International Journal of Theoretical Physics, Vol. 21, Nos. 3/4, 1982

New Computer Architectures and Their Relationship to Physics or Why Computer Science Is No Good[1]

W. Daniel Hillis,

Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts

Received May 6, 1981

In the past, computation scientists have found it convenient and productive to adopt a model of the computational universe that was very different from our models of the physical universe. This is changing. As we build bigger computers out of smaller components, our models of computation are forced to change. There is reason to hope that our new models, for specific systems, will be similar to the models of physics. The paper is divided into three sections. The first argues that computer science is missing many of the things that make the laws of physics so powerful—locality, symmetry, invariance of scale. This is why physics is so nice and computer science isn't. The second section gives an example of a new-wave computing machine, and shows some physicslike laws that apply to its computations. The final section gives some reasons for expecting this convergence of physical and computational law.

1. WHY COMPUTER SCIENCE IS NO GOOD

A computer designer is constrained by mundane problems that have no counterparts in the theoretical models of computer science: the size of connectors, the cost and availability of components, the mechanical layout of the system. Recently these factors have dictated a dramatic change in the

[1]This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Artificial Intelligence Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under contract with the Office of Naval Research, Contract N00014-80-c-0505. The author is supported by a fellowship provided by the Fannie and John Hertz Foundation.

way we design computers. Things don't look the same. Wires cost more than gates, software costs more than memory, and the air conditioner takes up more room than the computer. Our current models of computation are inadequate for designing or even describing our new architectures. An abstract model is powerful only when it allows us to pay attention to certain aspects of a situation while ignoring others. Our current models seem to emphasize the wrong details.

The areas where computational models are weak are often the areas where they differ from physical models. In physics, for example, many fundamental quantities are conserved, whereas in our old models of computation data can be created or destroyed at no cost. This is a difference and a weak point. The big air conditioner sitting next to the small computer is testimony to this fact. Other differences in physical and computational models also seem to cause problems. I will point to only one sort of difference here, the difference in locality, although similar arguments could be made for symmetry, linearity, or continuity.

In the physical universe, the effect that one event has upon another tends to decrease with the distance in time or in space between them. This allows us to study the motions of the Jovian moons without taking into account the motion of Mercury. It is fundamental to the twin concepts of Object and Action. Locality of action shows itself in the finite speed of light, the inverse square of fields, and in macroscopic statistical effects like rates of reaction and the speed of sound. In computation, or at least in our old models of computation, an arbitrarily small event can, and often does, cause an arbitrarily large effect. A tiny program can clear all of memory. A single instruction can stop the machine. In computation there is no analog of distance. One memory location is as easily influenced as another.

Fundamental to our old conception of computation was the idealized connection, the wire. A wire, as we once imagined it, was a marvelous thing. You put in data at one end and simultaneously they appear at any number of useful places throughout the machine. Wires are cheap, take up little room, and don't dissipate power.

Lately, we have become less enamored with wires. As switching components become smaller and less expensive we begin to notice that most of our costs are in wires, most our space is filled with wires, and most of our time is spent transmitting from one end of the wire to the other. We are discovering that it appeared as if we could connect a wire to as many places as we wanted, only because we did not yet want to connect to very many places. We have been forced to notice that we cannot measure a signal without disturbing it, so, for example, we must drive a wire with power proportional to the number of inputs that sense it. Of course we knew this before, but the fact seems more significant when the number in question is ten million, instead of just ten. Also, real wires take up room. Since we are building in mere 3-space, it is impractical to connect components arbitrarily. When we were wiring up a few hundred vacuum tubes this was not a problem, but today we need to wire together hundreds of millions of components and we need to do it in a smaller space. Most of the wires must be short. There is no room for anything else. (There are also similar problems with memory locations, which are just wires turned sideways in time.)

Our models of computation do not offer much help in solving the problem. Until recently, they abstracted the wire away into a costless and volumeless idealized connection. Our old models impose no locality on connections, even though the real world does. This is a prime example of where our old models break down. In classical computation the wire is not even considered. In current engineering it may be the most important thing. Something is wrong with the theory.

2. AN EXAMPLE OF NEW WAVE COMPUTER ARCHITECTURE

In this section I will describe a new type of computing machine, with some laws of behavior that are similar in form to physical laws. The intended application of the machine has nothing to do with physics. It is designed for knowledge retrieval and deduction operations, a problem of artificial intelligence.

Artificial intelligence is the study of making machines smart. It is a field limited by its tools. Currently, our mechanisms are too slow. A typical AI program, written today, knows only a few hundred facts. We would like to increase this to a few million, but the programs already take minutes to make decisions which need to be made in seconds. Scaled up, they would take years. Von Neumann machines, even if they are built of exotic ultrafast components, are unlikely candidates for solving these problems. They are limited by the speed of light. A supercomputer inside a six-inch cube, which sends a single signal from one corner to the other, would have a cycle time of at least 1 nsec. This is less than a factor of a hundred better than currently available machines—not nearly fast enough to solve our million-scaled AI problems.

The potential solution to the problem is concurrency. Integrated circuit technology makes it economically feasible to produce millions of computing devices to work on different parts of the problem in parallel. We are now in the process of designing such a machine at the MIT AI Laboratory (Hillis, 1981). The proposed architecture, called the connection machine, is a locally connected array of processing–memory cells. A medium-sized machine, built today, might have a million such cells. Unlike the classical word-at-a-time von Neumann architecture, each object in memory has associated with it not only the hardware necessary to hold the state of the object, but also the hardware necessary to process it.

The machine will be used to manipulate information stored in semantic networks (Woods, 1975; Quillian, 1968). Semantic networks are a way of representing knowledge as labeled graphs. The vertices represent concepts and the edges represent relations between those concepts. Semantic networks are a favorite representation tool of artificial intelligence. Unfortunately, retrieving information from such a network often involves searching through the entire network—a slow job on a serial machine. Worse yet, the desired information may not even be explicitly stored in the network; it may need to be deduced. The connection machine was designed for such deduction and retrieval operations. I will not discuss here how it accomplishes this task. Instead, I will skip directly to describing the machine's physical structure.

If you were to pull out one of the printed circuit boards of the proposed connection machine, you would see an array of identical integrated circuit packages connected in a grid. Each package would be wired to only it four nearest neighbors, as in the grid shown in Figure 1. If you were to remove one of the integrated circuits and look at the chip under a microscope, you would see the same picture repeated. If you were to unfold the rack of printed circuit boards, you would see that they too were wired in a locally connected two-dimensional grid. This sort of scale invariance may be commonplace in physics, but until recently it has been unusual in computer science.

The repeating grid structure was chosen not because we like it, but because we can wire it. We might have chosen a one- or a three-dimensional

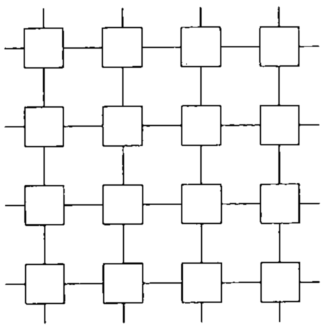

Fig. 1. At any scale, the machine is a grid.

structure, but if we had chosen, say, a four-dimensional grid or a boolean *n* cube, we would not have been able to put it together. It would have been all wires. The thing that makes us wire it as a grid at the chip level also makes us wire it that way at the board level and at the system level. The scale invariance comes naturally.

Unfortunately, our semantic network problems do not arrange themselves into nice rectangular grids. Each cell only needs to talk to a few other cells, but they are not necessarily the cells in the immediate neighborhood. The machine must have some mechanism for nonlocal communication. It would be nice if we could run a wire between any two cells that need to communicate, but we cannot, for the reasons stated above. Instead, the entire grid functions as a packet-switched communications network—a sort of postal system where messages travel from one cell to another by being relayed through the grid. Cells communicate by sending messages to each other.

A message is addressed to the appropriate cell by specifying the relative displacement in the grid of the recipient from the sender (example: up two and over five). This does not specify the route the message is to take, just its destination. The sender mails a message by handing it to a neighbor, and the neighbor decides on the basis of the address which way to send the message next. If the *y* displacement is positive, it will go up. If the *x* displacement is negative, it will go left. The neighbor modifies the address by incrementing or decrementing is appropriately, so that when the message reaches its intended destination both displacements will be zero. For example, a communicator receiving a message addressed "two up and two over" can change it to "one up and two over" and send the message to the communicator above. This is illustrated in Figure 2.

Delivering messages takes time, so the distance between communicating cells is important. The metric is not the same as in Euclidean space because there are no diagonals. The taxicab metric (delta *x* plus delta *y*) is

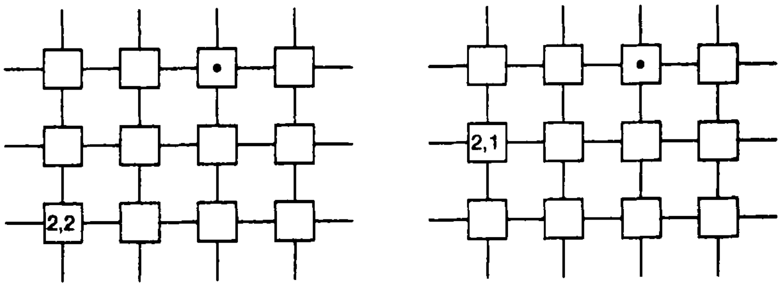

Fig. 2. A message moving toward its destination.

closer, but even this needs some refinement. The problem is that each cell has only a finite number of states, so it can only handle a few messages at one time. Messages may need to sidestep congestion. It is no problem to design local routing algorithms that will accomplish this, but the effective distance between two objects is increased. We need a metric that takes this into account.

We define the distance between two points as the average communication time between them. In an empty cellular space, this is the same as the taxicab metric. The presence of an intervening object distorts the metric because messages must flow around it. The curvature of the optimum message paths (geodesics) increases with the density of objects. The farther away the objects, the less the effect, so there is a local distortion in the metric proportional to the density of objects.

This distortion is not quite the same as physical gravity, and I will not suggest that the causes of the two are similar, but it is interesting to find an effect in computation that is so similar in form to one in physics.

Here is another one. Imagine that two cells are sitting next to each other in the grid. Imagine that the left cell communicates mostly with cells off to the right and the right cell communicates with cells to left. It would be advantageous (in the sense of minimizing communication time) if the cells were to exchange places, bringing each of them nearer to the cells with which they communicate. The hardware of the cell cannot move, but two cells can exchange all internal states. The effect is the same. The computation object that was in the right cell moves to the left cell, and vice versa. (Interested parties must be informed of the change of address, but this turns out to be easy on the connection machine.)

By this mechanism, with some refinements, the hardware of the machine causes each cell to migrate in the direction in which it sends most of its messages. Groups of intercommunicating cells will tend to cluster. In such a system, paths of communication act like attractive forces which bind the cells together. At a larger scale, the clusters act like objects. They have strong internal forces and weaker interactions with other objects. Communication between two clusters tends to pull them together. This motion is a cumulative effect of the local behavior of the individual cells, but it can be analyzed as a macroforce between two objects. There is no need to pay attention to the detailed interactions of the individual cells.

I could give a specific local rules that cause the macroforces to behave like $F = Ma$, but that would miss the point. The point is not that this is a good model of physics (it isn't), but that the laws that describe its behavior will be similar in form to physical laws. Remember that the purpose of the machine has nothing to do with physics. It was designed the way it was for good, hard engineering reasons: the cost of connectors, the need to dissipate heat, the volume of wires. Any similarity to physics, living or dead, is purely unintentional. But not coincidental.

3. NEW HOPE FOR A SCIENCE OF COMPUTATION

Progress in physics comes by taking things apart, in computation, by putting things together. We might have had an analytic science of computation, but as it worked out we learned more from putting together thermostats and computers than we did from taking apart monkey brains and frogs eyes. The science of computation, such as it is, is synthetic.

The respective models of physics and computation reflect the difference in approach. For example, in classical physics most quantities are continuous. As physicists probe deeper into lower and more fundamental levels of reality, things begin to look discrete. The physicist of yesterday measured. The physicist of today counts. In computation things are reversed. We have begun in the other direction, and, because we have begun only recently, we have not gone far. This is one of the reasons that computer science seems to be "no good"—we have not gotten beyond counting. Knowing the lowest level rules is good, but it is in no way sufficient. Quantum chromodynamics is not much use in designing bridges. Computer science is not much use in designing computers.

I am not discouraged. While physics is looking down into lower and lower levels, computer science is looking up. It is looking up because systems are becoming large enough for there to be a forest to see through the trees.

There are two sorts of things that could be called computational models, and I would like to make clear which one I am talking about. By "computational model," we could mean a model of all possible computational worlds. There have been a few important steps toward such metacomputational theory (theories of servomechanisms, Turing computability, information theory), but so far a complete and coherent model is still beyond sight. The second sense of "computational model," the one that I am using in this paper, is a model of a particular computational system. Physics may be such a model. Physical law does not need to describe what might happen in any possible universe, just this one. In computation, the distinction is more important because we design our own worlds. The connection machine described earlier is an example of such a world.

I see no way to predict the development of a generalized theory of computation, but I do see reasons to expect good, clean, useful models of specific computational systems—models that will look like physics. The first reason is that physical law itself seems to be such a model. If the universe is a computing machine, then we know that at least some computing machines have elegant laws. This view of the universe is well represented elsewhere (Landauer, 1967; Toffoli, 1977), and I will not dwell on it.

The second reason for believing in physical–computational model convergence is more profound, and therefore more likely to be wrong. Both sciences study large systems of weakly interacting components. Such systems, with local rules of interaction, often seem to have simple macro laws. There may be a "law of large systems," corresponding to the statistical "law of large numbers." The statistical law says that the sum of many random variables always has a simple gaussian distribution, whatever the distributions of the variables. A sum represents less information than its addends, and the gaussian distribution has minimum information. In the same way, when we add together the individual behaviors of components we lose information. Only the simple linear properties show through. Classical physics is simple because only simple additive properties, like momentum, remain visible at the macro scale.

The final reason for expecting physicslike behavior in computational systems is that all of our computing machines must be implemented in the physical world. As our components become smaller and more efficient, they must inherit some of the constraints of the physical laws. Machines will have three-dimensional connectivity, because space is three dimensional. They will have limited propagation rates, because space has a finite speed of light. As less is wasted between function and implementation, the physics begins to show through.

These conjectures will be tested, because in the future we will be building even larger computing machines, out of even smaller components. Perhaps we will grow crystals, with each lattice point a processor. What will computation look like with a mole of processors? Much like physics, I think. When this happens, we can look forward to new models of computation, models that may inherit some of the power and the beauty of physical law.

REFERENCES

Backus, J. (1978). "Can programming be liberated from the von Neumann style?" Communication of the ACM, Vol. 21, No. 8., pp. 613–641, (August).

Hillis, W. D. (1981). "The Connection Machine," A.I. Memo No. 646, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Landauer, R. (1967). "Wanted: a physically possible theory of physics," *IEEE Spectrum* 4, 105–109.

Woods, W. (1975). "What's in a link: Foundations for semantic networks," in *Representation and Understanding*. Academic Press, New York.

Quillian, M. (1968). "Semantic memory," in *Semantic Information Processing*, Minsky, ed. MIT Press, Cambridge, Massachusetts.

Toffoli, T. (1977). "Cellular automata mechanics," Tech. Rep. No. 208, Logic of Computers Group, CCS Dept., The University of Michigan (November).