Parallel Quantum Computation*

Norman Margolus
MIT Laboratory For Computer Science
Cambridge Massachusetts 02139

June 1989

Abstract

Results of Feynman and others have shown that the quantum formalism permits a closed, microscopic, and locally interacting system to perform deterministic *serial* computation. In this paper we show that this formalism can also describe deterministic *parallel* computation. Achieving full parallelism in more than one dimension remains an open problem.

1 Introduction

In order to address questions about quantum limits on computation, and the possibility of interpreting microscopic physical processes in informational terms, it would be useful to have a model which acts as a bridge between microscopic physics and computer science.

Feynman and others[2, 6, 10] have provided models in which closed, locally interacting microscopic systems described in terms of the quantum formalism perform deterministic computations. Up until now, however, all such models implemented deterministic *serial* computation, i.e., only one part of the deterministic system is active at a time.

*This research was supported by the Defense Advanced Research Projects Agency and by the National Science Foundation.

We have the prejudice that things happen everywhere in the world at once, and not sequentially like the raster scan which sweeps out a television picture. It would be surprising, and perhaps a serious blow to attempts to ascribe some deep significance to information in physics, if it were impossible to describe *parallel* computations within the quantum formalism.

In this paper, we extend the discussion of a previous paper[10] to obtain for the first time a satisfactory model of parallel "quantum" computation, but only in one dimension. The two-dimensional system discussed in [10] is also shown to be a satisfactory model, but the technique used here only allows one dimension to operate in parallel: the more general problem of the possibility of fully parallel two- or three-dimensional quantum computation remains open.

2 Computation

The word *computation* is used in many contexts. Adding up a list of numbers is a kind of computation, but this task requires only an adding machine, not a general purpose computer. Similarly, we can *compute* the characteristics of airflow past an aircraft's wing by using a wind tunnel, but such a machine is no good for adding up a list of numbers.

An adding machine and a wind tunnel are both examples of computing machines: machines whose real purpose is not to move paper or air, but to manipulate information in a controlled manner. It is the rules that transform the information that are important: whether the adding machine uses enormous gears and springs, or microscopic electronic circuits, as long as it follows the addition algorithm correctly, it is acting as an adding machine.

A *universal computer* is the king of computing machines: it can simulate the information transformation rules of any physical mechanism for which these rules are known. In particular, it can simulate the operation of any other universal computer—thus all universal computers are equivalent in their simulation capabilities. It is an unproven, but thus far uncontradicted contention of computer theory that no mechanism is any more universal than a universal digital computer, i.e., one that manipulates information in a discrete form.

Assuming a finite universe, no machine can have a truly unbounded memory; what we mean when we talk about a *general purpose computer* is a machine that, if it could be given an unbounded amount of memory, would be a universal computer. (In common usage, the terms *general purpose computer* and *computer* are synonymous.) Similarly, when we talk about a finite set of logic elements as being *universal*, we mean that an unbounded collection of such elements could constitute a universal computer.

An adding machine is not a general purpose computer: a certain minimum level of complexity is required before universal behavior is possible. This complexity threshold is quite low: aside from memory, a few dozen logical NAND gates, suitably connected, can be a computer. On the other hand, some modern computers contain millions of logic elements in their central processors: this doesn't let these computers solve any problems that the humblest microcomputer couldn't solve; it simply lets them run faster. Except for speed and memory capacity, there is no difference in computational capability between a Cray-XMP and an IBM-PC.

3 Quantum computation

Although all general purpose computers can perform the same computations, some of them work faster, use less energy, weigh less, are quieter, etc., than others. In general, some make better use of the computational opportunities and resources offered by the laws of physics than do others. For example, since signals travel so slowly (it takes about a nanosecond to go a foot, at the speed of light), there is a tremendous speed advantage in building computers which have short signal paths. Modern microprocessors have features that are only a few hundred atoms across: such small components can be crowded close together, allowing the processor to be small, light, and fast.

As we try to map our computations more and more efficiently onto the laws and resources offered by nature, we are eventually confronted with the question of whether or not we can arrange for extremely microscopic physical systems to perform computations. What we ask here is in a sense the opposite of the hidden variables question: we ask not whether a classical system can simulate a quantum system in a microscopic and local manner, but rather, whether a quantum system can simulate a classical system in such a manner.

All of our discussion of quantum computation will be based on autonomous systems: we prepare the initial state, let the system undergo a Schrödinger evolution as an isolated system, and after some amount of time we examine the result.[1] Since the Schrödinger evolution is unitary, and hence invertible, we must base our computations on reversible logic[7].

4 Reversible computation

Until recently, it was thought that computation is necessarily irreversible: it was hard, for instance, to imagine a useful computer in which one could not simply erase the contents of a register. It was to most people a rather surprising result[3, 7, 8, 9, 13] that computers can be constructed completely out of invertible logic elements, and that such machines can be about as easy to use as conventional computers. This result has thermodynamic consequences, since it turns out that a reversible computer is the most energy efficient engine for transforming information from one form to another. This result also means that computation is not necessarily a (statistically irreversible) macroscopic process.

As an example of an invertible logic element, consider the *Fredkin gate* of Figure 1. This gate is in fact its own inverse (two connected in series give the identity function), and this gate is a universal logic element: you can construct any invertible logic function out of Fredkin gates. A logic circuit made out of Fredkin gates looks much like any conventional logic circuit, except that special "mirror image circuit" techniques are used to avoid the accumulation of undesired intermediate results that we aren't allowed to simply erase (see [7] for more details).

Feynman made a quantum system simulate a collection of invertible logic gates connected together in a combinational circuit (i.e., one without any feedback).[2] In Feynman's construction, only one logic element was active (i.e., transforming its inputs into outputs) at any given time: the different gates were activated one at a time as they were needed to act on the output of gates that were active earlier. We can imagine a sort of "fuse" running through our circuit: as the active part of the fuse passes each circuit element

[1]For some types of computations, we can't set a very good limit on how long we should let it run before looking. In such cases, we would simply start a new computation if we look and find that we aren't finished.

[2]Although combinational circuitry can perform any desired logical function, computers are usually constructed to run in a cycle, reusing the same circuitry over and over again. The parallel models discussed later in this paper run in a cycle.

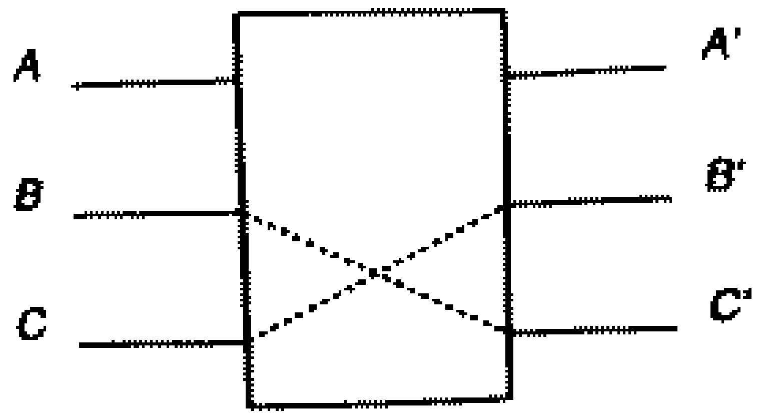

Figure 1: Fredkin gate. The top *control* input goes through unchanged ($A' = A$), and the bottom inputs either go straight through also (if $A = 1$) or cross (if $A = 0$ then $B' = C$ and $C' = B$).

in turn, it activates that element. Using a collection of two-state systems (which he called *atoms*) to represent bits, Feynman made a "quantum" version of this model. In what follows, we will think of our two-state systems as spin-$\frac{1}{2}$ particles.

5 Feynman's quantum computer

In 1985, Richard Feynman[6] presented a model of computation which was quantum mechanically plausible: there seems to be no fundamental reason why a system like the one he described couldn't be built.[3] In his idealization, he managed to arrange for all of the quantum uncertainty in his computation to be concentrated in the time taken for the computation to be completed, rather than in the correctness of the answer. Thus if his system is examined and a certain bit (state of a spin) indicates that the computation is done, then the answer contained elsewhere in the system is always correct. What's more, he managed to make his computation run at a constant rate.

His system consists of two parts: a collection of reversible logic gates,

[3] Less physical models were proposed earlier by Benioff[1], who seems to have been the first to raise the question of quantum computation in print.

each made up of several interacting spins, and a chain of "clock" spins which passes next to each gate in turn. Note that we will think of each wire that runs between two gates as being a very simple reversible gate: one that exchanges the values of the spins at its two ends. In this way we are able to write down a unitary operator F_k that describes the desired behavior of the k-th gate: for a given invertible gate such as the Fredkin gate or a wire, we can write this operator down explicitly in terms of raising and lowering operators. For example, for a wire F_i joining spin a of gate m and spin b of gate n, the rule is

$$F_i = a_m b_n^\dagger + a_m^\dagger b_n + a_m a_m^\dagger b_n b_n^\dagger + a_m^\dagger a_m b_n^\dagger b_n$$

where a and b are lowering operators at the two spins, and $a^\dagger$ and $b^\dagger$ are their Hermitian adjoints, which are raising operators on the two spins.

Without any claim yet to a connection with quantum mechanics, we can cast the overall logical function implemented by an N-gate invertible combinational logic function into the language of linear operators acting on a tensor product space as follows:

$$F = \sum_{k=1}^{N} F_k c_k c_{k+1}^\dagger \tag{1}$$

where c_k is the lowering operator on the clock spin that passes next to the k-th gate F_k. If we start all of the clock spins off in the down state except for the spin next to the first gate, then if F acts on this system, only the term

$$F_1 c_1 c_2^\dagger$$

will be nonvanishing. This term will cause the spins acted upon by the first gate to be updated, the first clock spin will be turned down, and the second clock spin will go up. Similarly, if F acts again, the second gate will update, and the up spin will move to the third position. Clearly if the initial state has only a single clock spin up, F will preserve that property. Using the position of the up clock spin to label the state, then if $|1\rangle$ is the initial state, $F|1\rangle = |2\rangle$, and in general $F|k\rangle = |k+1\rangle$. We have thus been able to write the forward time-step operator as a sum of local pieces by serializing the computation—only one gate in the circuit is active during any given step.

Notice that the operator $F_k^\dagger$ is the inverse of F_k, since the role of raising and lowering operators is interchanged. Similarly, $F^\dagger$ is the inverse of F, since each term of the former undoes the action of the corresponding term of the latter, including moving the clock spin back one position. Now if we add together the forward and backward operators, we get an Hermitian operator $H = F + F^\dagger$ which is the sum of local pieces, each piece acting only on a small collection of neighboring spins (a gate). At this point we make contact with quantum mechanics, by seeing what happens if we use this H as the Hamiltonian in a Schrödinger evolution.

If we expand the time evolution operator $U(t) = e^{-iHt}$, we get

$$U(t) = 1 - iHt - \frac{H^2t^2}{2} + \cdots = 1 - i(F + F^\dagger)t - \frac{(F + F^\dagger)^2t^2}{2} + \cdots$$

and so we get a sum of terms, each of which is proportional to F or $F^\dagger$ to some power. Thus if $|k\rangle$ is evolved for a time t, it becomes $e^{-iHt}|k\rangle$ which is a superposition of configurations of the serialized computation which are legitimate successors and predecessors of $|k\rangle$: each term in the superposition has a single clock spin at some position, and the computation is in the corresponding state.

Feynman now noted that the operators F_k don't affect the dynamics of the c_k's: we can consider $F = \sum_{k=1}^{N} c_k c_{k+1}^\dagger$ for the purposes of analyzing the evolution of the clock spins. But then $H = F + F^\dagger$ supports superpositions of the one-spin-up states called spin waves, as is well known. When we add back in the F_k's, the computation simply rides along at a uniform rate on top of the clock spin waves. This point will be discussed in more detail below, when we extend this serial model to deal with parallel computation.

6 Parallel computation

Serial computers follow an algorithm step by step, completing one step before beginning the next; parallel computers make it possible to do several parts of the problem at once in order to finish a computation sooner. Although Feynman's construction is based on a serial model, his idea of concentrating all of the quantum uncertainty into the time of completion, while leaving none in the correctness of the computation, can be extended to parallel computations[10]. Maintaining correctness is again achieved simply by construction of the Hamiltonian: states in the Hilbert space that correspond to configurations on a given computational orbit form an invariant subspace under the Schrödinger evolution. This property of the Hamiltonian does not, in general, say anything about the rate at which we can compute. Here we show that Feynman's technique for making a serial model of quantum computation run at a constant rate can, in fact, also be extended to apply to a parallel system, in particular to the one-dimensional analogue of the case considered in [10]. From this, we can derive a way of making the two-dimensional system considered in [10] compute at a constant rate, but with parallelism that extends over only one dimension.

For simplicity, our discussion of parallel computers will be confined to cellular automata (CA): uniform arrays of computing elements, each connected only to its neighbors. These systems can be universal in the strong sense that a given universal cellular automaton (assuming it's big enough) can simulate any other computing structure of the same dimensionality at a rate that is independent of the size of the structure.[4] By showing that, given any desired (synchronous) CA evolution, we can write down a Hamiltonian that simulates it, we will have shown that the QM formalism is computationally universal in this strong sense, at least for one-dimensional rules.

Feynman's model involved only states in which a single site was active at a time. In order to accommodate both neighbor interactions and parallelism in quantum mechanics, we find that we are forced to consider asynchronous (no global time) computing schemes (but still employing invertible logic elements). For suppose that our Hamiltonian is a sum of pieces each of which only involves neighbor interactions

$$H = \sum_{x,y,z} H_{x,y,z} \tag{2}$$

Then consider the time evolution $1 - iHt$ over an infinitesimal time interval. When this operator acts on a configuration state of our system, we get a superposition of configuration states: one term in the superposition for every term in the sum (Equation 2) above. If we want all of the terms in this superposition to be valid computational states, then we must allow configurations in which one part has been updated, while everything else has been left unchanged.

[4] This isn't the usual definition of universality in CA, but it is the one that we'll use here.

7 Local synchronization

One can perform an effectively synchronous computation using an asynchronous mechanism by adding extra state variables to keep track of relative synchronization (how many more times one portion of the system has been updated than an adjacent portion). To use an analogy, consider a bucket brigade carrying a pile of stones up a hill. You hand a stone to the first person in line, who passes it on to the next, and so on up the hill. An *asynchronous* computation would correspond to every individual watching the person ahead of himself, and passing his stone along when the next person has gotten rid of theirs. This involves only local synchronization. A *synchronous* computation would correspond to having everyone pass on their stones whenever they hear the loud tick of a central clock. Notice that both schemes get exactly the same sequence of stones up the hill; only the timing of when a given stone moves from hand to hand changes.

Now let us consider a one-dimensional cellular automaton. We imagine a row of cells, each containing a few bits of state. Our evolution will consist of two phases: first, we group the cell at each even-numbered position with the cell to its right, and perform a logical transformation on the state of these two cells; then we regroup the cells so that each even-numbered cell is associated with the cell to its left, and again we update the pair. We alternate these two kinds of steps to produce a dynamics. Notice that if the transformation we perform on each pair of cells is an invertible logic function, then the overall dynamics will be invertible.

Since cells are updated in pairs, it is really unnecessary for the entire system to be globally synchronous: we can achieve effectively the same result by local means. Imagine that we take our configuration of cells, and to each cell we add an extra number, which is the number of times that cell has been updated. In a synchronous updating scheme, all cells would start out with this number set to zero, and this number would increment uniformly throughout the system: if one cell is at step 27, all cells are. But suppose we start out with the same initial data, and only update one pair (with the appropriate grouping for an even-numbered step). Since the result of this updating only depends on the contents of these two cells, it makes no difference whether or not any other cells have been updated yet. Next, we could update some more pairs. Now suppose two adjacent pairs have been updated: we have four consecutive cells that correspond to the synchronous time step number

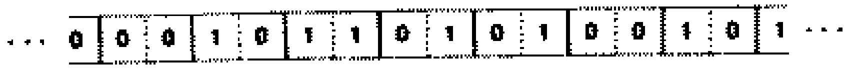

Figure 2: A section of a one-dimensional pairing automaton showing only the states of the clock bits in each cell. The solid bars bracket the pairing used for even times, the dotted for odd times.

one, and are labeled as having been updated once. The middle two cells of these four are a correct group for an odd-numbered synchronous step, and so we can update this odd pair and label them as having been updated twice. Each of these two cells is now ready to be updated again as part of even pairs, as soon as the adjacent cells catch up! Thus we can perform an asynchronous updating of pairs, using the count of updates for each cell to tell us when adjacent cells can be updated as a pair. As long as we observe this protocol, we can update cells in any order and retain the property that any cell that is labeled as having been updated n times is at the same state that it would have had if the whole system had been updated synchronously n times.

Notice that with this scheme, two adjacent cells cannot get more than one step apart in update-count: since this count is only used to tell whether a given cell is using the even step pairing or the odd step pairing, and to tell if adjacent cells are at the same step, we only need to look at the least significant bit of the update-count. Thus if we take our original synchronous automaton and add a single bit of update-count to each cell, we can run the system asynchronously while retaining a perfectly synchronous causality.

In Figure 2 we show a possible state for the update-count bits (henceforth we'll call them *clock bits*) in a one-dimensional pairing automaton of the type we've been discussing, which is consistent with an evolution starting from a synchronous initial state. In Figure 3 we use a spacetime diagram to integrate the relative time phases: arbitrarily calling the time at the left hand position $t = 0$, we mark cells using the relative time information encoded in the clock bits. As we move across, if a cell is at the same time as its neighbor to the left, we mark it at the same time on this diagram, if it is ahead, we mark it one position ahead, etc. The result is a diagram illustrating the hills and valleys of time present in this configuration. Note that we can tell if a given cell in Figure 2 which is at a different time phase than its neighbor to the left is ahead or behind this neighbor by seeing whether or not it is waiting

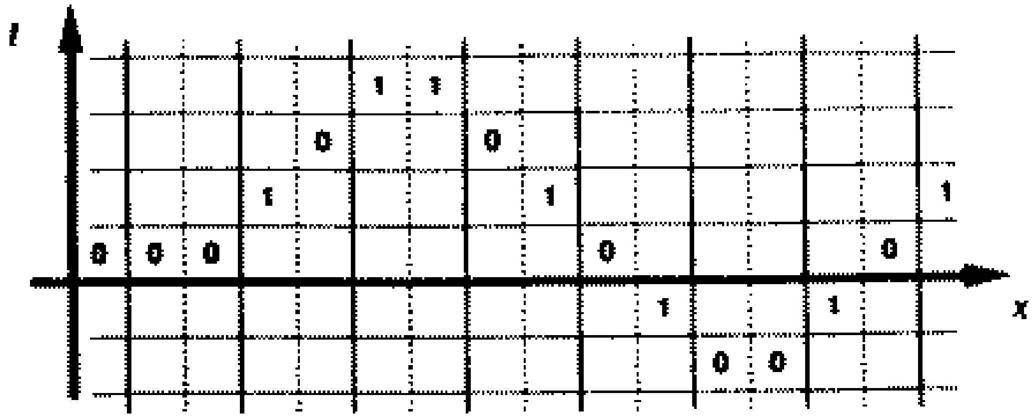

Figure 3: A spacetime diagram showing relative times of adjacent clock spins corresponding to the data in the previous figure. Pairing of cells is indicated as before.

for the neighbor to catch up in order to be paired with it.

Note that if we allow backward steps, this synchronization scheme still works fine: we can imagine that a backward step is simply undoing a forward step, getting us to a configuration we could have gotten to by starting at an earlier initial synchronous step, and running forward.

These configurations then, with their hills and valleys of time, will be the classical configurations which our quantum system will simulate.

8 A "quantum" parallel automaton

Again we imagine a collection of interacting spins as our computational system. Let $|n, \alpha\rangle$ be a state on our locally synchronized computational trajectory, where n refers to time and α refers to other information needed to uniquely specify a configuration. Since our configurations have no global moment of time, we use an integrated notion of time: we simply add up the equivalent synchronous times for all cells in the automaton, and divide by the number of cells in a single block. With this normalization, if we have a configuration at integrated time n and we take a step forward at a single block, then the resulting configuration will be at time $n+1$.

We imagine that our system has two kinds of spins at each site in our one-dimensional chain of cells: data spins and clock spins. We'll let D_i be our rule for updating the block of data spins belonging to two adjacent cells at locations i and $i+1$; $D_i^\dagger$ is the inverse rule. We imagine that we have a single spin-$\frac{1}{2}$ clock spin at each cell, and that c_i is the lowering operator acting on the spin at cell i. Now we can define F, our forward time-step operator:

$$F = \sum_{i \text{ even}} D_i c_i^\dagger c_{i+1}^\dagger + \sum_{i \text{ odd}} D_i c_i c_{i+1} = \sum_i F_i \tag{3}$$

This operator, acting on a state $|n,\alpha\rangle$, produces a superposition of states each of which belongs to time $n+1$. Similarly, $F^\dagger$ takes us backwards one time step. Note however, that $F^\dagger$ is not the inverse of F. Nevertheless, on the subspace of computational configurations (those that can be obtained by a sequence of local updatings starting from a synchronous configuration) F and $F^\dagger$ commute: this property, which will be proven below, will be crucial in our construction.

As before, we let $H = F + F^\dagger$, and if we expand the time evolution operator $U(t) = e^{-iHt}$ we get a superposition of terms, each involving products of F_i's and $F_j^\dagger$'s for various i's and j's. Since each such term, acting on a computational configuration, gives us another computational configuration (by construction of the clock bits), the time evolution U doesn't take us out of our computational subspace.

8.1 Running in parallel

Now we would like to have our parallel computation run forward at a uniform rate. We are imagining that our space is periodic: the chain of cells is finite and the ends are joined. Designating one particular state of the equivalent globally synchronous computation as $t=0$, we can assign a value of t to every configuration on each synchronous computational orbit, and from these assign a value of n to the integrated time on every locally synchronized computational configuration. Thus we can construct an operator N which, acting on a configuration $|n,\alpha\rangle$, returns n:

$$N\,|n,\alpha\rangle = n\,|n,\alpha\rangle$$

From this we can construct a *computational velocity* operator V:

$$V = \frac{[N,H]}{i} = \frac{[N,F]}{i} + \frac{[N,F^\dagger]}{i}$$

But $NF\,|n,\alpha\rangle = (n+1)F\,|n,\alpha\rangle$, since F takes $|n,\alpha\rangle$ into a superposition of states all of which correspond to time $n+1$, and so

$$[N,F]\,|n,\alpha\rangle = (n+1)F\,|n,\alpha\rangle - nF\,|n,\alpha\rangle = F\,|n,\alpha\rangle$$

and similarly, $[N,F^\dagger]\,|n,\alpha\rangle = -F^\dagger\,|n,\alpha\rangle$. Thus on this subspace,

$$V = \frac{F - F^\dagger}{i}$$

Now for the average computational velocity $\langle V\rangle = d\,\langle N\rangle\,/dt$ to be constant, we would like V to commute with H. So the question becomes, does V commute with H? Now $[V,H] = [(F-F^\dagger)/i, F+F^\dagger] = 2[F,F^\dagger]/i$ and so this is the same as the question, does F commute with $F^\dagger$?

Each term in the product $F\,F^\dagger$ involves one F_j and one $F_k^\dagger$. Clearly if $|j-k| \geq 2$, then $[F_j, F_k^\dagger] = 0$, since the two operators act on disjoint sets of spins. If $|j-k| = 1$, then the product $F_j F_k^\dagger$ vanishes when applied to a computational state, since either F_j or $F_k^\dagger$ vanishes: either the pair of cells at k and $k+1$ are not ready to take a step backwards (and so $F_k^\dagger$ vanishes), or if they are ready to go back and $F_k^\dagger$ acts on them, then in the resulting configuration these two cells are only ready to take a step forward if they are paired together again, and so F_j vanishes. Thus the commutator of F and $F^\dagger$ can be written

$$[F,F^\dagger] = \sum_k [F_k, F_k^\dagger] = \sum_k F_k F_k^\dagger - \sum_k F_k^\dagger F_k$$

which, when applied to a computational configuration, just gives the difference between the number of blocks that are ready to go backwards, and the number that are ready to go forwards. Now the question of commutation is reduced to a question about the computational configurations: "Is it true that the number of blocks ready to go forward is always equal to the number ready to go back?"

For the two-dimensional case considered in [10], the answer is no, but for a one-dimensional automaton with periodic boundaries, the answer is yes: in a flat (globally synchronous) configuration, the answer is clearly yes, and it is easy to check that any sequence of updates preserves this property.[5]

Now we can make our cellular automaton run at a uniform rate: we use as our initial state a superposition of eigenstates of V which has a fairly narrow ΔN, so that the integrated time in our computation is fairly definite.[6] Since $\langle V \rangle$ is constant, this state will evolve at a uniform rate, as desired.

8.2 Relating the models

It turns out that the one-dimensional version of Feynman's serial model is a special case of the model discussed above: if we complement the meaning of every second clock spin (say, all the ones at even positions), Equation 3 becomes

$$F = \sum_{i \text{ even}} D_i c_i c_{i+1}^\dagger + \sum_{i \text{ odd}} D_i c_i c_{i+1}^\dagger = \sum_i D_i c_i c_{i+1}^\dagger$$

which is of exactly the same form as Equation 1. An initial state containing a single *up* clock spin and all the rest down would correspond, in our parallel system of Equation 3, to all of the even clock spins up, and all of the odd ones down, except for the spin at the active position k, which is the same as its two neighbors. Since updating in our parallel model only occurs at positions where two adjacent clock spins are the same, there are only two active blocks in such an alternating configuration: the block involving k and $k+1$, which will be a step forward if updated, and the block involving k and $k-1$, which will be a step back if updated. If we draw a spacetime diagram of the clock spins around position k (see Figure 4) showing the relative synchronization implied by the alternating pattern of clock spins, we see that it forms a staircase with a landing that moves up and down in time as its leading edge or trailing edge is updated. Because the space is periodic, the top of this

[5] Equivalently, one can simply observe that between every two blocks that are ready to go forward, there is always a block that is ready to go back, and vice versa, and so in a periodic configuration the number ready to go forward is always equal to the number ready to go back.

[6] This also avoids the necessity of performing the whole computation ahead of time in order to construct the initial superposition: we simply truncate the small-amplitude long-time terms of our initial superposition, effectively adding a small error term to our state whose amplitude doesn't grow with time[16].

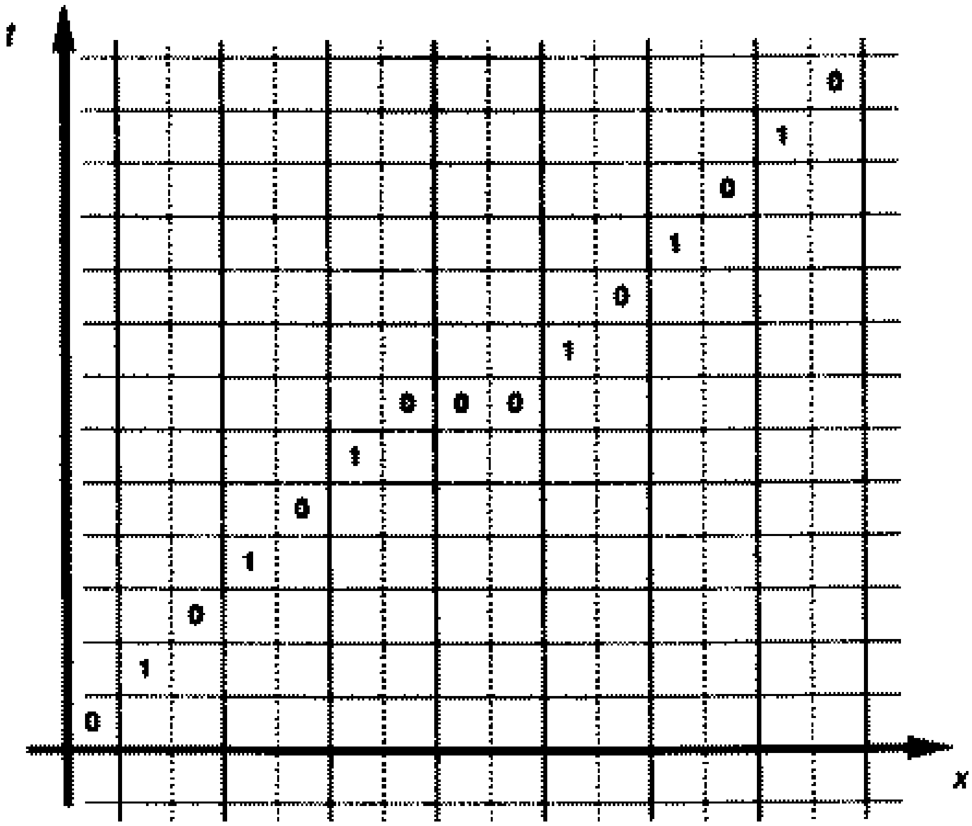

Figure 4: A spacetime diagram of the active region of a parallel one-dimensional cellular automaton with a staircase configuration.

staircase is connected to the bottom: this configuration is not on the orbit of any synchronous parallel computation.

9 Fixing the 2-d model

In [10] I gave a two-dimensional analog of the parallel model discussed here, using a particular universal reversible cellular automaton which was updated using a 2-d version of the locally synchronized block partitioning discussed above. There I was unable to make the model run at a uniform rate; the parallel technique used above can in fact be extended to make this earlier model run, but with only one dimension of parallelism. The idea is to sweep a one-dimensional parallel active region across the two-dimensional system using staircase and landing configurations analogous to what we saw in the previous section: we initialize the rows of our 2-d clock spins (they were called guard bits in [10]) with an alternating pattern of horizontal stripes (all of the even rows up, the odd rows down) except for a single row (the active region) that is the same as its two neighboring rows. Then every column contains exactly one segment with three consecutive clock spins that are the same, and in fact each column of clock spins, when represented on a spacetime diagram, looks exactly like Figure 4. It is easy to verify that this property is preserved by the dynamics, and that the dynamics of the active region is isomorphic with that of our 1-d parallel model. Thus if we make a wave packet state out of configurations on the same computational orbit as a staircase with a landing, we can make this wave packet run repeatedly across our system, doing a line of updates in parallel as it travels up the staircase.

Note that the CA model of [10] has the property that one can perform any desired computation by constructing patterns of up and down values in the data spins that resemble conventional computer circuitry: gates, signals, wires, etc. In such patterns, no signals need ever go outside a fixed-sized region. Thus the fact that a staircase configuration is not on the computational orbit of any synchronous computation doesn't mean that such a configuration can't perform an equivalent computation: the arrangement of clock spins outside of the fixed-sized region containing the circuit of interest is irrelevant as long as computation is able to proceed within this region, and as long as relative synchronization is never locally violated.

Of course what we would really like is to have a fully parallel 2-d system, but at least we now have shown that we can have parallelism in a computationally universal quantum Hamiltonian system with only neighbor interactions.

10 Conclusions

The study of the fundamental physical limits of efficient computation requires us to consider models in which the mapping between the computational and physical degrees of freedom is as close as is possible. This has led us to ask whether the structure of quantum mechanics is compatible with parallel deterministic computation. If the answer was no, then such computation would in general have to be a macroscopic phenomenon. In fact, at least in one dimension, it does seem possible to construct plausible models to simulate any locally-interacting deterministic system at a constant rate and in a local manner. The problem of finding satisfactory models of fully parallel quantum computation in more than one dimension remains open.

Physically motivated models of computation such as those considered here, in which individual degrees of freedom have both a computational and a physical interpretation, act as bridges between theoretical physics and theoretical computer science. Computers constructed (for efficiency) with a physics-like structure may be usefully analyzed using concepts and techniques imported from physics[11]; computational reinterpretations of such imported physical concepts may someday prove useful in the study of physics itself.

11 Acknowledgments

I would like to gratefully acknowledge conversations with R. P. Feynman in which he pointed out to me the relationship between my parallel model and his serial model, and discussions with L. M. Biafore in which it became evident that the one-dimensional version of my parallel QM construction might be made to run at a uniform rate.

References

[1] P. A. Benioff, "Quantum mechanical Hamiltonian models of discrete processes that erase their own histories: application to Turing machines," *Int. J. Theor. Physics* **21** (1982), 177–202.

[2] P. A. Benioff, "Quantum mechanical Hamiltonian models of computers," in the proceedings of a conference "New Ideas and Techniques on Quantum Measurement Theory," (Jan. 1986). *Ann. New York Acad. Sci.* **480** (1986), 475–486.

[3] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development* **17** (1973), 525.

[4] D. Deutsch, "Quantum theory, the Church-Turing hypothesis, and universal quantum computers," *Proc. Roy. Soc. Lond. A* **400** (1985), 97–117.

[5] R. P. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys.* **21** (1982), 467.

[6] R. P. Feynman, "Quantum mechanical computers," *Opt. News* **11** (1985).

[7] E. Fredkin, T. Toffoli, "Conservative logic," *Int. J. Theor. Phys.* **21** (1982), 219.

[8] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development* **5** (1961) 183.

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

[10] N. Margolus, "Quantum computation," in the proceedings of a conference "New Ideas and Techniques on Quantum Measurement Theory," (Jan. 1986). *Ann. New York Acad. Sci.* **480** (1986), 487–497.

[11] N. Margolus, "Physics and computation," (Ph. D. Thesis) *Tech. Rep. MIT/LCS/TR-415*, MIT Laboratory for Computer Science (1988).

[12] A. Peres, "Reversible logic and quantum computers," *Phys. Rev. A* **32** (Dec. 1985), 3266–3276.

[13] T. Toffoli, "Computation and construction universality of reversible cellular automata," *Journal of Computer Systems Science* **15** (1977), 213.

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

[15] T. Toffoli, N. Margolus, *Cellular automata machines: a new environment for modeling,* MIT Press (1987).

[16] W. H. Zurek, "Reversibility and stability of information processing systems," *Phys. Rev. Lett.* **53** (1984) 391.