Power of Fast VLSI Models is Insensitive to Wires' Thinness.

Gene Itkis, *Leonid A. Levin*[*]
Department of Computer Science
Boston University
111 Cummington St, Boston, MA 02215
(e-mail to: Lnd@cs.bu.edu or Gene@cs.bu.edu)

Abstract

VLSI f-models allow the switching time to decrease to $f(D)$ when the length of all wires is restricted by D.[1] We call them *fast* if the decrease is slightly superlinear, i.e. $\sum_k f(1)/f(k)$ is $O(1)$. The fast models are so strong and robust that their computational power cannot be increased by any combination of the following: (1) making zero the width of each wire of length d, except for its $\log d$ segment, thus eliminating layout and area considerations; (2) allowing wires to transmit $\log d$ bits simultaneously; (3) making the switching time $f(d)$ of each node depend only on the length d of its own input wires, thus enabling small subcircuits to run faster; (4) changing f while preserving $\sum_k 1/f(k)$; (5) enabling the nodes to change connections arbitrarily in the runtime. We construct a kind of operating system *Link Server* or *Linx*, for short, which simulates all these powers on-line. The condition $\sum_k f(1)/f(k) = O(1)$ cannot be weakened.

[*]The second author is also affiliated with MIT. Supported by NSF grant DCR-8607492.Appeared in FOCS-1989

[1]Various f-models were studied in [Seitz 79], [Thompson 80], [Mead Conway 80], [Chazelle Monier 81a,b], [Bilardi Pracchi Preparata 81], [Ramachandran 82], [Nath Maheshwari Bhatt 83], [Aggarwal 83,85] [Vitanyi 85] and other papers. Among possible $f(d)$, they consider C, Cd^t, $C(\log d)^t$ where constant C depends only on the chip's diameter and t is fixed. [Seitz 79], [Thompson 80], [Paterson Ruzzo Snyder 81], [Aggarwal 83,85], and others allowed the switching time of each node depend on the length of only its own input wires; [Seitz 79] has shown that such (self-timed regioned) chips can be feasible, practical, and reliable. [Vitanyi 85] notes that sublinear f require increase in either C or wire width damaging performance and the (vanishing exponentially) probability of defect-free fabrication. A simpler case of our result, restricted to two-way wires was given in [Levin 82].

1 The Result

1.1 Virtual Chips

Wires' area is a major issue in VLSI. Most n-node graphs of constant degree cannot be mapped into an Euclidean ball with $o(n)$ measure of border, if edges be disjoint curves of constant width.[2] So, one dimension is lost to wires, which makes generic VLSI circuits effectively one-dimensional.

But suppose a VLSI chip designer takes a cavalier approach. He just tells the manufacturer which nodes are to be connected, and not only does not specify the layout of the wires, but does not even leave any room for them. Moreover, he assumes the ability to change those connections in the run-time. And with all this he expects the manufacturer to somehow implement his chip without an area increase or a computation slow-down of more than a constant factor. We give an operating system, that allows, in fast models, to escape any penalty for such a sloppiness.

Let an $n\times n$ mesh of cellular automata[3] have an additional "link" power: any string A of nodes may write the relative address of any other (equally long) string B of nodes distance d away and, $f(d)$ steps later, the information in B gets to A. We simulate in real time such chips by a same size circuit *Linx* implemented without link power.

Formally *links* are connected horizontal[4] strings of automata in special $\{0,1\}$-states bounded on both ends by automata in special $\{),(\}$-states. We call $0,1,),($ the *link states*. No automata transi-

[2]see [Barzdin, Kolmogorov 67] for the 3-dimensional case. The proof is the same for 2,4, or more dimensions.

[3]i.e. array of identical finite automata, such that the next state of each is a function of its current state, its external input, and the current states of its neighbors within a constant distance.

[4]The result easily extends to links of other shapes.

tions change them (they change only when links are "served"). Link content is viewed as a relative address (of the left end) of its *target* — horizontal string of automata the same size as the link. The *link period* $f(d)$ is a monotone function of the distance d from a link to its target, where f is a parameter of the model. $f(d)$ steps after its creation, the link is "served", i.e. its content is replaced (overriding the automata transition rules) by the content (*picture*) of the target at the (*local*) time proportional to the target's relative (local) address (times $f(d)/d$). (See 2.1 for the precise definition of local address/time.)

We call such imaginary virtual arrays *soft chips*. They resemble pointer machines, like in [Kolmogorov Uspensky 58]. A *type* is a class of chips of all sizes built out of the same automata. We will take a liberty to speak about chips, while in fact meaning *types* of chips. The chips are real (called *hard*) if they have no transitions to link states $0, 1,), (.$

Simulation. Let $I(i,t)$ be the external inputs supplied at time t to the automaton C_i on chip C, and $Q_{C,I}(i,t)$ be its resulting state at time t. Chip A *simulates* B (of the same size) if for some function X and all I, i, t, we have $Q_{A,I}(i,t) = \langle Q_{B,I}(i,t), X_I(i,t)\rangle$, provided $Q_{A,I}(i,0) = \langle Q_{B,I}(i,0), x\rangle$ for a special x.

Constructibility. Let our time unit be $f(1) = 1$. The link period f must (as we prove below) be easy to compute. Namely, a monotone function f is *constructible* if $f(d) = d^{O(1)}$ and *hard* chips can compute the list $\{\langle d, f(d)\rangle : f(d–1) < f(d) < t\}$ in time t. f may depend on the chip size D or other parameters written initially in the chip, like $f_D(d) = \min\{D, d\log^2 d\}$. Note that the polynomial bound means that link's space is sufficient to store (though not necessarily to compute) $f(d)$. By at most doubling f, we can make $\log_2 f$ integer. Then f is constructible iff *hard* chips in time t can compute $d = \max\{x : f(x) \le t\}$ and $f(d)$.

Initialization. Let some non-link states of automata be *passive* i.e. remain unchanged as long as all neighbors are passive. At time 0 all but one (corner) automata are passive.

1.2 Main Theorem

Theorem 1 *All (types of) soft chips can be simulated by hard (types of) chips in f-model, if and only if f is constructible and $\sum_d 1/f(d) = O(1)$.*

So, linear delay, say, $f(d) = d/c$, where c is the speed of light, is not sufficient, while $f(d) = d\log^2 d$ is. The constructibility is trivial if f is stored initially on the chip as a table.

1.2.1 Proof of Necessity

Constructibility. Suppose a hard chip A simulates an f-chip. Create one link per line of the chip, so that the link in line i targets a location distance i away. Start a counter in each link when it is created. The counter is stopped when the target picture overwrites the address in the link. The counters contain $f(i)$ when they are stopped. If the i-th counter, when stopped, has the same value as $(i-1)$-st counter, it is erased. Thus the list $\{\langle d, f(d)\rangle : f(d–1) < f(d) < t\}$ is computed in time t. This satisfies one half of the constructibility definition. Next we show the necessity of the polynomial bound. For some d and $T \gg f(d)$, consider a $T \times T$ square on the chip during one time interval of length $f(d)$. Cover the odd lines of the square with links targeting even lines distance d away. The links are created at random times (using external inputs) in the first half of the time period. The square contains the information about the ages of all the links. At the middle of the interval the information density $\Theta(\log f(d)/\log d)$ (length of the value of the age divided by the length of the link) must be $O(1)$. Thus $f(d) = d^{O(1)}$. So, the constructibility of f is necessary.

Convergence of $\sum 1/f$. Consider a horizontal line M through the middle of $n \times n$ chip. Cover the upper half with links targeting locations symmetric with respect to M. The automata of the lower half reflect random bits from their external inputs. Each automaton distance d above M receives a bit from the other half every $f(2d)$ time steps. Therefore, the average density of the information crossing M per step per automaton is $\sum_d 1/f(2d) = O(1)$. As f is monotone, $\sum_d 1/f(d) < \infty$. ■

1.2.2 Structure of the Proof of Sufficiency

The rest of the paper completes the proof by outlining the simulation algorithm. We construct a *hard* chip A to simulate an arbitrary soft chip B. Thus, A-automata must have two components: "work" fields and B-fields (see *Simulation* in 1.1). The B-fields reflect the states of the corresponding B-automata and change according to the transition rule of B (except when links are served). The "work" fields are used to simulate the link powers. Chip A runs the operating system *Linx*, which simulates the link power making the B-fields indistinguishable from the chip B. Linx

is the same for all chips B, except that work fields are large enough to hold B-automata states.[5] *Linx* runs in the work fields of A, interacting with the *B-fields* when expected.

We group links (according to the distances) into *levels*. The general task of a level is *Concurrent Read (CR)*, where any number of links (called *coherent*) may target the same picture. We reduce CR to *Exclusive Read (ER)* which allows no coherent links. For this reduction coherent links select *leaders* which exclusively read the picture and share it with the others. Different levels use separate resources and act independently in most procedures. Some cooperation of levels is, however, needed. It is limited to few subroutines and regulated by special *engagement rules*.

While our algorithm is optimal, its present form is too complicated for practical use (the constants are not optimized). A more careful analysis (and may be some sacrifice of excessive generality) could make it more practical.

2 General Strategy

2.1 Notation and Tools

Local (Relative) Addresses. We break the binary coordinates of a link and its target into the suffixes (all four of the same length) and prefixes. The prefixes must be as long as possible while differing by at most 1 in each dimension. The length k of the suffix is the link's *level*. Link's *target square* includes all t with the same prefix. The four times larger concentric *local* (k-)square includes l itself. *Local addresses* are coordinates within the local square. *Local time* for the square is the last $\log f(2^k)-2$ digits of time. For a fixed k, local time of reading a target is proportional to its local address (and $f(2^k)/2^k$). A *k-corporation* is a set of all links of level k in one local k-square. Each k-corporation has its own resources to service its links. The interval from 0 to the next 0 of local time is called a *corporate cycle*.

Locations and Agents. Automata may, among other transitions, exchange the content of particular fields with their neighbors, which we refer to as *movement* of fields. *Location* is either a link or a group of fields moving in a regular way (independent of the links distribution). A moving group of fields serving a particular link is an *agent*. Unlike locations, agents (and sometimes even their pieces) can move irregularly. Location/agent a is *in* (*on*) b if automata of b include those of a. A (piece of an) agent is always in some location, l_1, until it is moved to another location l_2 by an operation *take*(l_2, l_1) or *drop*(l_1, l_2). Take/drop work only when l_1 is in l_2. They differ because l_1 and l_2 may belong to different levels and each command is issued by the level of its first argument. The rules restricting the use of take/drop are described among other *rules of engagement*. An agent has return and target (local) addresses, and the data fields. Say, an agent is *served* when it is at the return address and has a copy of the target picture in its data fields. Wlog, make length of each agent a power of 4. Agents of the same level and with the same target are called *coherent*. A set of all coherent agents in a corporation is an *agency*.

Roads and Trains. Each level i is assigned to a *class* c_i. Different classes have no interaction. Thus, class is implicitly unique below. There are two types of locations, besides links: (segments of) *i-trains* for level i, and c-roads for levels of class c. Roads stretch diagonally along the chip and move vertically or horizontally, or are fixed and horizontal. Each i-train is on a c_i-road: moving along it in either direction or fixed (achieving diagonal, vertical or horizontal movement on the chip). We represent $f(d)$ as $\Omega(dh(\lfloor \log d \rfloor))$, where $h(k) = O(k^{1.1})$ and $\sum_k 1/h(k) \le 1$ (sec. 3.1). Classes are defined so that $h(i){=}h(j) \stackrel{\text{def}}{=} \overline{h}(c)^2$ and $|i{-}j| > 4\log h(i)$ for any levels i, j in a class $c(=c_i{=}c_j)$, and c has $\le \overline{h}(c)/4$ levels. We space c-roads evenly by $\overline{h}(c)$. Some i-trains are restricted to one i-square (switch roads at the borders), while others to strips of $h(i)$ adjacent i-squares. Each c-road has a *dense* (with fields in each automaton) i-train for *one* $i \in c$, or *sparse* trains of *all* levels of c. Dense i-trains are spaced by $h(i)$. The density of a sparse i-train on its road is $\ge \overline{h}(c_i)/4$.

Tickets. A k-agent's *ticket* is a pointer to a c_k-road, i.e. a counter with its distance to the agent, or the time it enters the local square. We *pad* binary integers by leading zeros to extend the length to the nearest power of 4, and represent pointers by *counters* of the form $x(x)(x)x$, where x is a shorter counter or a padded integer. Thus, we split the integers of a counter in four, counting down (or merge, counting up) whenever their lengths pass a power of 4. Any $2k$ consecutive digits of a counter determine its value or 2^k, whichever is less. This is useful for coordinating agents of different lengths. The *seat* of

[5] If states of A-automata consist of two fields: *state* and (binary) *data* and only the data fields are to be read by the links, then the operating system *Linx is* the same for absolutely all chips.

an agent is the part of its ticket's road (while in the agent's square) which will pass over the agent's current position. Agents with overlapping seats are in *conflict*. All k-agents with tickets to the same road are also in conflict if there are more of them then fit in k-trains on this road. The conflict is *local* if each conflicting agent is in the other's square, and *distant* otherwise.

Slices and Ordering. A *k-slice* of a $D \times D$ square for $D \geq h(k)$ is a $D \times (D/h(k))$ rectangle. It has (within a constant factor) the volume of k-trains inside the square. Unless specified otherwise, we speak of k-slices of k-squares. We assume row-major (left-to-right, top-down) order of locations. Fixed trains/links in a square are *k-sorted* by field α if their k-agents are located in the increasing order of α.

2.2 Algorithm Overview

A link, once created, starts a time counter. In the next corporate cycle Linx creates and serves the agent. At the end of the link period $f(d)$ the picture from the agent is moved to the link. For each level k, Linx initializes the Infrastructure determining the class membership and defining locations. Then Linx keeps iterating CR. Initialization and each run of CR take $O(2^k h(k))$ steps. CR consists of Preprocessing, Active Stage, and Postprocessing. Preprocessing performs two major tasks. First, leaders are selected by Square Sort. Coupled with the Picture Share (PS) in the Postprocessing it reduces CR to ER. Second, Ticketing (Ti) is done to avoid local conflicts in Active Stage. Finally, the Postprocessing eliminates the distortions the cycle (its Ticketing) has caused in the higher levels' locations (Restore Towers) and distributes pictures in the agencies (Picture Share).

The Active Stage consists of $h(k)$ Slice Read (SR) cycles. In each SR a slice from the target square is copied onto the k-trains (Picture Slice Load). In the next cycle the agents targeting this slice will be picked up by the roads and then loaded into trains (Load Up). When all information is in the k-trains, Sort Trains gets the pictures to their agents. Then the agents are UnLoaded, and the next SR cycle starts. Since the read is exclusive there are $< 4^{k+1}/h(k)$ (bits of) agents to pick up. Thus the agents can be marked V or H so that $< 2^{k+1}/\overline{h}(c_k)$ V-agents are in any vertical line in the square, and H-agents in any horizontal one. (We describe loading only V-agents. H-agents are loaded similarly.) So, the k-agents can be loaded on the road in time $O(2^k)$ if there are no conflicts with the other levels of the class. The Ticketing prevents some (local) of these conflicts.

The following *rules of engagement* govern the interaction between different levels (for V-agents):

1. *take/drop*(l_1, l_2) are possible only onto an empty location and if l_1 is in l_2 and belong to the same class as the agent moved. Links and higher level (*senior*) locations cannot *take/drop* k-agents. A k-agent always remains in its k-square.

2. A *prime* agent P is created by a k-link at the start of the link period and disappears when the link is served. It always remains in links and dense trains of its class levels. In Passive Stage it remains in k-locations. In Active Stage it remains in its vertical column v.

3. The *copy* C of P is created when P first meets its ticket's road r_1 and merges into P, when its return ticket's road r_2 brings C back to P. C always remains on $r_{1,2}$ or in dense k-trains. While on $r_{1,2}$, C stays in its *seat* (in v) or in sparse trains of k or junior levels.

4. $r_{1,2}$ can *take* C only in v and *drop* it only on P. No other location can *drop* C. k-locations cannot *take* prime agents from or *drop* them into junior locations.

Note that levels of different classes are completely independent. We will see that level k can ignore junior levels.

2.3 Subproblems

Consider the following simpler problems for a k-corporation S, or a column (row) C of $h(k)$ k-corporations. Since all levels operate simultaneously, we solve these problems (except *Init*) using only k-trains and links, or (when so specified) a regulated access to the roads. Let s be a k-slice of S.

Init(S) In time $O(f(2^k))$ designate all trains/roads for S.

Concurrent Read(S) In $O(2^k h(k))$ serve the agents of S.

Square Sort(S) In time $O(2^k h(k))$ sort the agents of S and mark one leader in each agency.

Exclusive Read(S) In time $O(2^k h(k))$ serve all marked agents of S, none of which target same picture.

Picture Share(S) In time $O(2^k h(k))$ distribute the leaders' pictures throughout their agencies.

Slice Read(s) In time $O(2^k)$ serve the ticketed k-agents targeting (exclusively) the pictures in s.

Ticketing(C) In time $O(2^k h(k))$ ticket the k-agents in C, avoiding local conflicts with agents of level k or higher. The (pieces of) conflicting agents may be moved within C between their locations or to the k-trains, observing the *rules of engagement*. The tickets are "two way": for the roads to Load Up and to UnLoad later.

Restore Towers(C) In time $O(2^k h(k))$ return to their previous locations (the pieces of) the senior agents moved by Ticketing.

Load Up/Unload(s) In time $O(2^k)$ copy to/from the k-trains the k-agents ticketed for s without conflicts.

Train Sort(H) In time $O(D)$ train-sort H (the k-agents in the k-trains inside a $D \times D$ square).

We abbreviate problems names to the first letters.

Lemma 1 *There exist algorithms (sec. 3) for subproblems TA, RS, LU, UL, TS, Init and (using trains and links only) for the following reductions:*
(Init, CR) $\to$ *Linx;*
(ER, SS, PS) $\to$ *CR;* *(SR, Ti, RT)* $\to$ *ER;*
(LU, UL, TS) $\to$ *SR;* $TS \to SS \to PS$.

The algorithm for CR is: SS; ER; PS. The ER algorithm is: Ti; (for all slices do SR); RT. Only LU and UL use roads explicitly. Others use links and trains only. (see sec. 3)

3 Subroutines

3.1 Infrastructure Initialization

We simplify f to the form $\overline{f}(d) = 2^{\lfloor \log d \rfloor} h(\lfloor \log d \rfloor) = O(f(d))$, with $h(i) = O(i^{1.1})$ and $\sum_i 1/h(i) \le 1$. Let $f_1(d) = \min\{f(d),\ d\log^{1.1} d\}$, $f_2(d) = 4^{\lceil C + \log_4 f_1(d) \rceil}$, where $C > \log_4 \sum_d 1/f_1(d)$. Let $i = \lfloor \log d \rfloor$, and let $h_1(i) = f_2(2^i)/2^{(i-1)}$. Then $1 \ge \sum_d 1/f_2(d) \ge \sum_i 1/h_1(i)$.

Class Assignment. Init assigns recursively each level k to class c_k. First, it computes $h_1(i)$ for all $i \le k$ and groups them into classes. Then k is added to the first class c, such that for all $i \in c$: $h_1(k) = h_1(i) \stackrel{\text{def}}{=} \overline{h}_1(c)^2$, $k - i > 4 \log 4h_1(k)$, and $\|c\| < \overline{h}_1(c)/4$. Let $H(a) = \sum_{c=1}^{a} 1/\overline{h}_1(c)$. To see that $H(\infty) < \infty$ consider $C = \{c : \|c\| = \overline{h}_1(c)/4\}$. Indeed, $\sum_{c \in C} 1/\overline{h}_1(c) < \sum_i 1/h_1(i) \le 1$. Also, the values of h_1 have form 4^j with at most $4 \log h_1(i) = 8j$ classes not full to capacity for each j. Thus, $\sum_{c \notin C} 1/\overline{h}_1(c) \le \sum_j 8j/2^j < \infty$.

Roads. Standard interval $I_s \subset [0,1]$ is a set of reals which start with $.s$ in binary. $I_{s_1} \bigcap I_{s_2} = \emptyset$ iff neither string is a prefix of the other. Any interval $I \subset [0,1]$, contains a unique maximal standard subinterval $\overline{I} = I_s \subset I$ of length $|\overline{I}| > |I|/4$, and thus s is of length $|s| < 2 - \log |I|$. A road number k is assigned to class c if k's reverse $k^R \in \overline{I} \subset I = (H(c\text{–}1), H(c))$. Define $\overline{h}(c) = 1/|\overline{I}|$ and $h(i) = (\overline{h}(c_i))^2$. Obviously, $\overline{h}_1 < \overline{h} < 4\overline{h}_1$ and $2^i \cdot h(i) = O(f_2(2^i)) = O(f(2^i))$. Note that the above construction implies that the c-roads are spaced evenly by $\overline{h}(c)$. Constructibility of f allows computing s_j-s in time $O(f(2^i))$.

Sparse k-Trains consist of *cars* of the same length l_k as k-agents and volume $\ge l_k/4$. Cars consist of continuous segments of length $> \overline{h}(c_k)$. Let $C_k = \{i : i \le k, l_i = l_k, c_i = c_k\}$. Represent $s_k = \|C_k\|$ as a binary number padded by leading 0-s to the length $\log \overline{h}(c_k)$. Level k claims cars with interval $\overline{h}(c_k) l_k$ starting at position $s_k^R \cdot l_k$. Segments claimed by several levels are assigned to the lowest one. This assures that the portion of any car claimed by junior levels is the same as their density ($\overline{h}(c)$) plus at most one car of each smaller length (cars of the same length are disjoint). Thus, at least 1/4 of each car is unclaimed by junior levels. Each segment is long enough to keep "courtesy" bits for all other levels. Through these bits separated segments of one car can communicate. Thus, cars can simulate various continuous structures, like counters.

3.2 Square Sort (SS)

Train Sort can adopt algorithms for sorting on the word model mesh from [Schnorr Shamir 86], [Leighton Leiserson Shwabe 89] or other works. adjacent nodes communicate via trains. The slow down is compensated by the the decrease in the number of nodes keeping the time within $O(2^k)$. Some mesh cells may be empty (sorting field: ∞).

A *k-block* is an $a \times b$ rectangle, where $a = b$ or $a = 2b$ or $b < a = l_k$. *Square-Sort(S)* works much like a recursive merge-sort. It marks all agents, divides block S in two, then recursively sorts in parallel and *Square-Merge*s them.

3.2.1 Square Merge (SM)

SM merges two sorted adjacent blocks, X, Y, into one sorted block Z. Locations of Z coincide with those of X plus Y, but use different fields. SM is similar to a standard external merge, treating X, Y as tapes and trains as internal memory. For the first cycle it loads the first non-empty slices x, y, z from X, Y, Z respectively (x, y containing agents and z – empty links). Empty "train seats" are treated as links/agents with ∞ sorting field. Then SM *Train-Sort*s x together with y, and (in separate fields) z. The agents preceded by all agents (not empty train seats) of the other slice must stay on the trains. The others can be placed in the available empty links, then sorted back and unloaded from the trains. Thus at least one of slices x, y, z is exhausted and is replenished for the next cycle (with ∞-s if X or Y is exhausted). The agents preceded by other(s) (including the largest previously unloaded) with the same sorting field are unmarked.

3.3 Ticketing (Ti)

Ti tickets each agent, avoiding local conflicts, to a road that will pass over it during its SR cycle. First, k-corporations do independently *Initial Reservation (IR)*, avoiding conflicts among the k-agents. There may still be conflicts with ticketed (without local conflicts) senior agents. Then, *Local Conflict Reduction (LCR)*, considers each vertical k-*tower* (i.e. an $l_k \times 2^k h(k)$ rectangle) C independently. The set of *locally conflicting agents* ($LCS(C)$) is the set of all i-agents ($i > k$) locally conflicting with k-agents in C. LCR shuffles the agents in LCS(C) within C, reducing $\|LCS\|$ to $< 2^k$. Then the reduced LCS(C) is dispersed among the fixed k-trains in C, allowing the k-agents locally conflicting with it to be ticketed to dense k-train roads.

Initial Reservation numbers agents $\bmod 2^k/\overline{h}(c_k)$ in the snake-like order along the columns, separately for each target slice. First it counts V-agents ($< 2^k/\overline{h}(c_k)$) of each column. Then it sums the numbers to the left of each column, and shifts the counters accordingly.

Local Conflict Reduction cycles the dense k-trains around the tower, taking (the segments of) the senior agents conflicting locally in the k-square (swapping with the agent, if any, already on the train). When all trains have passed, no conflicting senior agents are left in the links. Otherwise, such an agent together with 2^k senior agents which passed it on the trains must have two agents conflicting with the same k-agent and thus with each other. This contradicts to the rules of ticketing senior agents.

3.4 Slice Read (SR)

The heart of the SR cycle is the Load-Up and Un-Load.

Load-Up (UnLoad is similar and uses separate system of roads) places k-agents on the roads to be picked up from there by sparse trains within $O(2^k)$ steps. During LU/UL sparse k-trains may take and keep senior agents (obeying engagement rules), but only while they conflict with k-agents. The LU/UL consists of three stages: During *clearing* the k-seats on the approaching roads are *marked* (in the end of this stage the marks become *labels*). Then each k-seat s is assigned a k-car to *guard* it from its *client*: the (piece of) lowest level $j > k$ agent conflicting for j-labeled seat s. During the next (*usage*) stage, every guard is between its seat and its client, and to the left of the seat. If a guard meets its client it takes it, preventing from ever reaching the seat during this stage. During usage the k-agents can use their seats (guarded from the conflicting senior agents), on the way between the sparse k-trains and locations off the road. The labels disappear when usage ends. In the last, *exiting*, stage the guards return their clients to the guarded seats unless intercepted by their other guards. All three stages occur inside one stage of any senior level. Therefore, if a senior level j guards a more senior i-agent in the beginning of the three stages it continues to do it till their end. This way, every conflicting (piece of a) senior agent is guarded by a car of k or higher level. A collision would occur if a non-empty seat attempted to take an agent (from a sparse train or from off the road). A senior agent taken onto (or off) the road inside a k-square, cannot collide with k-agents (or be k-guarded), since such a conflict would be local. If the senior agent remains on the road through the k-cycle it must be guarded by either k or higher level and thus cannot get to the k-labeled seat in this k-square. So, no collisions occur. Clients are recognized by guards using counters with distances to the seats.

3.5 Postprocessing (RT and PS)

Restore Towers is essentially similar to LCR and is performed by sorting the tower by the "train-return-address".

Picture Share First Square Sorts all agents, including non-leaders (by target address). Then it loads each slice on the trains, distributes the leaders' pictures through the sorted agencies on the fixed trains, and unloads the slice. Finally, the corporation is Square Sorted by return addresses.

4 Variations

4.1 Alternative Statement of the Main Theorem

The main theorem is also conveniently expressed in terms of *Kolmogorov Complexity*, $K(d)$ which is the length of the shortest binary program from which d is computable by the universal prefixless algorithm. The function K is useful because of its property of being the smallest (within a constant) r.e. function with $\sum 2^{-K(d)} < \infty$.

Main Theorem. (Alternative formulation) *All (types of) f-chips can be simulated by* hard *(types of) chips, iff f is constructible and $f(d) = \Omega(2^{K(d)})$.*

If the function f is produced by an outside device and is input to the chip, then, $f(d) = d \cdot 2^{K(\log_2 d)}$ can be achieved.[6]

4.2 Timing

The "picture taking" time (at which the picture received by the link appeared at its target) could be determined in several ways: 1) "ordered" by the link; 2) quickly and locally computable by the link (as it is in our case); 3) obtained from the "time stamp" of the picture when it is received. Case 3) would simplify the algorithm but complicate seriously the notion of simulation in the Main Theorem. The first case would render our model to be the most powerful and flexible, but we can't break Murphy's Law too often. Here is why simulation of such model is impossible: Consider a $k \times k$ square, where each automaton is targeted by $k/2$ links requesting consecutive time points. Then, $\Theta(k^3)$ bits must stay at or leave the square, while only $O(k^2)$ can.

[6] It is the smallest monotone f in $\Omega(2^{K(d)})$ [Gacs 87].

5 Write and R/W Generalization

Straightforward *simplifications* of the algorithm produce Exclusive Write. Concurrent Write is also possible given a conflict resolution rule. A unification of Read and Write would be to put two local addresses per link so the picture from the first address be copied to the second in one link period.

Acknowledgements

We thank Alok Aggarwal for many most helpful comments.

References

[1] A. Aggarwal, "Period-Time Tradeoffs for VLSI Models with Delay," IEEE, Proc., FOCS 1983, pp. 372-384. Submitted to SIAM J. Comput, 1989.

[2] A. Aggarwal, "Tradeoffs for VLSI Models with Subpolynomial Delay," ACM, Proc., STOC 1985.

[3] A.V. Aho, J.D. Ullman, and M. Yannakakis, "On Notions of Information Transfer in VLSI Circuits," ACM, Proc., STOC, 1983.

[4] R.P. Brent and H.T. Kung, "The Chip Complexity of Binary Arithmetics," ACM, Proc., 12th STOC, 1980.

[5] Ja.M. Barzdin', A.N. Kolmogorov, "On Realization of Nets in 3-Dimensional Space," *Problems of Cybernetics*, 19:261-268 (1967).

[6] G. Bilardi, M. Pracchi, and F.P. Preparata, "A Critique and an Appraisal of VLSI Models of Computation," IEEE J. on Solid State Circuits, SC-17(4):696-702, 1982.

[7] B. Chazelle and L. Monier, "A Model of Computation for VLSI with Related Complexity Results," ACM, Proc. 13th STOC, 1981, pp.318-325.

[8] W.D. Hillis, "The Connection Machine," MIT Press, Cambridge, Mass, 1985.

[9] Peter Gacs, "Lecture Notes on Descriptional Complexity and Randomness" Technical Report 87-013, Computer Science Department, Boston University, 1987.

[10] Z. Galil and W. Paul, "An Efficient General Purpose Computer," ACM, Proc., STOC, 1981.

[11] A.N. Kolmogorov, V.A. Uspenskii, "On the Definition of an Algorithm," *Uspekhi Mat. Nauk,* 13:3-28 (1958); *AMS Transl.* 2nd ser. 29:217-245 (1963).

[12] H.T. Kung, "Why Systolic architectures," Computer Magazine, IEEE, January 1982.

[13] H.T. Kung and C.E. Leiserson, "Systolic Arrays (for VLSI)," SIAM, Sparse Matrix Proc. 1978, ed. I.S. Duff and G.W. Stewart, pp. 256-282.

[14] F.T. Leighton, C.E. Leiserson and E. Shwabe, "Theory of Parallel and VLSI Computation," Lecture Notes, MIT/LCS/RSS 6, 1989.

[15] C.E. Leiserson and B.M. Maggs, "Communication-Efficient Parallel Algorithms for Distributed Random-Access Machines," Algorithmica, (3):53-77, 1988

[16] L.A. Levin, "VLSI Complexity," Section 3 of NSF Proposal MCS-8304498, 1982.

[17] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass, 1980.

[18] C. Mead and M. Rem, "Minimum Propagation Delays in VLSI," IEEE *J. on Solid State Circuits*, SC-17:773-775, 1982. Correction: *Ibid*, SC-19:162 (1984).

[19] D. Nath, S. N. Maheshwari, and P. C. P. Bhatt, "Efficient VLSI Networks for Parallel Computation Based on Orthogonal Trees," IEEE Trans. Comput. 1983.

[20] M. Paterson, W. Ruzzo, and L. Snyder, "Bounds on Minimax Edge Length for Complete Binary Trees," ACM, Proc., 13th STOC, 1981.

[21] V. Ramachandran, "On Driving Many Long Lines in a VLSI Layout," IEEE, Proc. 23rd FOCS, 1982.

[22] C.P. Schnorr and A. Shamir, "An Optimal Sorting Algorithm for Mesh Connected Computers," ACM, Proc. 18th STOC, 1986.

[23] C. L. Seitz, "Self-timed VLSI systems," Caltech Conference on VLSI, 1979, pp. 345-354.

[24] Ch.L. Seitz, "Ensemble Architectures for VLSI - A Survey and Taxonomy," Proc. of MIT Conference on Advanced Research in VLSI, ed. P. Penfield, Jr., Artech House, 1982, pp.130-132.

[25] C.D. Thompson, "A Complexity Theory for VLSI," Ph.D. dissertation, Computer Science Dept., Carnegie-Mellon University, 1980.

[26] L.G. Valiant, "A Scheme for Fast Parallel Computation", SIAM J. Comput., **11** (2), 1982, 350-361.

[27] P.M.B. Vitanyi, "Area Penalty for Sublinear Signal Propagation Delay on Chip," IEEE, Proc, 26th FOCS, 1985, pp.197-207.

[28] P.M.B. Vitanyi, "Locality, Communication, and Interconnect Length in Multicomputers," SIAM J. Comput., 1988.

[29] P.M.B. Vitanyi, "Non-Sequential Computation and the Laws of Nature," VLSI Algorithms and Architectures, Lecture Notes in Computer Science 227, pp. 108-120, Springer Verlag, Berlin, 1986.

[30] A.C. Yao, "The Entropic Limitations on VLSI Computation," Proc. STOC, 1981.