I/O Overhead and Parallel VLSI Architectures for Lattice Computations

Mark H. Nodine
Daniel P. Lopresti
Jeffrey S. Vitter

Computer Science Department
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-89-18

March 1989

I/O Overhead and Parallel VLSI Architectures for Lattice Computations

Mark H. Nodine

Daniel P. Lopresti

Jeffrey S. Vitter

Department of Computer Science

Brown University

Providence, RI 02912

March 6, 1989

Summary

In this paper we introduce a complexity measure for inputs and outputs (I/Os) in two-dimensional lattice computations called the I/O overhead ψ. We show by pebbling arguments that a lower bound for the I/O overhead is proportional to n^{-1}, when there are n^2 processing elements available for the computation. We show that if the results are required to be read at every generation, that the lower bound is the constant 2. We then examine four architectures and show that one of them, the multi-generation sweep architecture, also has I/O overhead proportional to n^{-1} and we derive and compare the constants of proportionality. Finally, we prove a closed-form for the discrete minimization equation giving the optimal number of generations to compute for the multi-generation sweep architecture.

1. INTRODUCTION TO LATTICE COMPUTATIONS

A two-dimensional cellular automaton, in its simplest form, is a discrete, infinite rectangular grid of cells, each of which assumes one of two possible states (on or off) at any given instant. The evolution of a cellular automaton takes place in discrete time steps. Each cell determines in parallel what its state will be at the next time step, based on its current state and the states of the cells around it. Cellular automata can be generalized to lattice computations, in which each cell retains more than a single bit of information. In this paper, we restrict ourselves for brevity to the class of lattice computations where the computation at each cell requires information from its eight neighboring cells, as well as from itself. We call these 9-cell lattice computations. The algorithms and lower bounds we develop can be extended easily to more general lattice computations.

A famous example of this kind of cellular automaton is the game of "Life", which was introduced by John Conway in 1969. Despite the apparent simplicity that purely local rules govern the time evolution of the system, Life exhibits complex behavior and in fact possesses the same computational power as the Turing machine [Dewd85] and admits a universal constructor to allow self-replicating structures [Poun85].

Lattice computations can be used in physical simulations. Examples of simulations which use such automata are 2-dimensional lattice gas computations [Kuge87] and 2-dimensional diffusion, fluid dynamics, spin glasses, and ballistics [Toff87]. A chip to solve the Poisson equation has been implemented using lattice computation techniques [Mano88].

Highly local data movement coupled with a tremendous potential for parallelism would seem to make these problems an ideal match for VLSI. It has been demonstrated, however, that VLSI-based machines performing such computations are severely constrained by input/output (I/O) requirements [Kuge87]. Using a simpler new argument, we improve, by a constant factor, the theoretical lower bound on I/O which can be derived from [Kuge87]. We then present and analyze four VLSI architectures within this framework. One of these, the multi-generation sweep architecture, is optimal in that it meets the lower bound asymptotically to within a small constant factor. In all cases, we derive and compare the constants of proportionality.

The quantity of interest in the following analysis is I/O overhead, which we define as

$$\psi \equiv \frac{I + O}{Cg}, \tag{1}$$

where

I = the total number of input operations,

O = the total number of output operations,

C = the total number of cells computed,

g = the total number of generations computed.

The I/O overhead ψ reflects the amount of I/O needed per unit of computation, independently of the problem size. This quantity does not assume that the results must be viewed at every generation; the number of I/O operations is amortized against the amount of progress made.

In the next section, we give a simple lower bound argument based on pebbling to show that the I/O overhead for 9-cell lattice computations is at least $1/\sqrt{2S}$, where S is the amount of on-chip storage. In Section 3, we give several architectures for doing lattice computations, one of which, the multi-generation sweep architecture, is asymptotically equivalent to the lower bound. Section 4 proves a closed form for the optimal number of generations to compute in the multi-generation sweep architecture before looking at the results. Our conclusions are given in Section 5.

2. LOWER BOUNDS ON I/O OVERHEAD

Graph pebbling is a powerful technique for proving computational lower bounds [Cook73], [Hong81], [Sava87], [Kuge87]. In the case of 9-cell lattice computations, a general result in this last work can be specialized to

$$\psi \geq \frac{1}{8\sqrt{\alpha}} n^{-1} ,$$

assuming that each of n^2 processors (e.g. an $n \times n$ array) possesses α bits of local storage. This result can be improved by factor of $\sqrt{2}$ using their arguments by taking advantage of the connectivity of 9-cell lattice computations instead of the 5-cell computations Kugelmass *et al.* considered.

In this section we improve upon this lower bound on ψ by using a much simpler argument. This analysis makes no assumptions about when results of the computation are viewed. If the states of all cells must be examined after each generation, we are able to prove that $\psi \geq 2$. This last result implies that VLSI implementations might best be used for analyzing the long-term behavior of cellular automata.

Our arguments, like those of [Kuge87], are based on the red-blue pebble game. The red-blue pebble game is played by placing pebbles on the vertices of a **computation graph**, $G = (V,E)$, which is a directed acyclic graph (DAG) that models a computation. Each vertex in$V$ corresponds to a result which is computed at some stage of the computation. An edge $e = (v_1,v_2) \in E$ in the graph indicates that the result corresponding to vertex v_1 is used to compute the result for vertex v_2. Pebbles represent memory locations, and when a pebble is on a vertex, it indicates that the result is in some memory location. Specifically, a red pebble represents a result that is in memory on the chip, and a blue pebble represents a result that is in off-chip memory. There is an implicit assumption that the number of red pebbles is limited, while the number of blue pebbles is as large as needed. The actual pebbling takes place according to the following rules:

1. A pebble of either color may be removed from a vertex at any time.
2. A red pebble may be placed on any vertex which has a blue pebble.
3. A blue pebble may be placed on any vertex which has a red pebble.
4. If all the immediate predecessors of a vertex v have a red pebble, then a red pebble may be placed on v.

Rule 1 represents the forgetting of information (usually to re-the memory which was holding a previous result). Rules 2 and 3 model input and output operations, respectively. Rule 4 models a computation taking place within the chip. In this paper we ignore internal computations and consider only I/O. Our measure of performance is the number of applications of rules 2 and 3.

Those vertices of the graph that have no predecessors are the inputs to the chip, and those that have no successors are the outputs. The starting configuration has blue pebbles on all the inputs.

Definition: A **pebbling** of a computation graph is a sequence of ordered pairs, (n, v), where $n \in \{1,\ldots,4\}$ is a rule number and $v \in V$, such that starting from the initial configuration and applying the rules to the

nodes in sequence results in all of the outputs having blue pebbles. An **S-pebbling** is a pebbling with the restriction that at most S red pebbles are used at any one time. A **blocked pebbling** is a pebbling where the inputs and outputs are done a group at a time.

We define the computation graph, $G_g = (V_g, E_g)$, for doing a lattice computation for g generations as follows: We assume that the size of the problem during all g generations can be bounded by an $m \times m$ array of cells. The cells on the border have either three or five neighbors, rather than eight neighbors. The vertex set is

$$V_g = \{(i,j,t) \mid 0 \le i,j < m \text{ and } 0 \le t \le g\},$$

and the edge set is

$$E_g = \{((i,j,t),(i',j',t')) \mid (i,j,t),(i',j',t') \in V_g \text{ with } t' = t+1,\ |i'-i| \le 1, \text{ and } |j'-j| \le 1\}.$$

The computation is then a sequence starting at time $t = 0$ and progressing towards larger t. When $t = g$, the outputs have been computed, and the computation halts.

Theorem 1: For lattice computations with $g \ge 2$, in which there are $S < m$ bits of on-chip storage, we have

$$\psi \ge \frac{1}{\sqrt{2S}}.$$

Proof: The basic idea of the proof is twofold: First, we use Lemma 1 below to show that every S-pebbling strategy with T I/Os can be simulated efficiently by a blocked $2S$-pebbling of roughly T/S blocks in which inputs are blocked together S at a time and outputs are blocked together S at a time. Then we use Lemmas 2 and 3 to bound the number of internal computations that can be done on the lattice between the time of one blocked input and the time of the next blocked input. Since we know how many nodes there are in the lattice that must be red-pebbled, this gives us a lower bound on the number of inputs required.

From Lemma 3, below, the maximum number of internal computations that can be done without I/O using $2S$ red pebbles is

$$\frac{(2S)^{3/2} - 2S}{2}.$$

Since there are a total of gm^2 lattice elements which must be red-pebbled, this means that the number of blocks of inputs is at least

$$G = \left\lceil \frac{gm^2}{\frac{1}{2}((2S)^{3/2} - 2S)} \right\rceil \ge \frac{gm^2}{\frac{1}{2}(2S)^{3/2}} + \frac{gm^2}{2S^2} \ge \frac{gm^2}{\sqrt{2}\,S^{3/2}} + 1.$$

Here, we have used the facts that $m > S$ and $g \ge 2$. Since each S-pebbling of the lattice with T I/Os has a corresponding blocked $2S$-pebbling with $\lceil T/S \rceil$ input blocks, every S-pebbling must have at least $S(G-1)$ input operations. Thus, from formula 1 defining the I/O overhead, the number of input operations is at least

$$\frac{gm^2}{\sqrt{2S}},$$

leading to

$$\Psi \geq \frac{1}{\sqrt{2S}}. \blacksquare$$

Lemma 1: For any red-blue S-pebbling of T_i inputs and T_o outputs, there exists a red-blue blocked $2S$-pebbling that can simulate it of the following type:

1. There are $\lceil T_i/S \rceil$ groups of inputs, each consisting of at most S inputs, and $\lceil T_o/S \rceil$ groups of outputs, each consisting of at most S outputs.

2. The individual I/O operations are a subsequence of the original pebbling.

Proof: This can be proved by an easy simulation of the original pebbling. We can think of S red pebbles as keeping track of outputs in the original schedule that are being buffered for writing in a group. The other S red pebbles are used for inputs and internal computations. Clearly this can be done because the original pebbling uses only S pebbles. We cache the inputs together. We include any input in the schedule that is not already resident in internal memory or the input cache, and that is not computable from such nodes via rule 4; otherwise the input is not needed. For the case of the output schedule, outputs that will be overwritten by a later output in the same group can be ignored. This proves Lemma 1. ■

Lemma 2: Let us consider a pebbling game in which an initial configuration of $S < m^2$ red pebbles is given, and only rules 1 and 4 are allowed for pebbling moves (that is, only internal computations and removing pebbles are allowed). Then there is a pebbling for maximizing the number of computations in the lattice such that all red pebbles are placed on nodes in earlier generations before any are placed on nodes in later generations.

Proof: Let R be the set of vertices pebbled by some pebbling starting from an initial configuration K of S–1 red pebbles. (We do not need to consider configurations of S red pebbles, because the first operation starting from such a configuration will always be an application of rule 1, resulting in a configuration of S–1 red pebbles.) We show that there is a schedule which pebbles earlier generations before later ones that results in all the vertices in R being pebbled. Let $g + 1$ be the first generation that contains some node in R. Since $S < m^2$, there is a pebble on some vertex v on generation g which contributes to the placing of exactly one pebble on generation $g + 1$, say on vertex w. Apply rule 4 to place a red pebble on w and apply rule 1 to remove the pebble from vertex v. Since the pebble on vertex v was used only in computing w, removing the pebble from there does not decrease the number of possible vertices which can be pebbled. Moreover, we can consider this as a new starting configuration K' with a new set $R' = R - \{w\}$, since again at most $S - 1$ pebbles are on the graph. Continuing in this way guarantees that all the vertices in R eventually have a red pebble on them.

Since this can be done for any pebbling, then in particular any pebbling which maximizes the number of nodes pebbled can be redone in an order which pebbles earlier generations before later ones. ■

Lemma 3: The maximum number of nodes in the lattice that can be pebbled using rule 4 alone, given a starting configuration of S red pebbles, is $(S^{3/2}–S)/2$, assuming that $S < m$.

Proof: Let us consider a schedule that maximizes the number of nodes pebbled, using rule 4 only. By Lemma 2 we can assume that the schedule proceeds generation by generation, for generations 0, 1, ..., g–1. Each generation will be pebbled in parallel. Let $A_i \geq 0$ be the number of pebbles moved while pebbling

generation i+1 from generation i, and let $B_i \geq 0$ be the number of pebbles "left behind" on generation i. We have

$$\sum_{0 \leq i \leq g-1} B_i = S, \tag{2}$$

since each red pebble must eventually be left behind, or more moves would be possible. By the constraints of the lattice, for any value of i we have

$$A_i \leq \left(\sqrt{S} - 1\right)^2. \tag{3}$$

It is easy to see that A_i is maximized if all S pebbles are on level i at some point in the schedule, arranged in a square in a corner of the lattice. The fact that $S \leq m$ means that the square can touch at most one corner. If this is the case, then A_i will be exactly $\left(\sqrt{S} - 1\right)^2$.

Likewise, the border of each cluster of pebbles must be left behind unless the border is at the edge of the lattice. The perimeter of a cluster of S pebbles is at least $4\sqrt{S}$, at most half of which can be along the edge of the lattice, so we have

$$B_i \geq 2\sqrt{A_i}. \tag{4}$$

We can get an upper bound on $\sum_i A_i$, the number of nodes pebbled, by maximizing $\sum_i A_i$ subject to the constraints (2), (3), and (4), where A_i and B_i are allowed to be real nonnegative numbers. A simple convexity argument shows that $\sum_i A_i$ cannot exceed a case where $A_i = \left(\sqrt{S} - 1\right)^2$ for $S/\left(2(\sqrt{S}-1)\right)$ values of i. This gives us the bound $\sum_i A_i \leq (S^{3/2}-S)/2$, which proves the lemma. ■

There is a corollary to the theorem which we will use in the next section.

Corollary 1: In an array of n^2 processors, each of which has α bits of storage, the I/O overhead is at least

$$\psi \geq \frac{1}{\sqrt{2\alpha}\, n}. \tag{5}$$

Proof: We simply set $S = \alpha n^2$. ■

What kind of lower bounds can we derive from pebbling arguments if we insist that we want to see the results of the calculation after every generation?

Lemma 4: The I/O overhead to compute one generation of a lattice computation and look at the results is at least 2.

Proof: In this case, the computation graph consists of only two layers, each with m^2 cells. Clearly it will take a minimum of m^2 inputs to red-pebble the inputs and a minimum of m^2 outputs to blue-pebble the outputs, once they have been red-pebbled. Thus, we have

$$\psi \geq \frac{2\,m^2}{m^2} = 2. \blacksquare$$

Corollary 2: Consider a lattice computation where we wish to examine all the cells after each generation has been computed. If no values are retained on the chip between generations, then the I/O overhead $\psi \geq 2$, regardless of the number of generations computed.

This lower bound is clearly worse than that in Corollary 1, where the cells do not have to be examined at each generation. This implies that the usefulness of a VLSI chip to do lattice computations would be enhanced if it were used in studying asymptotic behavior of initial configurations.

3. ARCHITECTURAL APPROACHES

In this section we present and analyze four parallel VLSI architectures for Life. All employ a two-dimensional processor grid. The limited-size architecture attempts to keep the entire problem in the processor array. The various sweep architectures permit the processor array to sweep over portions of a much larger problem. In the case of the sweep architectures, the I/O overhead is computed for the "steady-state" case, that is, assuming that the problem size is large with respect to the number of processors.

3.1. LIMITED-SIZE ARCHITECTURE

The most straightforward approach to simulating a lattice computation is to assume there exists a one-to-one correspondence between processors and cells. For this we use n^2 processors arranged in an $n \times n$ square mesh, each communicating directly with eight neighbors, as shown in Figure 1. Only computations of size $m \times m$, with $m \leq n$, can be considered, as there are no provisions for saving intermediate results. Each processor requires one bits of storage to hold its current state. The algorithm used is:

1. Input $m \times m$ problem.
2. Compute for G generations.
3. Output results.

A short derivation using equation (1) shows that the I/O overhead for this architecture is

$$\psi_{ls} = 2n^{-1} + 2mn^{-2} + O(m^2n^{-3}) .$$

With $\alpha = 1$, the lower bound from equation (5) tells us that

$$\psi > \frac{1}{\sqrt{2}} n^{-1} .$$

Thus, the limited-size architecture has an I/O overhead at most 2.83 times the optimal. In practice, of course, this scheme is not useful because it ignores problems larger than $n \times n$.

3.2. ARRAY SWEEP ARCHITECTURE

In this architecture, the complete lattice computation is stored in an $m \times m$ grid of memory cells. The processor array repetitively loads $n \times n$ subproblems, updates states by a single generation, and writes results back to their original locations. In this manner, the processors wind their way through the problem space, tessellating the computation as demonstrated in Figure 2. The algorithm for this architecture is:

1. For each subproblem in tessellation, do steps 2-4.
    2. Input $n \times n$ subproblem.
    3. Compute for 1 generation.
    4. Output results.

Here the I/O overhead is found to be

$$\psi_{as} = 2 + 4n^{-1} + 4n^{-2} .$$

Because the array sweep architecture outputs each state at every generation, theory predicts that $\psi \geq 2$. This is indeed the case.

A slight improvement over this is possible by observing that the rightmost two columns of a given computation become the leftmost two columns in the next computation; these values need not be written to memory as they will always be re-read immediately. With this change the I/O overhead is reduced to

$$\psi_{mas} = 2 + 2n^{-1} .$$

3.3. N/2-GENERATION SWEEP ARCHITECTURE

The array sweep architecture computes only a single generation for each subproblem before moving on to the next. This policy seems wasteful when I/O overhead is a primary concern. Why not compute multiple generations once a given subproblem has been loaded? The potential pitfall here is that states at the memory-processor border are not updated as they should be. As the computation progresses, these incorrect values propagate their effect inward. Eventually, after $(n-1)/2$ generations, only the centermost cell is correct (assuming n is odd). Nevertheless, we can proceed by saving this one valid value and loading a new subproblem. Fortunately, this last step is greatly simplified if each processor stores an additional bit, its original state. These values are left-shifted one position before beginning the next computation, so that only $n+2$ new values need be input along the array's right border. In this architecture, we no longer tessellate the problem space, but truly sweep it. The $n/2$-generation sweep architecture employs the following algorithm:

1. For each row in problem space, do steps 2-5.
    2. For each column in problem space, do steps 3-5.
        3. Left-shift original states one position, input $n+2$ new values.
        4. Compute for $(n-1)/2$ generations.
        5. Output result from centermost processor.

The I/O overhead can be computed as

$$\psi_{n/2} = 2 + 4n^{-1} - 4n^{-2} + O(n^{-3}) .$$

Thus, this architecture is slightly better than the original array sweep architecture, but not as good as the modified version.

3.4. MULTI-GENERATION SWEEP ARCHITECTURE

The previous two schemes are, in fact, the extreme cases in a spectrum of architectures. It is possible to work anywhere between the array sweep architecture and the $n/2$-generation sweep architecture, as shown in Figure 3. Processor requirements in this case are the same as those in the $n/2$-generation sweep architecture, except for the need to output a $k \times k$ matrix of results. The algorithm is changed slightly so that the array computes only $(n-k)/2 + 1$ generations in step 4.

I/O overhead for this architecture is easiest to derive in terms of k, but can be related to g since

$$k = n - 2\,(g - 1).$$

The equation is

$$\Psi_{mgs} = \frac{2(n - g + 2)}{g(n - 2g + 2)}. \tag{6}$$

If the asymptotics are done directly from this equation for I/O overhead, one gets the unrealistic idea that the I/O overhead can become arbitrarily good if the number of generations computed goes to infinity. This is not only unreasonable, since at most $\lfloor n/2 \rfloor$ generations can be validly computed, but it also seems to contradict the finding for the $n/2$-generation sweep architecture, which has a constant I/O overhead. The problem is that the approximation used in generating the asymptotic expression becomes invalid when g approaches $n/2$.

We can adopt a different approach and compute the value of g which minimizes the expression for I/O overhead. By setting the partial derivative of Ψ with respect to g equal to zero, we get

$$\tilde{g}_{min} = n + 2 \pm \frac{\sqrt{2}}{2}\sqrt{n^2 + 4n + 4}$$

$$= \left(1 \pm \frac{\sqrt{2}}{2}\right)(n + 2),$$

of which the more negative root gives the minimum. The tilde superscript is to indicate that this quantity is always an integer multiple of an irrational number, and is therefore always non-integral. What we want is the discrete minimum for equation (6). Taking the nearest integer, in fact, gives the discrete minimum (see section 4):

$$g_{min} = \left\lfloor \left(1 - \frac{\sqrt{2}}{2}\right)(n + 2) + \frac{1}{2} \right\rfloor.$$

Thus, we can calculate the asymptotic I/O overhead by noting that

$$g_{min} = \left(1 - \frac{\sqrt{2}}{2}\right)n + O(1),$$

so

$$\Psi_{mgs} = \left(6 + 4\sqrt{2}\right)n^{-1} + O(n^{-2}).$$

The pebbling bound in this case (assuming $\alpha = 2$) is

$$\Psi > \frac{1}{2}n^{-1}.$$

Thus, this architecture is asymptotically optimal and falls within a factor of around 23 of the best possible I/O overhead.

4. PROOF OF VALIDITY OF DISCRETE MINIMIZATION FORMULA

In the previous section, we found that the I/O overhead for the multi-generation sweep architecture was

$$\psi(g, n) = \frac{2\,(n - g + 2)}{g\,(n - 2g + 2)}.$$

Using calculus, we found that the value of g that minimizes ψ is

$$\tilde{g}_{min}(n) = \xi\,(n + 2),$$

where

$$\xi = 1 - \frac{\sqrt{2}}{2}.$$

We then stated without proof that taking the nearest integer function of this provided an integer value at which the minimum occurs:

$$g_{min}(n) = \left\lfloor \xi\,(n + 2) + \tfrac{1}{2} \right\rfloor.$$

This section proves that assertion.

Theorem 2: The formula for g_{min} above gives an integer value that minimizes $\psi(g, n)$.

Proof: The basic argument for the proof is given here; the lemmas proved below fill in the details. The values of $\tilde{g}_{min}$ consist of all the multiples of an irrational number, ξ. The nearest integer function will give the wrong answer only if the fractional part of that multiple of ξ is sufficiently close to 1/2 that the nearest integer function will round it one way, but the minimum would occur by rounding it the other. Lemma 5 shows that, for positive n, this will only happen if the fractional part is slightly larger than 1/2. If we let $m = \lfloor \xi n \rfloor$ then Lemma 6 computes the "window of vulnerability" $\varepsilon(m)$, that is, the maximum amount by which the fractional part of a failing value of n can exceed 1/2. This window of vulnerability is a decreasing function of m. Let N be the set of all n for which $\left\lfloor \tilde{g}_{min}(n) + 1/2 \right\rfloor \neq g_{min}(n)$. Then if $N \neq \varnothing$, it has a smallest element n'. Define

$$\chi(n) = \mathrm{Frac}\left(\xi n - 1/2\right), \tag{7}$$

so that $\chi(n)$ is the amount by which the fractional part of ξn exceeds 1/2 (or a value greater than 1/2 if the fractional part does not exceed 1/2). If $n'' < n'$, then $\chi(n') < \chi(n'')$, since otherwise $\xi n''$ would fall within its larger window of vulnerability, contradicting the assumption that n' is the smallest failing multiple. So we need consider only those values of n such that $\chi(n)$ is smaller than for any preceding value. Lemma 9 proves the explicit form of those numbers a_k. Finally, Lemma 10 demonstrates that each a_k falls exactly at the edge of a window of vulnerability, which means that

$$\psi(a_k - 2,\ g_{min}(a_k - 2)) = \psi(a_k - 2,\ g_{min}(a_k - 2) - 1),$$

so that rounding to either side produces the minimum. This concludes the proof of the Theorem. ■

4.1. INTRODUCTION TO THE STERN-BROCOT TREE

The proofs of the lemmas depend heavily on the properties of the Stern-Brocot (S-B) tree[Grah89]. This section explains how the tree is derived and gives the properties it has that are important to the proof.

The S-B tree starts with the fractions 0/1 and 1/0 and derives the next level by inserting between every pair of fractions m/n and m'/n' the fraction $(m + m')/(n + n')$. Figure 4 shows the top part of the tree. The fractions which evaluate to 0 and ∞ are not really part of the tree; they are merely seeds to get the tree started The S-B tree has several important properties:

1. All fractions that appear in the tree are in reduced form.
2. All reduced-form fractions appear in the tree.
3. If the fractions are read from the tree by inorder traversal, they are in ascending order.
4. We can treat the S-B tree as a binary search tree. If "L" means go down the left branch of the tree and "R" means go down the right branch, then all real numbers can be mapped to a unique element of the regular language $[LR]^{\infty}$. For rational numbers, that is those which can be given a finite representation, the unique infinite representation is the finite part followed by RL^{∞}. The element of $[LR]^{\infty}$ so generated is called the **S-B expansion** of a number.
5. If β is an irrational number, then the fractions generated in its S-B expansion are the simplest rational approximations to β in the sense that if m/n is an approximation to β, there exists an S-B expansion m'/n' such that $m' \le m$, $n' \le n$ and m'/n' is between m/n and β.

4.2. PROOF OF THE LEMMAS

The first lemma establishes that only multiples of ξ whose fractional parts are slightly greater than 1/2 could possibly cause the equation to fail.

Lemma 5: Let $h(n, \varepsilon)$ be the symmetric difference of $\psi(n, g)$ about its minimum for $\varepsilon > 0$, that is,

$$h(n, \varepsilon) = \psi(n, \tilde{g}_{min}(n) + \varepsilon) - \psi(n, \tilde{g}_{min}(n) - \varepsilon).$$

Then $h(n, 1/2) > 0$ for all integer $n > 0$.

Proof: This is mostly a matter of algebra. We find that

$$h(n, \varepsilon) = \frac{16\,\varepsilon^3}{(17 - 12\sqrt{2})(n+2)^4 - (18 - 12\sqrt{2})\,\varepsilon^2\,(n+2)^2 + 8\,\varepsilon^4}.$$

The numerator is clearly greater than zero for all n. This expression will change signs at those values of n for which the denominator is zero. When $\varepsilon = 1/2$, these values are $-3 - \sqrt{2}$, $-3 - \sqrt{2}/2$, $-1 + \sqrt{2}/2$, $-1 + \sqrt{2}$. Of these values, only the last one is positive, having value approximately 0.414214. Since the value of $h(1, 1/2)$ is about 3.52, and there are no sign changes larger than $n = 1$, we have $h(n, 1/2) > 0$ for all integer $n > 0$. ■

If this function were exactly symmetric about its minimum, then the nearest integer function on $\tilde{g}_{min}(n)$ would always produce the discrete minimum of $\psi(n, g)$. Since it is not, there are "windows of vulnerabil-

ity" if the fractional part of ξn is slightly larger than 1/2. The next lemma shows how large these windows of vulnerability are.

Lemma 6: If m is any integer, then the nearest integer function will produce the wrong result if

$$m + \frac{1}{2} < \xi n < m + \frac{1}{2} + \varepsilon(m),$$

for some n, where

$$\varepsilon(m) = \frac{1}{2}\left(\sqrt{2} - 1\right)\left(\sqrt{4m^2 + 4m + 2} - (2m + 1)\right). \tag{8}$$

Proof: A window of vulnerability exists when ξn falls between $m + 1/2$ and the point such that $\psi(n, m) = \psi(n, m+1)$ for some integer m and n. Given a value of m, we determine n from the equation

$$\xi n = m + \frac{1}{2} + \varepsilon,$$

or equivalently

$$n = \left(2 + \sqrt{2}\right)m + \left(2 + \sqrt{2}\right)\varepsilon - 1 + \frac{\sqrt{2}}{2}.$$

The condition $\psi(n, m) = \psi(n, m+1)$ leads to

$$2n^2 + 4n(-2m+1) + 4m(m-3) = 0.$$

Plugging in the value of n above gives

$$4\left(3 + 2\sqrt{2}\right)\varepsilon^2 + 4\left(1 + \sqrt{2}\right)\varepsilon - 1 = 0.$$

The values of ε solving this equation are

$$\varepsilon = \frac{1}{2}\left(\sqrt{2} - 1\right)\left(\pm\sqrt{4m^2 + 4m + 2} - (2m + 1)\right).$$

The negative root is negative for all n, so the positive root must be the correct one. ■

It is instructive to do the asymptotics on $\varepsilon(m)$ to see how large the windows of vulnerability are. A simple derivation reveals that

$$\varepsilon(m) = \frac{1}{8}\left(\sqrt{2} - 1\right)m^{-1} - \frac{1}{16}\left(\sqrt{2} - 1\right)m^{-2} + O(m^{-3}). \tag{9}$$

The next lemma gives the S-B expansion of ξ.

Lemma 7: The S-B expansion for ξ is L(LLRR)$^\infty$.

Proof: It is clear that the expansion begins with L, since $\xi < 1$. Let those fractions which would underestimate ξ according to the expansion above (those with an "R" in their next digit) be denoted by m_k/n_k and those which would overestimate ξ be denoted by p_k/q_k. We can derive expressions for m_k, n_k, p_k, and q_k by using recurrence relations (see Figure 5). For example, the recurrence for m_k and n_k when k is odd is given by the simultaneous recurrence

$$
\begin{aligned}
m_1 &= 1 ; \\
m_k &= 3m_{k-2} + 4p_{k+1} ; \\
n_1 &= 4 ; \\
n_k &= 3n_{k-2} + 4q_{k+1} ; \\
p_2 &= 1 ; \\
p_k &= 2m_{k-3} + 3p_{k-2}; \\
q_2 &= 3 ; \\
q_k &= 2n_{k-3} + 3q_{k-2}.
\end{aligned}
$$

From this recurrence, we can get generating functions for m_k and n_k when k is odd:

$$
m_{odd}(z) = \sum_{k\ \text{odd}} m_k z^k = \frac{z^3 + z}{z^4 - 6z^2 + 1},
$$

$$
n_{odd}(z) = \sum_{k\ \text{odd}} n_k z^k = \frac{4z}{z^4 - 6z^2 + 1},
$$

which yield

$$
m_{2k-1} = \frac{1}{2}\left(\left(1+\sqrt{2}\right)^{2k-1} + \left(1-\sqrt{2}\right)^{2k-1}\right), \qquad \text{for } k \geq 1;
$$

$$
n_{2k-1} = \frac{1}{2}\left(\left(4+3\sqrt{2}\right)\left(1+\sqrt{2}\right)^{2k-2} + \left(4-3\sqrt{2}\right)\left(1-\sqrt{2}\right)^{2k-2}\right), \qquad \text{for } k \geq 1;
$$

Similarly, we get

$$
m_{2k} = \frac{1}{4}\left(\left(4+3\sqrt{2}\right)\left(1+\sqrt{2}\right)^{2k-2} + \left(4-3\sqrt{2}\right)\left(1-\sqrt{2}\right)^{2k-2}\right), \qquad \text{for } k \geq 1;
$$

$$
n_{2k} = \frac{1}{2}\left(\left(7+5\sqrt{2}\right)\left(1+\sqrt{2}\right)^{2k-2} + \left(7-5\sqrt{2}\right)\left(1-\sqrt{2}\right)^{2k-2}\right), \qquad \text{for } k \geq 1;
$$

Let us define $\Delta_k = \xi n_k - m_k$, which is the fractional part of ξn_k. It is straightforward to show that

$$
\Delta_{2k-1} = \left(3-2\sqrt{2}\right)^k, \qquad \text{for } k \geq 1;
$$

$$
\Delta_{2k} = \xi\left(3-2\sqrt{2}\right)^k, \qquad \text{for } k \geq 1.
$$

Since these quantities are always positive, m_k/n_k always underestimates ξ. Similarly, it can be shown that p_k/q_k always overestimates ξ. Thus, these fractions must constitute the S-B expansion of ξ. ■

The next lemma tells about the convergence properties of m_k/n_k.

Lemma 8: The fractions m_k/n_k converge monotonically to ξ.

Proof: Let Δ_k be as in Lemma 7. Let δ_k be defined by $\delta_k = \xi - m_k/n_k$. Then we have $\delta_k = \Delta_k/n_k$. We get

$$\frac{\delta_{2k-1}}{\delta_{2k}} = \frac{(7+5\sqrt{2})(1+\sqrt{2})^{2k-1} + (7-5\sqrt{2})(1-\sqrt{2})^{2k-1}}{(1+\sqrt{2})^{2k} + (7-5\sqrt{2})(1-\sqrt{2})^{2k-1}}.$$

This converges rapidly to $3 + 2\sqrt{2} \approx 5.282$ and is always greater than 5.28. Similarly,

$$\frac{\delta_{2k}}{\delta_{2k+1}} = \frac{(1+\sqrt{2})^{2k+2} + (7-5\sqrt{2})(1-\sqrt{2})^{2k+1}}{(1+\sqrt{2})^{2k} + (41-29\sqrt{2})(1-\sqrt{2})^{2k-1}}.$$

This also converges rapidly to $3 + 2\sqrt{2}$ and is always greater than 5.28. Since these ratios are both greater than one, these fractions converge monotonically. ■

We want the set of numbers the fractional part of whose multiples of ξ approach 1/2 from the top more closely than any of the preceding numbers. It is easy to see from the generating functions that when k is odd it follows that m_k is odd and n_k is even. Thus, if we define, as before, $\Delta_k = \xi n_k - m_k$, then we get

$$\xi\frac{n_{2k-1}}{2} - \frac{m_{2k-2} - 1}{2} = \frac{1}{2} + \frac{\Delta_{2k-1}}{2},$$

where the two fractions on the left-hand side are integers and Δ_{2k-1} gets exponentially small. Thus, the series of numbers

$$b_k = \frac{n_{2k-1}}{2} = \frac{1}{4}\left((4+3\sqrt{2})(3+2\sqrt{2})^{k-1} + (4-3\sqrt{2})(3-2\sqrt{2})^{k-1}\right), \qquad \text{for } k \geq 1, \tag{10}$$

would seem to be a good candidate for the numbers we are seeking. The next lemma proves that this is actually the case.

Lemma 9: Let us define $\chi(n)$ as in equation (7). The values of b_k defined in equation (10) have the property that

$$\chi(b_k) = \min_{1 \leq n \leq b_k} \{\chi(n)\}.$$

Proof: Assume that there exists an $n < b_k$ such that $\chi(n) < \chi(b_k)$. By definition, there exists an m such that

$$\xi n - m = \frac{1}{2} + \chi(n).$$

Doubling this equation leads to

$$\xi(2n) - (2m+1) = 2\chi(n).$$

Thus, the fraction $(2m+1)/2n$ is a rational approximation which underestimates ξ. By property 5 of the S-B tree, there must be an underestimator to ξ in its S-B expansion, m_j/n_j, with $n_j \leq 2n$, which is at least as good as $(2m+1)/2n$. In turn, $2n < 2b_k = n_{2k-1}$, so that $j < 2k-1$. But now we have two fractions in the S-B expansion of ξ, of which the one with the smaller index approximates ξ better. This is a contradiction

to Lemma 8. Thus, the assumption that there was such a value of n was incorrect, and the lemma is proved. ■

The only thing remaining to finish the proof of the Theorem is to show that $\chi(b_k) \geq \varepsilon(m(b_k))$ for all k, where $m(n)$ is defined by $m(n) = \lfloor \xi n \rfloor$.

Lemma 10: We have $\chi(b_k) = \varepsilon(m(b_k))$.

Proof: We know that

$$\chi(b_k) = \frac{\Delta_{2k-1}}{2} = \frac{1}{2}\left(3-2\sqrt{2}\right)^k.$$

We also know from Lemma 9 that

$$m(b_k) = \frac{m_{2k-1} - 1}{2}.$$

Plugging into equation (8), we get

$$\varepsilon(m(b_k)) = \frac{1}{4}\left(\sqrt{2}-1\right)\left(\sqrt{\left(1+\sqrt{2}\right)^{4k-2} + 2 + \left(1-\sqrt{2}\right)^{4k-2}} - \left(1+\sqrt{2}\right)^{2k-1} - \left(1-\sqrt{2}\right)^{2k-1}\right).$$

The item in the square root is a perfect square, so

$$\varepsilon(m(b_k)) = \frac{1}{4}\left(\sqrt{2}-1\right)\left(\left(1+\sqrt{2}\right)^{2k-1} - \left(1-\sqrt{2}\right)^{2k-1} - \left(1+\sqrt{2}\right)^{2k-1} - \left(1-\sqrt{2}\right)^{2k-1}\right)$$

$$= \frac{1}{2}\left(3-2\sqrt{2}\right)^k.$$

The expressions for $\chi(b_k)$ and $\varepsilon(m(b_k))$ are thus equal. ■

It is known that the set of fractional parts of all multiples of any irrational number β is dense on the unit interval (0, 1), which is to say that given any $0 < \gamma < 1$ and $\varepsilon > 0$, there exists an integer n such that the fractional part of βn is within ε of γ. It turns out that the fractional parts are uniformly distributed along the unit interval. Let us consider the following probabilistic argument about whether the discrete minimum equation is true for all values of n: We assume that in any interval m to $m+1$, there is a probability of $\varepsilon(m)$ that a multiple of ξ falls within the ε-window. If we define

$$\lambda = \frac{1}{16}\left(\sqrt{2} - 1\right),$$

the probability that all of the ε-windows is missed is, from equation (9)

$$\Pr\{\text{everywhere correct}\} \leq \prod_{m \geq 1}\left(1 - 2\lambda m^{-1} + \lambda m^{-2}\right).$$

Taking logs, we get

$$\log(\Pr\{\text{everywhere correct}\}) \leq \sum_{m \geq 1} \log\left(1 - 2\lambda m^{-1} + \lambda m^{-2}\right) \leq \sum_{m \geq 1}\left(-2\lambda m^{-1} + \lambda m^{-2}\right) = -\infty.$$

Therefore, we find that $\Pr\{\text{everywhere correct}\} = 0$.

So it seems that the fact the discrete minimization formula is everywhere correct is a bit of a surprise: even though the fractional parts of the multiples of ξ are uniformly distributed on the unit interval, the probabilistic argument is not valid because the fractional parts of the multiples of ξ do not approach 1/2 from the top until the ε-window has shrunk just enough to be missed.

5. CONCLUSIONS

In this paper we discussed I/O overhead ψ as a measure of merit for parallel VLSI architectures for cellular automata. We derived theoretic lower bounds on ψ based on the red-blue pebbling game. We then presented and analyzed four potential architectures, demonstrating that one, the multi-generation sweep architecture, was optimal in terms of ψ to within a small constant factor. Finally, we proved the discrete minimization formula which results in the optimal performance of the multi-generation sweep architecture.

It is natural to question whether the asymptotic differences exhibited in this paper manifest themselves in chips of reasonable size. Table 1 indicates values of ψ for three of our schemes; the multi-generation sweep architecture is noticeably superior even when n is relatively small. A value of 11 might be achievable with current technology, resulting in better than a factor of 2 in I/O performance; using wafer technology, a chip with 21^2 processors is quite conceivable, in which case there is more than a factor of 4 improvement in the I/O overhead of the multi-generation sweep architecture versus the normal sweep architecture.

It is interesting to note that when n is larger than about 6, the I/O performance of the array sweep architecture is almost independent of n. In a problem like this, for which the limiting factor is likely to be I/O, this gives the non-obvious result that it does not help very much to be able to pack more processors onto a chip.

We conclude that a VLSI implementation of a two-dimensional lattice computation does not need to be severely restricted by its I/O performance as long as it is not necessary to view the results at each generation. In this case, the increased complexity of designing a chip to use the multi-generation sweep architecture will probably be more than offset by the additional I/O efficiency.

An interesting extension currently being investigated is the I/O overhead associated with simulating neural net computations.

	array sweep	n/2-gen.	multi-generation	
n	Ψ	Ψ	g_{min}	Ψ
1	4	4	1	4
2	3	—	1	3
3	2.67	3	1	2.67
4	2.5	—	2	2
5	2.4	2.67	2	1.67
6	2.33	—	2	1.5
7	2.28	2.5	3	1.33
8	2.25	—	3	1.17
9	2.22	2.4	3	1.07
10	2.2	—	4	1
11	2.18	2.33	4	0.9
21	2.10	2.18	7	0.51
101	2.02	2.04	30	0.11
1001	2.00	2.00	294	0.01

Table 1. I/O overhead for various architectures.

6. REFERENCES

[Cook73] S. A. Cook, "An Observation on Time-Storage Tradeoffs," Proceedings of the 5th Annual ACM Symposium on the Theory of Computing, May 1973, pp. 29-33.

[Dewd85] A. K. Dewdney, "Computer Recreations," Scientific American, volume 252, number 5, May 1985, pp. 18-30.

[Grah89] M. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley (1989), Chapter 4.

[Hong81] J. W. Hong and H. T. Kung, "I/O Complexity: The Red-Blue Pebble Game," Proceedings of the 13th Annual ACM Symposium on the Theory of Computing, May 1981, pp. 326-333.

[Kuge87] S. D. Kugelmass, R. Squier, and K. Steiglitz, "Performance of VLSI Engines for Lattice Computations," Journal of Complex Systems, volume 1, number 5, October 1987, pp. 939-965.

[Mano88] S. Manohar, "Supercomputing with VLSI," Ph.D. Thesis, Brown University, 1988.

[Poun85] W. Poundstone, *The Recursive Universe*, Contemporary Books, Inc.: Chicago (1985).

[Sava87] J. E. Savage and J. S. Vitter, "Parallelism in Space-Time Trade-offs," Advances in Computing Research, volume 4, 1987, pp. 117-146.

[Toff87] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*, MIT Press: Cambridge, MA (1987).

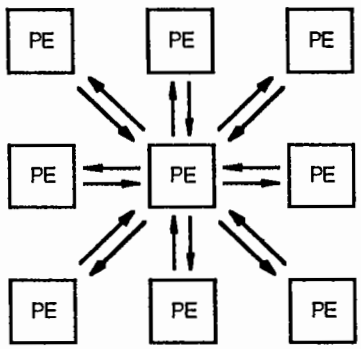

Figure 1. Inter-processor communication.

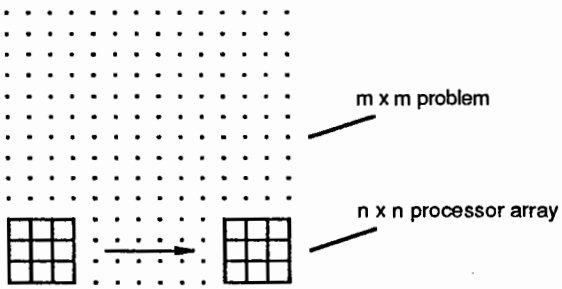

Figure 2. Tessellating the problem space.

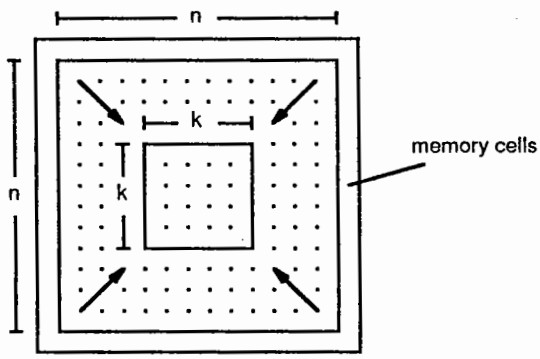

Figure 3. Multi-generation sweep architecture.

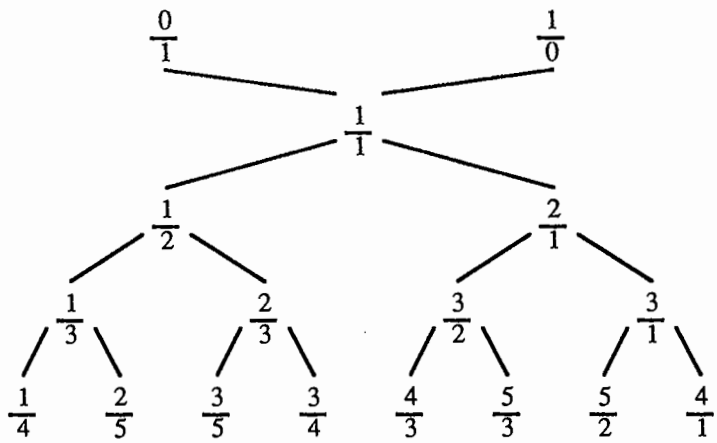

Figure 4. Top of the Stern-Brocot Tree.

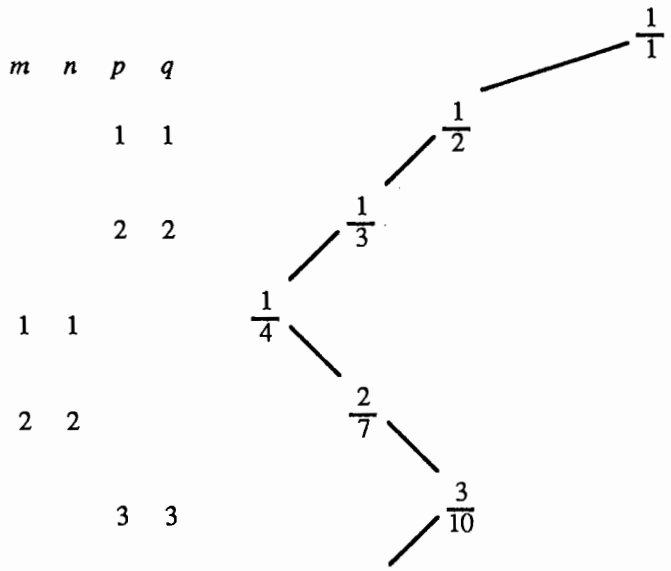

Figure 5. Stern-Brocot expansion of $1 - \sqrt{2}/2$. The table at the left indicates which subscript of which variables corresponds to each fraction of the tree.