents a small volume element of the system. In the simulation, the characteristic speed of the fastest relevant signals within the system (the speed of light, for microscopic physics; the speed of particles, for fluid dynamics; the speed of contagion, for an epidemic) is determined by the speed of propagation of information within the cellular automaton—one cell per time step. In a closed system, the novelty value of a signal produced by some significant event decreases as the news reverberates through and through; the time taken by a signal to traverse the system several times gives a rough estimate of the time necessary to attain equilibrium.

With the fully-parallel array considered in this section, which is only 4096-cell wide, a signal will traverse the whole cellular-automaton in less than 100 μsec, and in one second will have looped through several thousand times. In other words, most experiments aimed at studying non-equilibrium behavior will last a fraction of a second, and those concerned with equilibrium properties will be over in a few seconds.

To sum up, for many applications the fully-parallel architecture invests on speed more resources than are justified by the relatively small size; a machine of this kind would lie idle most of the time. One would rather have many more cells even at the cost of slower processing, up to the point where a significant experiment may last several hours.

Display: The above considerations also apply to the problem of displaying or otherwise extracting information *in parallel* from the simulation for evaluation and further processing. Real-time visual display from such a fast parallel array is virtually impossible (until such time when each cell can be equipped *in situ* with a light-emitting device), and at any rate experiments lasting a fraction of a second would not be rewarding from a visual viewpoint. Of course, at the end of an experiment data can be shifted out of the array in a serial way for more conventional processing, but in this case the internal architecture of the machine is of little relevance.

Flexibility: Perhaps the most significant limitation of a fully-parallel array is its lack of flexibility. Our discussion so far has assumed that the transition function of a cell can be realized with a few dozen gates. This can be done only if one knows *a priori* which specific cellular automaton or small class of similar cellular automata one wants to realize. On the other hand, we would like to be able to program the array so as to realize *any* cellular automata law allowed by the wired-in neighborhood interconnections, or at least a very large class of them. In other words, the next state of a cell should depend on that of its neighbors not through a small, fixed combinational network, but through a look-up table that can be initialized at the beginning of a simulation run. The number of bits in this table will be $N = s^{n+1} \log s$ (where s is the number of states per cell and n the number of neighbors). Practical values for N range from a few hundred to a few millions.

To sum up, in a general-purpose machine the memory required for the transition-function look-up table vastly overwhelms that required for the state variables of each cell, and the number of cells one can fit on a single chip shrinks from 4096 to esssentially 1![4]

[4] In a variant of this architecture, the entries of the look-up table (which is identical for

In conclusion, the fully-parallel architecture may be attractive now for certain asynchronous (unclocked) cellular automata schemes with very simple laws (few transistors per cell)—perhaps certain versions of soft-circuitry.[5] This architecture may become more attractive sometime in the future, when (a) a technology is available for glueing edge-to-edge chips with thousands of contacts per side, (b) we know *individual* cellular automata (presumably arrived at by using a general-purpose machine) that are worth casting in silicon because the specific systems they model have important applications (one can think of a large silicon wind-tunnel[4]), and (c) it will be practical to make arrays containing millions rather than thousands of chips, and/or fabricate and interconnect chips containing millions of cells, so as to make machines whose size is better matched to their speed.

---

all cells) are broadcast serially to the whole array at each time step. Individual cells can again be made very small without loss of flexibility, but operation of the machine is slowed down by a factor of a few hundred to a few millions. Interconnection problems remain the same.

[5]VLSI chips whose circuit is "downloadable" as software. Individual cells of the cellular automaton would become at will gates, pieces of wire, or memory elements depending on the initial setting of certain state bits.

Bibliography

[1] E. F. Codd, *Cellular Automata*, Academic Press (1968).

[2] M. Creutz, *Phys. Rev. Lett.* **50** (1984), 1411.

[3] D. Farmer, T. Toffoli, and S. Wolfram (ed.), *Cellular Automata*, North-Holland (1984).

[4] U. Frisch, B. Hanslacher, and Y. Pomeau, "A Lattice Gas Automaton for the Navier-Stokes Equation," Preprint LA-UR-85-3503, Loas Alamos National Laboratory (1985), submitted *Phys. Rev. Lett.*

[5] M. Gardner, "The fantastic combinations of John Conway's new solitaire game 'life', " *Sc. Am.* **223**:4 (1970), 120–123.

[6] J. Hardy, O. de Pazzis, and Y. Pomeau, *Phys. Rev.* **A13** (1976), 1949.

[7] B. Hayes, "The cellular automaton offers a model of the world and a world unto itself," *Scientific American* **250**:3 (1984), 12–21.

[8] N. Margolus, "Physics-like models of computation," *Physica* **10D** (1984), 81–95.

[9] N. Margolus, T. Toffoli, and G. Vichniac, "Cellular Automata Supercomputers for Fluid Dynamics Modeling," MIT Lab. for Comp. Sci., Tech. Memo LCS-TM-296 (December 1985), submitted to *Phys. Rev. Lett.*

[10] N. H. Packard and S. Wolfram, "Two-dimensional cellular automata," *J. Stat. Phys.* (March 1985).

[11] C. Reiter, "Life and death on a computer screen," *Discover* (August 1984), 81–83.

[12] T. Toffoli, *Cellular Automata Mechanics*, Tech. Rep. no. 208, Comp. Comm. Sci. Dept., The University of Michigan (1977).

[13] T. Toffoli, "CAM: A high-performance cellular-automaton machine," *Physica* **10D** (1984), 195–204.

[14] T. Toffoli, "Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics," *Physica* **10D** (1984), 117–127.

[15] T. Toffoli, N. Margolus, *Cellular Automata Machines: a new environment for modeling* (to be published by MIT Press, Fall 1986).

[16] T. Toffoli, N. Margolus, "Invertible cellular automata," (in preparation).

[17] J. B. Tucker, "Cellular automata machine: the ultimate parallel computer," *High Technology* 4:6 (1984), 85–87.

[18] G. Y. Vichniac, "Simulating physics with cellular automata," *Physica* **10D** (1984), 96–115.

[19] S. Wilson, Q. Holmes, and T. Limperis, "A new machine vision system called "PIXIE," Tech. Rep., Applied Intelligent Systems, Ann Arbor, MI (ca. 1984).

[20] S. Wolfram, "Statistical mechanics of cellular automata," *Rev. Mod. Phys.* **55** (1983), 601.