Feasible Models of Computation: Three-Dimensionality and Energy Consumption*

Peter Sanders, Roland Vollmar, Thomas Worsch
University of Karlsruhe, Department of Informatics,
76128 Karlsruhe, Germany

Abstract

Using cellular automata as models of parallel machines we investigate the relation between $(r-1)$- and r-dimensional machines and constraints for the energy consumption of r-dimensional machines which are motivated by fundamental physical limitations for the case $r = 3$. Depending on the operations which must be considered to dissipate energy (state changes, communication over unit-length wires, ...), some relations between the relative performance of 2-dimensional and 3-dimensional machines are derived. In the light of these results it seems imperative that for feasible models of computation energy consumption has to be considered as an additional complexity measure.

1 Introduction

Emphasizing that models of parallel computation must always be an abstraction from reality is almost a tautology, else we would not have a model but a description of a particular machine. What simplifying assumptions we make, very much depends on the pursued goal. When we want to devise algorithms which expose as much parallelism as possible, a powerful model like the CRCW-PRAM is adequate. However, practical experience shows that real computers have additional limitations worth modeling. If we assume the speed of information propagation to be effectively infinite and the cost of wires to be negligible, fixed degree interconnection networks are one adequate model [5].

However, at now feasible clock speeds approaching 1 GHz no information can travel further than 30 cm during one clock cycle. The currently dominant technology of information transmission by charging wires costs an additional order of magnitude in latencies. Since large parallel computers can have a diameter of many meters, transmission latencies are an issue. (We do not concern ourselves with telecommunication where latencies are even more crucial [21].) In addition, interconnection networks turn out to be a major cost factor for large parallel computers. These problems are particularly important for the design of a scalable architecture. Ideally, one would like to have a generic architecture where the maximum interconnect length is independent of the number of nodes and the interconnect cost never exceeds a constant fraction of the total system cost [7].

Usually the discussion of these issues is constrained to 2-D layouts[1] of computers because chips and printed circuit boards allow only a constant number of layers.

*This is technical report 2/97 of the Department of Informatics, University of Karlsruhe. It is also available at `http://liinwww.ira.uka.de/~worsch/papers/`.

[1]Usually "r-dimensional" will be abbreviated to "r-D".

The starting point for this paper is to look at the general r-dimensional case in order to identify *nontrivial* scaling issues which show up for higher dimensions with a particular emphasis on the case $r = 3$ which is important for large parallel computers.

What kind of model is adequate here? Abstract models like LogP [3] or BSP [15] can be parameterized to model the maximum communication latency but in order to be useful for designing portable parallel algorithms they do not allow to exploit local communication. The traditional way used in papers like [2, 9, 13] is to enhance the circuit model of computation by additional properties like cycles with latches, wire delays, energy consumption and space requirement. For practical questions, e.g., as studied in the PATMOS series of workshops these models get very complex. For example, in VLSI design tools, very detailed multilevel descriptions from register transfer languages down to analog simulations are used.

Conceptually much simpler than the above models is the very low level *grid model of circuits* [23] which describes the circuit as a regular orthogonal grid of cells which must consist of the same material. This model can be considered universal since inaccuracies in the production technology forbid to exploit arbitrarily accurate placement of materials. The cell model is on the other hand equivalent to cellular automata where the state of an automaton-cell encodes the states of a fixed number of VLSI-cells.[2] Note that the additional feature of synchronous operation does not increase the performance of cellular automata [16]. We will therefore use the cellular automata model for our studies and only note here without proofs that the results can also be translated into an appropriately enhanced circuit model. Our theorems overlap with curcuit model results stated in [13] and the papers cited there. However, our results for higher dimensions illuminate additional aspects like the relation between speed and energy consumption. Also, we consideer functions with a single output bit which makes it possible to use the formalisms developed in formal language theory.

The price we pay for this simple model is that we have to neglect various constant factors due to the difference in space and time requirements for wires and "compute cells" and the fact that the average packing density in the third dimension will be smaller than in the other two dimensions for currently realistic technologies. We therefore only make propositions in asymptotic notation ($\mathrm{O}(\cdot)$, $\Omega(\cdot)$, ...). Although the universe is finite, we assume to be able to scale the architectures to a sufficient size to make it meaningful to renounce constant factors. This assumption has to be made for every asymptotic analysis; otherwise, every physically feasible machine would have to be considered a finite automaton.

As a motivation of our assumptions we use fundamental physical principles and some basic properties of actual or proposed technologies which also indicate some practical relevance of the results obtained.

This paper is organized as follows: After introducing some basic definitions in Section 2 we discuss the relative performance of $(r-1)$-dimensional and r-dimensional machines in Section 3. In Section 4 it is shown that the energy consumption has to be taken into account for $r = 3$ and that this has some interesting implications for the performance of the machine. Section 5 summarizes the tradeoffs between different models. Finally, Section 6 summarizes the consequences of the technical results on a more abstract level.

[2] In a VLSI-cell there might be infinitely many levels of a physical quantity like voltage but for a digital computer they fall into a finite number of equivalence classes.

2 Basic Definitions and Examples

Although from a physical point of view we are mainly interested in 3-dimensional systems, it turns out that all results can be generalized for higher dimensionality. Therefore everything will formulated for an arbitrary dimensionality $r \geq 2$. Nevertheless we will usually only speak of "planes" and "cubes" instead of "hyperplanes" and "hypercubes".

An r-dimensional cellular automaton (CA), or $\mathbb{Z}^r-$CA for short, consists of a collection of deterministic finite automata (called *cells*) positioned at all points of the underlying lattice $R = \mathbb{Z}^r$. The cells are identical in that they use the same *set of states* Q and work according to the same rule. Let $N^{(r)} = \{(x_1, \dots, x_r) \mid \sum_i |x_i| \leq 1\} \subset \mathbb{Z}^r$ denote the so-called r-dimensional von Neumann neighborhood. If r is clear from the context or not important, we will drop the superscript. It is known to be no loss of generality to consider only this type of neighborhood. The neighbors of the cell with coordinates $\mathbf{x}$ are the cells in $\mathbf{x} + N$.[3]

A *(global) configuration* of a CA is a mapping $c : R \to Q$, i.e.[4], $c \in Q^R$. The *local configuration* in c at position $\mathbf{x}$ is the mapping $c_{\mathbf{x}} : N \to Q : \mathbf{n} \mapsto c(\mathbf{x}+\mathbf{n})$. The local transition function (or local rule) of a CA is a mapping $\delta : Q^N \to Q$. It induces the global transition function $\Delta : Q^R \to Q^R$ describing one step of the CA according to $(\Delta(c))(\mathbf{x}) := \delta(c_{\mathbf{x}})$, i.e., all cells synchronously update their state according to δ.

The type of problems which will be considered as computational tasks for CA is the recognition of formal languages. Let A denote some *input alphabet* with $|A| \geq 3$ containing a symbol @. We assume the symbols of an input $w \in A^+$ of length $n = |w|$ to be numbered from 0 to $n-1$. We want to ignore problems concerning the way how the input symbols are fed into the CA and assume that they are provided in a "row major order" to an *input cube* of cells: For $k \in \mathbb{N}_+$ denote by HC_k the cube $\{(x_1, \dots, x_r) \mid \forall i : 0 \leq x_i < k\}$. Let $m = \lceil n^{1/r} \rceil$. Then at the beginning of a computation a cell in HC_m with coordinates $(x_1, \dots, x_r)$ receives the j-th symbol of w, if $0 \leq j = \sum_{i=1}^{r} x_i m^{i-1} < n$. The initial configuration for an input w will be denoted c_w. In c_w all cells not containing an input symbol are in a so-called *quiescent state*. In the course of a computation a CA may also use cells initially in the quiescent state (quiescent cells).

Without loss of generality, cell $\mathbf{0}$ has to produce the result, i.e., the information whether the input has been accepted or rejected. This is indicated by assuming a special state f_+ (resp. f_-) in the case of acceptance (resp. rejection).

We will only consider CA which halt for every input. The *time* complexity of a CA C is the function t where $t(n)$ is the maximum number of steps needed by C to accept or reject an input of size n. The *computation cube* $HC(n)$ is the smallest cube comprising all cells used during the computation for at least one input of size n.

The *extent* of $HC(n)$ (its side length) is always denoted by $d(n)$; the *size* of $HC(n)$ is $d(n)^r$. It should be noted that extent (and size) are only partially motivated by the wish to be able to measure the amount of used cells, in other words a kind of "space complexity". (Of course there are CA for which the number of non-quiescent cells is *significantly* smaller than the size of $HC(n)$.) Another aspect of extent that is of interest in this paper is that for nontrivial problems it gives a lower bound on the computation time.

[3]We write $\mathbf{v} + M$ for $\{\mathbf{v} + \mathbf{x} \mid \mathbf{x} \in M\}$.

[4]Y^X denotes the set of all functions from X into Y.

In Subsection 4.2 we will introduce a further measure called *(state) change complexity.*

Throughout the paper we will consider (variants of) two formal languages :

$$\begin{aligned} L_{\text{parity}} &:= \{w \mid w \in \{\mathtt{0},\mathtt{1}\}^+ \text{ and } w \text{ contains an even number of } \mathtt{1}\text{s}\} \\ L_{\text{vv}} &:= \{v\mathtt{@}^{|v|}v \mid v \in \{\mathtt{0},\mathtt{1}\}^+\} \end{aligned}$$

For a language L let $L^{[r]} := \{w \in L \mid |w|^{1/r}/3 \text{ is an integer}\}$. Words in $L^{[r]}$ have the nice property that they completely fill their input cubes (because $|w|^{1/r}$ is an integer). In the case of words $w \in L_{\text{vv}}^{[r]}$ it is even the case that each of the three parts of w completely fills a hyperrectangle which is one third of the input cube.

For the description of algorithms for these languages as well as for later constructions and proofs it is useful to define the following notation. Given some cube $W \subseteq R$ and some $1 \leq k \leq r$ and i let $HP_k(i)$ denote the plane of W consisting of all cells having coordinates with a k-th component $x_k = i$. When considering a CA for inputs of size n it is always to be understood, that planes refer to $HC(n)$.

L_{parity} as well as L_{vv} can be accepted using only the cells in the input cube in time $\Theta(n^{1/r})$ which is asymptotically optimal. In order to keep the descriptions of the algorithms clear we will only consider the cases of the "nice" languages $L^{[r]}$.

2.1 Algorithm. (for L_{parity}) As always let $m = m(n) = \lceil n^{1/r} \rceil$. In the first phase all the cells in $HP_r(m-1)$ send signals to their corresponding cells in $HP_r(0)$ computing the sum modulo 2 of all symbols in one row along the last dimension. The resulting values are then added according to the algorithm for the $r-1$ dimensional case in the side face $HP_r(0)$ of the input cube.

Note that during the first $n^{1/r}$ steps the signals amount to a kind of *activity front* moving through the cube. Hence, in each step the cells of only one plane are "active" (this notion will be made precise in Section 4).

2.2 Algorithm. (for L_{vv}) The recognition of $L_{\text{vv}}^{[r]}$ on $\mathbb{Z}^r$−CA is a little bit more complicated. The cells in $HP_r(0)$ send an activity front with speed 1/2, which initiates each plane $HP_r(i)$, $0 \leq i < m/3$ to send its information to the last third of the input with speed 1.[5] The first plane in the second v-part (which is able to identify itself because of its @-neighbors) waits for the first signals to arrive. As soon as this happens an activity front is started with speed 1/2 moving through the second v-part. Each time it arrives at a new plane its cells compare their own symbol to the one arriving from the first v part, which is killed (i.e. it does not move further). If one of the comparisons fails a rejection signal is sent to cell **0** with speed 1. If no such signals arrive during the first $(\frac{4}{3} + r)m$ steps the input is accepted. (This time can be measured easily by a signal running along the edges of the cube.)

If this algorithm is implemented in a standard CA, the sending of the symbols from the first v-part to the second one is achieved using activity fronts. In this case the $m/6$ activity fronts which have been started first simultaneously move through $m/3$ cells before beginning their comparisons. Hence in each of $\Theta(n^{1/r})$ steps there are up to $\Theta(n)$ "activities" per step, and therefore the algorithm may involve $\Theta(n^{(r+1)/r})$ state changes overall.

[5] In a standard CA this is done using further activity fronts which carry with them the symbols of the plane where they started. But note that this algorithm will also be adopted to a generalized model in Subsection 4.3.

Note that in the above algorithm we have only described what happens in the case of an input from $L_{\mathrm{vv}}^{[r]}$. We leave the case of an input *not* belonging to the language as an exercise.

3 $r-1$ Versus r Dimensions

The methods in this section are easier to explain using terms like embedding, simulation and layout of networks [11]. The translation of the results into CA is possible. But since there are further technical complications this will be done in another paper. An additional reason for an informal presentation is that we are convinced that the facts presented here are known or easy to reproduce by experienced people working on network embeddings. Since the results are needed in Section 5 and because we think that the techniques deserve wider publicity, we did not want to omit descritpions of the basic ideas however.

Note (added during final proof reading): The papers by Rosenberg and Leighton [12, 18, 19] contain results which are very closely related to those reported in Section 3.2.

3.1 Mutual Simulation of $r-1$ and r

Interestingly, it is not trivial to simulate a cubic $(r-1)$-D mesh on a cubic r-dimensional mesh of equal size (for $r>2$) if one is only interested in the case of one-to-one assignments of simulated processors to simulating processors and a sufficiently "nice" data flow during the simulation (i.e. small congestion and dilation). In particular, there is no embedding doing the job. However, by decoupling the computations of distant parts of the machine a simulation with constant slowdown can be achieved. (The case $r=3$ is described in [1] and it can be generalized to arbitrary dimension.)

On the other hand, there are many problems for which $\mathbb{Z}^r$−CA are faster than $\mathbb{Z}^{r-1}$−CA. For instance, for L_{parity} the speed ratio is in $\Theta\big(n^{1/(r(r-1))}\big)$. This result is tight in the sense that there are no languages for which the speed gap is even larger: An n-node cubic r-D mesh can be mapped one-to-one into an n-node cubic $(r-1)$-D mesh such that nodes which are immediate neighbors in the r-D mesh are at most $\Theta\big(n^{1/(r(r-1))}\big)$ hops apart in the $(r-1)$-D mesh. We only state the scheme for the case that n is a $(r-1)r$-th power: Cut the cube into $n^{1/r}$ slices. Scale each slice to size $n^{1/(r-1)} \times n^{1/(r-1)}$ such that cube nodes are $n^{1/(r(r-1))}$ nodes apart. Now interleave the $n^{1/r}$ stretched slices such that nodes adjacent along the r-th direction are also neighbors in the embedding, e.g. in a snake-like fashion. The resulting embedding has dilation $n^{1/(r(r-1))}$ and congestion $n^{1/(r(r-1))}+1$. Figure 1 depicts the embedding for $r=3, n=64$.

3.2 Layout Complexity of Butterfly Networks

An argument often brought forth against the feasibility of logarithmic diameter interconnection networks is their large VLSI layout complexity. For example, in [23, Theorem 6.2] it is shown that the 2-D layout of an $n \log n$-node butterfly requires area $\Theta\big(n^2\big)$. This does not only imply a high cost but also a maximum distance between nodes of $\Theta(n)$. So it would seem that such networks asymptotically have a *larger* latency than 2-D meshes $(n^{1/2})$.

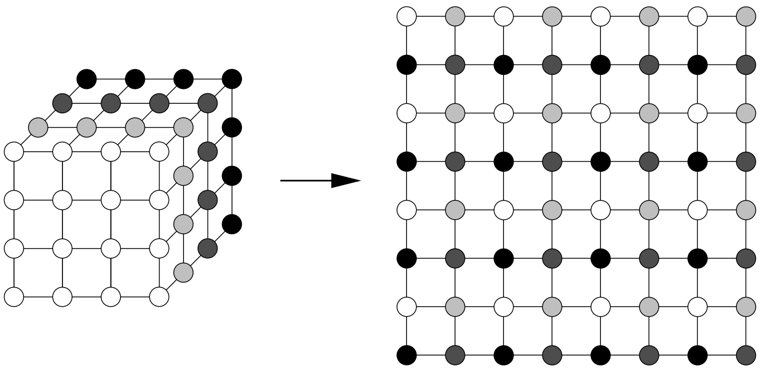

Figure 1: Embedding of a 4^3 cube into an 8^2 mesh.

The situation gets a different touch for general r. An $n \log n$ node butterfly can be assembled within an r-cube of diameter $\mathrm{O}\bigl(n^{1/(r-1)}\bigr)$ and volume $\mathrm{O}\bigl(n^{r/(r-1)}\bigr)$ with no wire longer than $\mathrm{O}\bigl(n^{1/(r-1)}\bigr)$: Over one layer with n processors, $\log n$ layers of connections are placed. Each of these layers consists of a certain number of cell planes. Switches and processors are connected in the r-th direction. The exchange connections alternate between the $r-1$ remaining directions. So the length of the wires only doubles every $r-1$ layers. It is also sufficient to double the height of the layers every $r-1$ layers in order to keep the wires at a constant distance. Figure 2 gives an example for $r = 3$, $n = 16$. We can think of this arrangement as a way to extend a $(r-1)$-D architecture by r-dimensional wiring in order to achieve a higher bisection bandwidth ($\Theta(n)$ versus $\Theta\bigl(n^{1-1/r}\bigr)$) (and possbily a lower latency by a constant factor).

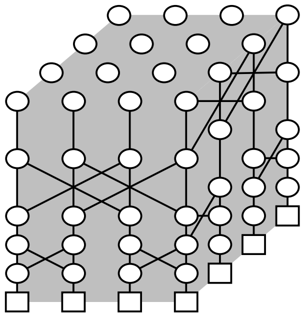

Figure 2: 3-D layout of a multistage butterfly with 2^4 processors and 4 layers of connections.

4 Energy Consumption

4.1 Is There a Problem?

It is well known that cooling and power supply of chips and entire computers is a crucial issue in hardware design. However, if the power consuming elements are arranged in two dimensions, cooling poses no limit to building larger machines since the surface of the machine grows in proportion with the number of active elements.

Power distribution faces the practical problem that for building larger chips the number of power supply pins needs to be scaled in proportion with the chip area. This is not possible if we insist on using only pads at the boundary of the chip. We get an analogous but more fundamental problem with cooling (and power supply) for 3-D machines since there is no additional space dimension left we can exploit.

A 3-D machine with extent d can have $\Theta(d^3)$ active elements but all the dissipated energy has to be transported through the surface of some cube with surface area $\mathrm{O}(d^2)$. (This holds regardless of the actual surface of the machine which could even be a fractal with an area of up to $\Theta(d^3)$.) On the other hand, for any given technology, the maximum allowable temperature at any point in the machine must not exceed a certain constant value, i.e., this limit cannot be scaled with the machine size. Therefore, it is not feasible for each cell of a large machine to consume one unit of energy in each step because eventually, the machine will become overheated. More precisely, we can state the following necessary condition for a physically feasible computation:

4.1 Proposition. (for the 3-dimensional case) *For every physically feasible computation, every subcube of the machine with extent d' and every interval of the computation of length t', no more than $\mathrm{O}\left({d'}^3 + t'{d'}^2\right)$ units of energy may be dissipated within this space-time interval of the computation.*

For the general case of r-D systems we restate this proposition as follows.

4.1 Proposition. (for the r-dimensional case) *For every physically feasible computation, every subcube of the machine with extent d' and every interval of the computation of length t', no more than $\mathrm{O}\left({d'}^r + t'{d'}^{r-1}\right)$ units of energy may be dissipated within this space-time interval of the computation.*

This generalization for example makes sense for the pin limitation problems for 2-D chips mentioned above. In this case we would have $r = 2$. The interest in the case $r \geq 4$ is probably a more theoretical one.

The crucial question now is: Which operations should be considered to consume energy? It follows from the laws of thermodynamics, that every *irreversible* computation (e.g. `and`, `or`) must consume at least $k_B T \ln 2$ of energy where k_B is the Boltzmann constant and T the temperature of the switching element in Kelvin. But this cannot imply a nontrivial lower bound on the energy consumption for the solution of any problem, since in principle universal computers can be built using only reversible gates. However, gates which actually consume less energy than the $k_B T \ln 2$ bound are currently only gedanken experiments and/or trade speed for energy consumption; so they cannot be used for building *fast* computers. Nevertheless it is interesting to note that reversibility has proved important (shown to be useful?) for the design of low power MOS circuits [17, 22] (although these circuits still consume much more energy than $k_B T \ln 2$ so that reversibility is no physical necessity for reducing energy consumption). In addition, some reversible computations generate a lot of "garbage" information [6] which either has to be stored somewhere or moved out of the machine. For a more detailed discussion refer to [4, 10, 14].

4.2 The Relation to Change Complexity

A simple approximation to the actual energy consumption of a machine is to count the (proper) state changes of its cells. Especially for CMOS this is quite accurate. Proposition 4.1 can therefore be reformulated in terms of the numbers of state changes.

For CA this corresponds to the concept of *(state) change complexity* which has been introduced in [24]: The change complexity of a CA C with time complexity t is the function s where

$$s(n) = \max_{w \in A^n} \left| \{ (\mathbf{x}, \tau) \mid \tau < t(n) \text{ and } (\Delta^\tau(c_w))(\mathbf{x}) \neq (\Delta^{\tau+1}(c_w))(\mathbf{x}) \} \right| .$$

A CA satisfying the constraints of Proposition 4.1 is called a $\mathbb{Z}^r-$CACE (where CE stands for "cold everywhere").

Obviously the relation $s(n) \geq t(n)$ always holds: In each step of a computation at least one cell has to change its state, because otherwise the reached configuration will never change again. On the other hand we also have:

4.2 Corollary. *A $\mathbb{Z}^r-$CA which fulfills Proposition 4.1 must have a time complexity of at least $t(n) \in \Omega\left(\frac{s'(n)-d'(n)^r}{d'(n)^{r-1}}\right)$ if $s'(n)$ changes occur in a cube of extent $d'(n)$.*

One of the basic tools for the results in Section 5 is Proposition 4.4 below that there are languages the recognition of which requires a certain nontrivial amount of state changes. As prerequisites some more formalism and a lemma are needed, which is a generalization of [24, Lemma 2].

We use the notation $\langle w \rangle$ for the "space time diagram" of an input w, i.e. the mapping $R \times \mathbb{N}_+ \to Q$ where $(\Delta^\tau(c_w))(\mathbf{x})$ gives the state of cell $\mathbf{x}$ at time τ. The restriction of $\langle w \rangle$ to a subset $M \subseteq R$ is denoted[6] $\langle w|M \rangle : M \times \mathbb{N}_+ \to Q$.

Let $R = M_1 \dot{\cup} M_2$ be a partition of the whole lattice of a CA in two disjoint subsets and let K denote the *border* "between" M_1 and M_2, i.e.

$$K = \{ \mathbf{x} \in M_1 \mid \exists \mathbf{n} \in N : \mathbf{x} + \mathbf{n} \in M_2 \} \cup \{ \mathbf{x} \in M_2 \mid \exists \mathbf{n} \in N : \mathbf{x} + \mathbf{n} \in M_1 \} .$$

The $\langle w|K \rangle$ will be called crossing sequences [8] because they will be used in the same way as Hennie's concept for Turing machines.

4.3 Lemma. *Let $R = M_1 \dot{\cup} M_2$ be a partition with border K and let w_1, w_2 be two inputs of equal length such that $\langle w_1|K \rangle = \langle w_2|K \rangle$. Then $\langle w_1|M_1 \rangle \cup \langle w_2|M_2 \rangle$ is the space time diagram for an input w. If $\mathbf{0} \in M_i$, w is accepted if and only if w_i is accepted.*

Here the notation $\langle w_1|M_1 \rangle \cup \langle w_2|M_2 \rangle$ is to be understood as the mapping $X : (M_1 \cup M_2) \times \mathbb{N}_+ \to Q$, where $X(\mathbf{x}, \tau) = \langle w_i|M_i \rangle(\mathbf{x}, \tau)$ for $\mathbf{x} \in M_i$.

Proof. Let $K_1 = M_1 \cap K$ and $K_2 = M_2 \cap K$. As always $n = |w_1| = |w_2|$. Let $X = \langle w_1|M_1 \rangle \cup \langle w_2|M_2 \rangle$.

Let w be the input corresponding to the input cube consisting of the M_1-part of the input cube for w_1 and the M_2-part of the input cube for w_2.

By induction on τ we prove: $\forall \tau \geq 0 : \forall \mathbf{x} : \langle w \rangle(\mathbf{x}, \tau) = X(\mathbf{x}, \tau)$.

- $\tau = 0$: by the construction of w.

[6] This notation should not be confused with the bra-ket notation from quantum mechanics.

- $\tau \to \tau+1$: Assume that $\mathbf{x} \in M_1$ (the case $\mathbf{x} \in M_2$ can be treated analogously). Then for all τ we have $X(\mathbf{x},\tau) = \langle w_1|M_1\rangle(\mathbf{x},\tau)$ and hence it suffices to prove $\langle w\rangle(\mathbf{x},\tau+1) = \langle w_1|M_1\rangle(\mathbf{x},\tau+1)$.

$$\begin{array}{ll}
\langle w\rangle(\mathbf{x},\tau+1) & \\
= (\Delta^{\tau+1}(c_w))(\mathbf{x}) & \text{definition of } \langle w\rangle \\
= \delta((\Delta^{\tau}(c_w))_{\mathbf{x}}) & \text{definition of } \Delta \\
= \delta(\mathbf{n} \mapsto (\Delta^{\tau}(c_w))(\mathbf{x}+\mathbf{n})) & \text{definition of } c_{\mathbf{x}} \\
= \delta(\mathbf{n} \mapsto X(\mathbf{x}+\mathbf{n},\tau)) & \text{induction hypothesis} \\
= \delta(\mathbf{n} \mapsto (\langle w_1|M_1\rangle \cup \langle w_2|M_2\rangle)(\mathbf{x}+\mathbf{n},\tau)) & \text{definition of } X \\
= \delta(\mathbf{n} \mapsto (\langle w_1|M_1\rangle \cup \langle w_2|M_2 \cap K\rangle)(\mathbf{x}+\mathbf{n},\tau)) & \text{because } \mathbf{x} \in M_1 \\
= \delta(\mathbf{n} \mapsto (\langle w_1|M_1\rangle \cup \langle w_1|M_2 \cap K\rangle)(\mathbf{x}+\mathbf{n},\tau)) & \text{because } \langle w_1|K\rangle = \langle w_2|K\rangle \\
= \delta(\mathbf{n} \mapsto \langle w_1\rangle(\mathbf{x}+\mathbf{n},\tau)) & \\
= \langle w_1\rangle(\mathbf{x},\tau+1) & \\
= \langle w_1|M_1\rangle(\mathbf{x},\tau+1) & \text{because } \mathbf{x} \in M_1
\end{array}$$

■

In the sequel we will need nondecreasing functions $f(n) \notin \mathrm{O}(\log n)$, i.e., satisfying $\lim_{n\to\infty} \frac{\log n}{f(n)} = 0$. The interesting case will be when $f(n)$ grows slowly; therefore we restrict ourselves to functions which are bounded by a polynomial of $\log n$. We will call such functions *almost-log.*

4.4 Proposition. *A $\mathbb{Z}^r$−CA recognizing L_{vv} makes a total of $\Omega\left(\frac{n^{(r+1)/r}}{f(n)}\right)$ state changes for every almost-log $f(n)$ in the subcube of extent $3n^{1/r}$ containing the input cube in its center. The same is true for $L_{\mathrm{vv}}^{[r]}$.*

Proof. Assume that C is a CA with $|Q|$ states accepting L_{vv} in polynomial time $t(n)$ (otherwise the change complexity is trivially hyperpolynomial). W.l.o.g. consider an input of size $n = m^r$ for some integer m. For $1 \le i \le \frac{m}{6}$ consider the partitions $M_{i1} \dot\cup M_{i2}$ and the corresponding borders K_i, such that $M_{i1} = \{(x_1,\ldots,x_{r-1},x_r) \mid -2i \le x_1 < m+2i, \ldots, -2i \le x_{r-1} < m+2i \text{ and } -2i \le x_r < \frac{m}{3}+2i\}$.

Obviously all M_{i1} encompass all symbols of the first third of an input but none of the last third. All the K_i are nonintersecting and their sizes can be bounded by $b_1 n^{(r-1)/r}$ for some common constant b_1.

For any i, $1 \le i \le \frac{m}{6}$, the total number of crossing sequences containing at most $g(n) := \frac{n}{f(n)}$ state changes in K_i is less than

$$\begin{aligned}
\binom{b_1 t(n) n^{(r-1)/r}}{g(n)} |Q|^{b_1 n^{(r-1)/r} + g(n)} &\le \left(b_1 t(n) n^{(r-1)/r}\right)^{g(n)} \left(|Q|^{1+b_1 f(n)/n^{1/r}}\right)^{g(n)} \\
&\le \left(2^{\log(b_1 t(n) n^{(r-1)/r})} |Q|^{b_2}\right)^{g(n)} \le 2^{b_3 g(n) \log n}
\end{aligned}$$

for sufficiently large n and some constant b_3. This follows from the observation that any such crossing sequence (and only these) can be constructed by independently choosing the states for the $b_1 n^{(r-1)/r}$ cells of K_i at time $\tau = 0$, $g(n)$ points in the space time diagram where a cell might change its state, and the corresponding new states.

Because of Lemma 4.3, the definition of L_{vv} and the shape and position of the borders K_i there must not be two words $w_1 \neq w_2$ in L_{vv} and an index i such that $\langle w_1|K_i\rangle$ and $\langle w_2|K_i\rangle$ are identical. Hence for each of the $m/6$ values of i there are at most $2^{b_3 g(n)\log n}$ words for which we can have at most $g(n)$ changes in K_i. For sufficiently large n we have

$$(m/6)2^{b_3 g(n)\log n} = 2^{b_3 g(n)\log n + \log(m/6)} = 2^{b_3 n(\log n)/f(n) + \log(n^{1/r}/6)} < 2^{n/3}$$

because $\lim_{n\to\infty} \frac{\log n}{f(n)} = 0$. Since the latter is the number of words of length n in L_{vv}, there must be at least one word w of length n which causes more than $g(n) = n/f(n)$ state changes in *each* of the $\frac{m}{6}$ crossing sequences K_i. Hence a total number of state changes for this w exceeds $\frac{mn}{6f(n)} \in \Omega\left(\frac{n^{(r+1)/r}}{f(n)}\right)$. ■

4.5 Corollary.

1. *L_{vv} cannot be accepted by any $\mathbb{Z}^r$−CACE in less than $n^{2/r}/f(n)$ time (where $f(n)$ is an arbitrary almost-log function).*

2. *L_{vv} can be accepted by a $\mathbb{Z}^r$−CACE in time $\Theta(n^{2/r})$.*

Proof.

1. This follows from Proposition 4.4 and Corollary 4.2.

2. This follows from Algorithm 2.2 and Proposition 5.5 below.

■

It should be noted that for the proof of Proposition 4.4 we did not restrict ourselves to CA which use only the input cube for their computations. The result holds for *all* CA. It can therefore also be exploited to give an example where any CACE has to be slower than a general CA due to a large change complexity only in a proper subcube of the input cube although the overall change-complexity is small. Let L_r denote the language of all words with the following r-dimensional arrangement: The central subcube of extent $n^{1/(r+1)}$ (and size $n^{r/(r+1)}$) contains a word from $L_{\mathrm{vv}}^{[r]}$ and the remainder of the input cube is filled with @ symbols everywhere.

4.6 Corollary. *Let f be an almost-log function.*

1. *A $\mathbb{Z}^r$−CA C recognizing L_r will make a total of $\Omega(n/f(n))$ state changes in the central subcube of the input cube of extent $3n^{1/(r+1)}$.*

2. *A $\mathbb{Z}^r$−CACE needs at least $\Omega(n^{2/(r+1)}/f(n))$ steps to accept L_r.*

Proof.

1. For the proof of the first part one has to observe that an algorithm for the recognition of L_r can be turned into an algorithm for the recognition of $L_{\mathrm{vv}}^{[r]}$ very easily: one only has to produce the output in another cell. Hence a low state change complexity for L_r would also imply a low state change complexity for $L_{\mathrm{vv}}^{[r]}$ contradicting Proposition 4.4.

2. The second part immediately follows from the first because of Corollary 4.2.

■

4.3 The Role of Communication

We have seen that technologies which limit the change complexity, considerably constrain the performance of 3-D machines. It is therefore important to look for relaxations. One candidate is communication. In terms of CA, handing information from one cell to another must involve state changes. Therefore communicating one bit of information through a simulated "wire" requires energy proportional to the length of the wire. Although this is really an issue for current CMOS technology (e.g. [20]), there are technologies which do have negligible energy consumption per unit of wire length. For example, modern optical fibers are very translucent for many kilometers.

Therefore we introduce a modification of the CACE model, namely with wires, denoted by $\mathbb{Z}^r$−CAWW: Each cell has access to unidirectional "wire"-registers for each of the $2 \cdot r$ coordinate directions. The information in these registers moves to the corresponding neighboring cell in every step without consuming energy. However, reading or writing a wire register requires one unit of energy. Furthermore it is required that the same energy constraints as for CACE must be satisfied.

4.7 Algorithm. Algorithm 2.2 can be implemented on a CAWW in such a way that the information about the first v-part is transmitted through the wires. In this case there are never more than two active planes and therefore Proposition 4.1 is fulfilled without any slowdown of the algorithms.

5 Comparison of the CA Models

In this section we will compare the different types of models introduced above.

Let $\mathbb{Z}^r$−CA−EXT−TIME(d,t) denote the families of languages recognized by $\mathbb{Z}^r$−CA with extent at most $\mathrm{O}(d)$ and time complexity at most $\mathrm{O}(t)$; for $\mathbb{Z}^r$−CACE a similar notations will be used.

We begin with the results concerning the simulations of higher-dimensional CA on lower-dimensional CA and vice versa. Observe that in both cases the size of the computation cube remains invariant.

5.1 Proposition. (Change of dimensionality)
If d is space-constructible in time t [25]:

1. $\mathbb{Z}^r$−CA−EXT−TIME$(d,t) \subseteq \mathbb{Z}^{r-1}$−CA−EXT−TIME$(d^{r/(r-1)}, d^{1/(r(r-1))}t)$

2. $\mathbb{Z}^{r-1}$−CA−EXT−TIME$(d,t) \subseteq \mathbb{Z}^r$−CA−EXT−TIME$(d^{(r-1)/r}, t)$

Proof.

1. The basic idea for the proof of this result has been sketched in Section 3. It should be noted that one has to solve the additional problem of rearranging the input. Refer to figure 1: A 2-D CA gets its input in the form "2 rows of white elements, 2 rows of light gray elements, 2 rows of dark gray elements, and 2 rows of black elements". Before the simulation can be started the elements have to be moved around to get the distribution as indicated on right hand side of figure 1. This can indeed be done on a CA in a *sufficiently small amount of time.*

2. This has been proved in [1] (for the case $r = 3$).

Obviously each $\mathbb{Z}^r-$CACE is a $\mathbb{Z}^r-$CA. Hence, for no language a $\mathbb{Z}^r-$CA has to be slower than a $\mathbb{Z}^r-$CACE. Furthermore for no language a $\mathbb{Z}^r-$CA has to be slower than a $\mathbb{Z}^r-$CAWW. The relations between models of different dimensionality, e.g., between $\mathbb{Z}^{r-1}-$CA and $\mathbb{Z}^r-$CACE are less obvious.

5.2 Proposition. (Restricted vs. unrestricted CA)
Let f be an almost-log function.

1. *There are problems for which $\mathbb{Z}^r-$CA are faster than every $\mathbb{Z}^r-$CACE by a factor of $\Omega\left(\frac{n^{1/r}}{f(n)}\right)$.*

2. *The same result holds for $\mathbb{Z}^r-$CAWW instead of $\mathbb{Z}^r-$CA.*

Proof.

1. According to Corollary 4.5 any CACE recognizing L_{vv} needs at least time $\Omega\big(n^{2/r}/f(n)\big)$. On the other hand L_{vv} can be recognized by an unrestricted $\mathbb{Z}^r-$CA in time $\Theta\big(n^{1/r}\big)$ (Algorithm 2.2).

2. This follows analogously from the fact that L_{vv} can be recognized by $\mathbb{Z}^r-$CAWW in time $\Theta\big(n^{1/r}\big)$ (Algorithm 4.7).

∎

As can be seen from the above results the high state change complexity in our examples stems "only" from the need for the "movement" of a lot of data over a long distance but not from the need for a lot of "computations". It is an open problem, whether there are also languages for which $\mathbb{Z}^r-$CA can be significantly faster than any $\mathbb{Z}^r-$CAWW (for which per definitionem energy constraints hold).

5.3 Proposition. (Restrictions versus dimensionality) *Let $r \geq 3$.*

1. *There are problems for which $\mathbb{Z}^r-$CACE are faster than $\mathbb{Z}^{r-1}-$CA by a factor of at least $\Theta\big(n^{1/(r(r-1))}\big)$. The factor cannot be larger for any problem as long as the computation cube coincides with the input cube.*

2. *There are problems for which $\mathbb{Z}^{r-1}-$CA are faster than $\mathbb{Z}^r-$CACE by a factor of at least $\Theta\big(n^{(r-2)/(r(r-1))}/f(n)\big)$.*

Proof.

1. L_{parity} can be recognized by $\mathbb{Z}^r-\mathrm{CA{\small CE}}$ in time $\Theta(n^{1/r})$ using Algorithm 2.1 (and analogously for dimensionality $r-1$). Of course this is also a lower bound. The tightness follows from Proposition 5.1.1.

2. L_{vv} can be recognized by $\mathbb{Z}^{r-1}-\mathrm{CA}$ in time $\Theta(n^{1/(r-1)})$ (Algorithm 2.2). On the other hand every $\mathbb{Z}^r-\mathrm{CA{\small CE}}$ needs at least time $\Omega(n^{2/r}/f(n))$ (Corollary 4.5).

■

Considering $\mathbb{Z}^2-\mathrm{CA}$ and $\mathbb{Z}^3-\mathrm{CA{\small CE}}$ to be feasible, but not $\mathbb{Z}^3-\mathrm{CA}$, the above results imply that in some cases, e.g., for L_{vv}, the time saved by going from (feasible) 2-dimensional CA to (feasible) 3-dimensional CA gets more than lost because of the restriction on the number of state changes. This is not only the case for L_{vv}. In general it follows immediately from Proposition 5.1 above:

5.4 Corollary. *If a language L can be recognized by a $\mathbb{Z}^r-\mathrm{CA}$ with extent $d(n)$ faster than by $\mathbb{Z}^r-\mathrm{CA{\small CE}}$ with extent $d(n)$ by a factor of $q(n) \notin \mathrm{O}(d(n)^{1/r})$, then L can be recognized faster by $\mathbb{Z}^{r-1}-\mathrm{CA}$ with extent $d(n)^{r/(r-1)}$ than by $\mathbb{Z}^r-\mathrm{CA{\small CE}}$ with extent $d(n)$.*

5.5 Proposition. (Reduction of state changes)

$$\mathbb{Z}^r-\mathrm{CA}-\mathrm{E{\small XT}}-\mathrm{T{\small IME}}(d,t) \subseteq \mathbb{Z}^r-\mathrm{CA{\small CE}}-\mathrm{E{\small XT}}-\mathrm{T{\small IME}}(d, d\cdot t)\,.$$

Proof. Let C be an arbitrary $\mathbb{Z}^r-\mathrm{CA}$. A $\mathbb{Z}^r-\mathrm{CA{\small CE}}$ E recognizing the same formal language works as follows: Each step of C (during which all cells may possibly change their states) is simulated in d steps of E. On the side face $HP_1(0)$ an activity front is started moving once through the cube and updating the states of the cells $HP_1(i)$ in the i-th step. ■

The example of L_{vv} shows, that choosing $d = n^{1/r}$ in Proposition 5.5 leads to an asymptotically almost optimal result. The proposition assures that recognition on $\mathrm{CA{\small CE}}$ is possible in time $\mathrm{O}(n^{2/r})$ while Corollary 4.5 provides a lower bound of $n^{2/r}/f(n)$ for every almost-log $f(n)$.

6 Conclusions and Future Work

This paper helps to understand some of the present and future problems of parallel machine design using cellular automata as a simple but accurate model. In particular, it shows that the third space-dimension has to be taken into account. In a sense, even machines traditionally thought as two-dimensional require the third dimension for cooling and power supply; so, why not exploit the third dimension for additional purposes in order to increase performance. However, cooling considerations show that the energy consumption must not be increased by more than a constant factor.

There are technologies which meet this constraint for memory and communication channels. If we are considering a moderately coarse grained machine which has

n processors with $\Omega(n^{1/2})$ memory cells for each processor, even supplying a full multistage-butterfly network (or hypercube, or ...) does not disproportionately increase the cost of the machine. (Currently, sophisticated networks are very expensive for *economic* reasons because they are not mass-produced like processors or memory.)

Exploiting the third dimension also for computations can be faster than any 2-D machine. However, there are also problems with such a high energy consumption that a flat two-dimensional arrangement is superior if communication consumes energy proportional to wire length. For classical nonreversible computing, the state change complexity of CA elegantly models the energy consumption and mirrors the amount of information transmission (and can therefore give hints for the division of computation processes).

Technological considerations suggest a number of remaining questions. For example, we believe that a more restricted variant of wires allowing only access to the endpoints, would only incur a logarithmic overhead compared to the current model. The scaling properties of free space optical interconnects might also be interesting. Other restrictions would treat memory cells differently than compute cells. Also, the energy complexity of memory access in the presence of memory hierarchies is worth looking at. The condition formulated in Proposition 4.1 is only a necessary condition. But it is not entirely trivial to actually devise a scalable cooling technology. For example, if it should turn out to be feasible to build 3-D nanoscale cellular automata [22] these will probably have to be cooled by heat diffusion which is very inefficient. In this case, Proposition 4.1 needs to be changed to $\mathrm{O}(d'^3 + t'd')$ for computations deep inside any subcube (because the time a unit of energy needs to reach the surface by a "random walk" grows quadratically with the diameter).

Acknowledgments

The authors gratefully acknowledge interesting and helpful discussions with Sebastian Egner and Jozef Gruska.

References

[1] A.-C. Achilles, M. Kutrib, and Th. Worsch. On relations between arrays of processing elements of different dimensionality. In R. Vollmar, W. Erhard, and V. Jossifov, editors, *Parcella '96*, pages 13–20. Akademie Verlag, 1996.

[2] A. Aggarwal, A. K. Chandra, and P. Raghavan. Energy consumption in VLSI circuits (preliminary version). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 205–216, Chicago, Illinois, 1988. ACM Press.

[3] D. Culler, R. Karp, et al. LogP: Towards a realistic model of parallel computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, 1993.

[4] R. P. Feynman. *Feynman Lectures on Computation.* Addison Wesley, 1996.

[5] L. M. Goldschlager. A universal interconnection pattern for parallel computers. *Journal of the ACM*, 29(4):1073–1086, 1982.

[6] J. S. Hall. A reversible instruction set architecture and algorithms. In *Physics and Computation*, pages 128–134, 1994.

[7] A. C. Hartmann and J. D. Ullman. Model categories for theories of parallel systems. In H. J. Lipovski and M. Malek, editors, *Parallel Computing – Theory and Comparisons*, pages 369–381. Wiley, 1987.

[8] F. C. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, 8:553–578, 1965.

[9] G. Kissin. Upper and lower bounds on switching energy in VLSI. *Journal of the ACM*, 38(1):222–254, 1991.

[10] R. Landauer. Zig-zag path to understanding. In *Physics and Computation*, pages 54–59, 1994.

[11] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publ., San Mateo, CA 94403, 1992.

[12] F. T. Leighton and A. L. Rosenberg. Three-dimensional circuit layouts. *SIAM Journal on Computing*, 15(3):793–813, 1986.

[13] T. Lengauer. VLSI theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 16, pages 835–868. Elsevier Science Publishers and MIT Press, 1990.

[14] M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1993.

[15] W. F. McColl. Scalable computing. In *Computer Science Today*, number 1000 in LNCS, pages 46–61. Springer, 1996.

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

[17] P. Patra and D. S. Fussell. On efficient adiabatic design of MOS circuits. In *Physics and Computation*, pages 260–269, 1996.

[18] A. L. Rosenberg. *Three-Dimensional Integrated Circuitry*, pages 69–80. Computer Science Press, Inc., 1981.

[19] A. L. Rosenberg. Three-dimensional VLSI: A case study. *Journal of the ACM*, 30(3):397–416, 1983.

[20] J. Smit and J. A. Huisken. On the energy complexity of the FFT. In *PATMOS Power and Timing Modeling for Performance of Integrated Circuits*, pages 119–132, 1995.

[21] A. S. Tanenbaum. *Computer Networks.* Prentice Hall, 3 edition, 1996.

[22] T. Toffoli. Power management alternatives for nanoscale cellular automata. In *Physics and Computation*, pages 303–307, 1996.

[23] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[24] R. Vollmar. Some remarks about the "efficiency" of polyautomata. *International Journal of Theoretical Physics*, 21:1007–1015, 1982.

[25] K. Wagner and G. Wechsung. *Computational Complexity.* D. Reidel, Dordrecht, 1986.