I/O Overhead and Parallel VLSI Architectures for Lattice Computations

Mark H. Nodine, Daniel P. Lopresti, *Member, IEEE,* and Jeffrey S. Vitter, *Member, IEEE*

Abstract—**In this paper we introduce input/output (I/O) overhead ψ as a complexity measure for VLSI implementations of two-dimensional lattice computations of the type arising in the simulation of physical systems. We show by pebbling arguments that $\psi = \Omega(n^{-1})$ when there are n^2 processing elements available. If the results are required to be observed at every generation, and no on-chip storage is allowed, we show the lower bound is the constant 2. We then examine four VLSI architectures and show that one of them, the multigeneration sweep architecture, also has I/O overhead proportional to n^{-1}. We compare the constants of proportionality between the lower bound and the architecture. Finally, we prove a closed-form for the discrete minimization equation giving the optimal number of generations to compute for the multigeneration sweep architecture.**

Index Terms—**Discrete minimization, input/output complexity, lattice computations, pebbling, VLSI.**

I. 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. Evolution of a cellular automaton takes place in discrete time steps called *generations.* 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 *nine-cell lattice computations.* The algorithms and lower bounds we develop can be extended easily to other interconnection patterns.

A famous example of a cellular automaton is the game of "Life," which was introduced by John Conway in 1969. Despite the apparent simplicity in having purely local rules govern the time evolution of the system, Life exhibits complex behavior and in fact possesses the same computational power as a Turing machine [2]. It also admits a universal constructor to allow self-replicating structures [8].

Manuscript received March 21, 1989; revised February 12, 1990. This work was supported in part by an NSF Presidential Young Investigator Award CCR-8451390 with matching funds from IBM, by NSF Research Grant DCR-8403613, by NSF Research Grant MPI8710745, by an NCR cooperative research and development agreement, by an IBM departmental grant, and by ONR Grant N00014-83-C-K-0146, ARPA Order 6320.

The authors are with the Department of Computer Science, Brown University, Providence, RI 02912.

IEEE Log Number 9100996.

Lattice computations have important applications in physical simulations. Examples of simulations that use such automata are two-dimensional lattice gas computations [5], diffusion-limited aggregation [7], two-dimensional diffusion, fluid dynamics, spin glasses, and ballistics [10]. A VLSI circuit to solve the Poisson equation has been implemented using lattice computation techniques [6].

Highly local data movement coupled with a tremendous potential for parallelism would seem to make these problems an ideal match for VLSI. Kugelmass *et al.* showed, however, that VLSI-based machines performing such computations are severely constrained by input/output (I/O) requirements [5]. Using a simpler new argument, we improve by a constant factor the theoretical lower bound on I/O that can be derived from their work. We then present and analyze four VLSI architectures within this framework. One of these, the multigeneration sweep architecture, is optimal in that it meets the lower bound asymptotically 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 the *I/O overhead* ψ, which we define as

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

where

I = number of input operations,
O = number of output operations,
C = number of cells computed,
g = number of generations computed.

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

In Section II, we give a lower bound argument based on pebbling to show that the I/O overhead for nine-cell lattice computations is at least $1/\sqrt{2S}$, where S is the amount of on-chip storage. In Section III, we give several architectures for lattice computations, one of which, the multigeneration sweep architecture, is within a constant factor of the lower bound. Section IV proves a closed form for the optimal number of generations to compute in the multigeneration sweep architecture before viewing the results. Our conclusions are given in Section V.

II. Lower Bounds on I/O Overhead

Graph pebbling is a powerful technique for proving computational lower bounds [1], [4], [5], [9]. For nine-cell lattice computations, a general result in Kugelmass *et al.* 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) has α bits of local storage. Their result can be improved by factor of $\sqrt{2}$ by taking advantage of the connectivity of nine-cell lattice computations instead of the five-cell computations they consider.

In this section, we improve upon this lower bound by a simpler argument. We make no assumptions about when results of the computation are viewed. If the states of all cells must be examined after every generation, we show that $\psi \geq 2$. This last result implies that VLSI implementations are more effectively used for analyzing the long-term behavior of lattice computations.

Our arguments, like those of Kugelmass *et al.,* are based on the *red–blue pebble game* [4]. 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 modeling a computation. Each vertex in V corresponds to a result that is computed at some stage of the computation. An edge $e = (v_1, v_2)$ in the graph indicates that the result corresponding to v_1 is used to compute the result for vertex v_2. Pebbles represent memory locations. A red pebble represents a result that is in memory on the chip, and a blue pebble represents a result that is off-chip. 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 that has a blue pebble.
3) A blue pebble may be placed on any vertex that 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 reuse the memory). 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 *lattice inputs,* and those that have no successors are the *lattice outputs.* The initial configuration has blue pebbles on all the lattice inputs.

Definition: A *pebbling* of a computation graph is a sequence of ordered pairs, (n, v), where $n \in \{1, \cdots, 4\}$ 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 lattice outputs having blue pebbles. An *S-pebbling* is a pebbling with the restriction that at most S red pebbles are on vertices of the lattice at any one time.

We define the computation graph $G_g = (V_g, E_g)$ for a lattice computation of g generations as follows: Choose m such that the size of the problem during all g generations can be bounded by an $m \times m$ array of cells. The vertex set is

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

and the edge set is

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

The cells on the border have either three or five predecessors, rather than eight. 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.

We first present a few lemmas leading up to our main lower bound result. The basic strategy is to bound the number of computations that can be done with no inputs starting from any configuration, and then compute the minimum number of inputs that must be done in order for every node to have been red-pebbled. We start by showing that we can consider only pebblings that red-pebble the generations in order.

Lemma 1: Let us consider a pebbling game in which an initial configuration of $S < m$ red pebbles is given, and only rules 1 and 4 are allowed for pebbling moves (that is, no I/O operations are allowed). Then there is 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, since the first operation in such a configuration will always be an application of rule 1, giving a configuration of $S - 1$ red pebbles. We show that there is a schedule that pebbles earlier generations before later ones that results in all the vertices in R being pebbled. Let $g + 1$ be the first generation containing some node in R. For $S < m$, there is a pebble on some vertex v on generation g that 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 that 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 that maximizes the number of nodes pebbled can be redone in an order that pebbles earlier generations before later ones. ■

Next, we show that we can group the inputs into phases, suffering at most a penalty of requiring twice as many red-pebbles.

Lemma 2: Any red–blue S-pebbling P with T input operations can be simulated by some $2S$-pebbling P' of the following type.

a) P' can be divided into *phases* such that in each phase, all inputs are done consecutively at the beginning of the phase.
b) P' has $\lceil T/S \rceil$ phases, each containing at most S input operations.
c) The individual input operations in P' are a subsequence of those in P.

Proof: This can be proved by an easy simulation of the original pebbling. Let $I_1, \cdots, I_T$ be the inputs in pebbling P. We transform P into P' by moving $I_{kS+2}, \cdots, I_{kS+S}$ to follow immediately after I_{kS+1}, for $k = 1 \cdots \lceil T/S \rceil$, eliminating those inputs that would be to a location that already has a red pebble on it. We then maintain the invariant that at the beginning of each phase k in P', the same nodes in the computation graph have red pebbles on them as in P just prior to doing I_{kS}. This can be done inductively. It is initially true since no nodes have red pebbles on them in either pebbling. Assume that P and P' are in the same configuration just prior to doing I_{kS}. Then P' must have $r < S$ pebbles on the graph, so it has at least S available to do all the inputs for the phase. After doing the inputs, P and P' both have $S - r$ pebbles left, so P' can exactly simulate P with the exception that it need not put a red pebble on any vertex already containing one. At the end of the phase, P' removes red pebbles from all vertices not covered with red pebbles by P, and the induction hypothesis is maintained. This proves Lemma 2. ■

Finally, we show how many nodes in the lattice can be red-pebbled using only internal computations.

Lemma 3: The maximum number of nodes in the lattice that can be pebbled with S red pebbles using rule 4 alone is $\left(S^{3/2} - S\right)/2$, assuming $S < m$.

Proof: Let us consider any schedule that maximizes the number of nodes pebbled using rule 4 only. By Lemma 1 we can assume that the schedule proceeds generation by generation, for generations $0, 1, \cdots, g-1$. Each generation will be pebbled in parallel. Let $A_i \geq 0$ be the number of red pebbles moved while pebbling generation $i+1$ from generation i, and let $B_i \geq 0$ be the number of red 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. The term A_i is maximized if all S pebbles are on generation 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 $(\sqrt{S}-1)^2$. Thus, for any value of i we have

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

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

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

We can get an upper bound on $\sum_i A_i$, the number of nodes pebbled, by maximizing $\sum_i A_i$ subject to constraints (2), (3), and (4), where A_i and B_i are allowed to be real nonnegative numbers. Thus, $\sum_i A_i$ is strictly bounded by a hypothetical case with $A_i = (\sqrt{S}-1)^2$ for $S/B_i = S/(2(\sqrt{S}-1))$ values of i. This gives us the bound $\sum_i A_i \leq \left(S^{3/2} - S\right)/2$, which proves the lemma. ■

We are now ready to prove our main lower bound result.

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

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

Proof: Lemma 2 showed that every S-pebbling strategy with T I/O's can be simulated efficiently by a $2S$-pebbling of roughly T/S phases in which at most S inputs occur at the beginning of each phase. We use Lemmas 1 and 3 to bound the number of internal computations that can be done on the lattice during one phase. Since we know how many nodes there are in the lattice that must be red-pebbled, we get a lower bound on the number of inputs required.

From Lemma 3, 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 a total of gm^2 lattice elements must be red-pebbled, the number of phases is at least

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

Here, we have used the facts that $m > S$ and $g \geq 2$. Since each S-pebbling of the lattice with T I/O's has a corresponding $2S$-pebbling with $\lceil T/S \rceil$ phases, 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 the result. ■

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

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

What happens if we insist that we want to see the results of the calculation after every generation? This corresponding lower bound is easy to determine.

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

$$\psi_{\text{every gen.}} \geq \frac{2m^2}{m^2} = 2, \tag{6}$$

as claimed. ■

So any lattice computation in which we wish to examine all the cells at each generation will have an I/O overhead $\psi \geq 2$, regardless of the number of generations computed, assuming no values are retained on the chip between generations. The bound in (6) is clearly worse than that of (5), where the cells do not have to be examined at each generation. VLSI chips to do lattice computations seem to be best suited for studying the asymptotic behavior of initial configurations.

III. Architectural Approaches

In this section, we present and analyze four parallel VLSI architectures for lattice computations. Each one uses 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.

A. 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 Fig. 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 storage location to hold its current state. The algorithm used is

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

It should be noted that, although the problem initially fits entirely within the array of processors, there is no guarantee that it will continue to do so as $G' \to \infty$. Conservatively, we must take $G = \lfloor (n-m)/2 \rfloor$. After this point, we are forced to stop and output the results, since we can no longer be sure that we will get the same result as would be obtained in an infinite grid. A short deviation using (1) shows that the I/O overhead for this architecture is

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

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

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

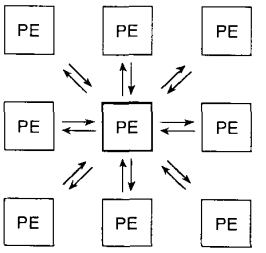


Fig. 1. Interprocessor communication.

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 $n \times n$.

B. Array Sweep Architecture

In this architecture, described by Toffoli [10], 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 off-chip locations. In this manner, the processors wind their way through the problem space, tessellating the computation as demonstrated in Fig. 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 one 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, (6) correctly predicts that $\psi \geq 2$. 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 reread immediately. With this change the I/O overhead is reduced to

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

C. $n/2$-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 $n+2$

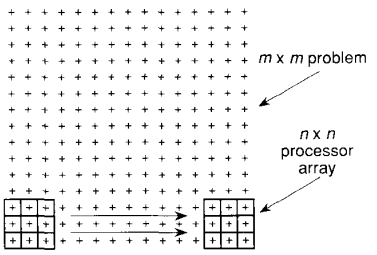


Fig. 2. Tessellating the problem space.

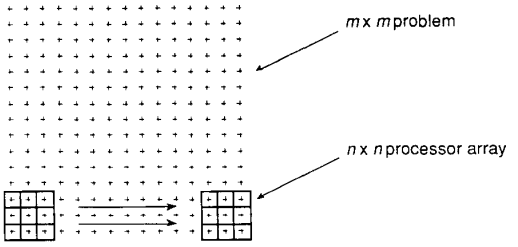


Fig. 3. Multigeneration sweep architecture.

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.

D. Multigeneration 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 Fig. 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.

Toffoli and Margolus mention this technique under the name "scooping," but do not analyze it from the standpoint of I/O efficiency [7], [10].

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

$$\psi_{mgs} = \frac{2(n-g+2)}{g(n-2g+2)}. \tag{7}$$

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 discrepancy 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 that minimizes the expression for I/O overhead. By setting the partial derivative of ψ with respect to g equal to zero, we get

$$n + 2 \pm \frac{\sqrt{2}}{2}\sqrt{n^2+4n+4} = \left(1 \pm \frac{\sqrt{2}}{2}\right)(n+2),$$

of which the negative root gives the minimum. Thus,

$$\tilde{g}_{\min} = \left(1 - \frac{\sqrt{2}}{2}\right)(n+2) \tag{8}$$

The tilde superscript is to indicate that this quantity is always an integer multiple of an irrational number and is therefore always nonintegral. What we want is the discrete minimum for (7). We show in Section IV that taking the nearest integer of (8) gives the discrete minimum:

$$g_{\min} = \left\lfloor \tilde{g}_{\min} + \frac{1}{2} \right\rfloor.$$

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 about 23 of the best possible I/O overhead.

IV. Proof of Validity of Discrete Minimization Formula

In the Section III, we found that the I/O overhead for the multigeneration sweep architecture was

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

From (8), we have the value of g that minimizes ψ:

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

where

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

Let $R(x)$ be the rounding function, defined by

$$R(x) = \lfloor x + 1/2 \rfloor. \tag{9}$$

We want to prove the assertion that taking the nearest integer function of $\tilde{g}_{\min}$ provides an integer value at which the minimum occurs, in other words

$$g_{\min}(n) = R(\tilde{g}_{\min}(n)) = R((n+2)\xi). \tag{10}$$

This formula could produce the wrong result if the fractional part of $(n+2)\xi$ falls in some "window of vulnerability" around $1/2$, so that the actual discrete minimum occurs at one integer value, but R rounds to the other integer value. The basic idea behind the proof of validity of (10) is to compute the window of vulnerability and then show that there is no smallest value of n for which the fractional part falls within that window.

A. The Stern–Brocot Tree

The proofs of the theorem depend heavily on the properties of the Stern–Brocot (S-B) tree [3]. 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 $(m+m')/(n+n')$. Fig. 4 shows the top part of the tree. The fractions that 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 that 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 b is an irrational number, then the fractions generated in its S-B expansion are the simplest rational approximations to b in the sense that if m/n is an approximation to b, there exists an S-B expansion m'/n' such that $m' \le m$, $n' \le n$, and m'/n' is between m/n and b.

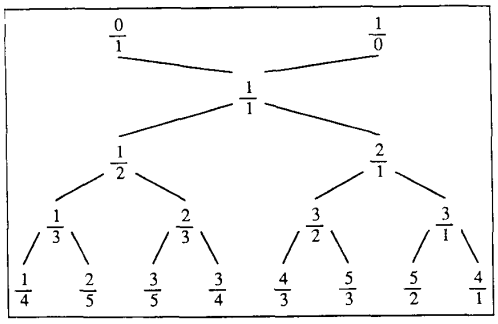


Fig. 4. Top of the Stern–Brocot tree.

B. The Lemmas

These are the lemmas which lead up to the main theorem of this section. The first lemma establishes that $\psi(n,g)$ is asymmetric about its minimum.

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

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

Then $h(n,\delta) > 0$ for all integer $n > 0$ and $0 < \delta \le 1/2$.

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

$$h(n,\delta) = \frac{16\delta^3}{\left((n+2)^2(\sqrt{2}-1)^2 - 4\delta^2\right)\left((n+2)^2(\sqrt{2}-1)^2 - 2\delta^2\right)}.$$

We can set the partial derivative with respect to δ equal to 0 to find the minima and maxima. When we do this, we find a double root at zero, two imaginary roots, and roots at

$$\delta_{\pm} = \pm\frac{1}{4}\sqrt{\sqrt{33}-3}\left(2-\sqrt{2}\right)(n+2) \approx \pm 0.243(n+2),$$

of which we take the positive root, since we are only interested in $\delta > 0$. $\delta_+ > 1/2$ for all $n > 0$. It also represents a maximum in the curve, since

$$\frac{\partial^2 h(n,\delta_+)}{\partial \delta^2} = -3\sqrt{\sqrt{33}-3}\left(67\sqrt{33}+385\right)\left(7\sqrt{2}+10\right) \cdot (n+2)^{-3} \approx -76142(n+2)^{-3}$$

which is negative for all $n > 0$. Since for all $n > 0$, $h(n,0) = 0$ and the first local extremum in the curve is a maximum at $h(n,\delta)$ with $\delta > 1/2$, we conclude that $h(n,\delta) > 0$ for all $0 < \delta \le 1/2$. ■

Now we can show that the nearest integer can only produce an incorrect result if the fractional part of ξn is slightly greater than $1/2$ for some n.

Lemma 6: Formula (10) for $g_{\min}$ can only produce an incorrect result if $\mathrm{Frac}(\xi n) = 1/2 + \varepsilon$, for some $\varepsilon > 0$.

Proof: Fig. 5 shows the behavior of ψ about its minimum for $m < \tilde{g}_{\min} < m+1$. If $\psi(n,g)$ were exactly symmetric about its minimum, then $R(\tilde{g}_{\min}(n))$ would always produce the discrete minimum of $\psi(n,g)$, where R is as defined in (9). From Lemma 5, we know that $\psi(n, \tilde{g}_{\min} + \delta) > \psi(n, \tilde{g}_{\min} - \delta)$, so that if the minimum falls at a fractional

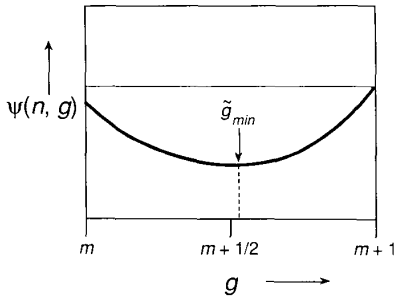


Fig. 5. Behavior of ψ near its minimum.

part that is $1/2 + \varepsilon$, $\psi(n, m)$ may be less than $\psi(n, m+1)$. No similar problem exists if $\varepsilon < 0$. ■

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

Lemma 7: If m is any integer, then (10) will produce the wrong result if

$$m + \frac{1}{2} < \tilde{g}_{\min}(n) < m + \frac{1}{2} + \varepsilon(m),$$

for some n, where

$$\varepsilon(m) = \frac{1}{2}\left(\sqrt{2} - 1\right)\left(\sqrt{4m^2 + 4m + 2} - (2m+1)\right). \tag{11}$$

Proof: A window of vulnerability exists when $\tilde{g}_{\min}(n)$ 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

$$\tilde{g}_{\min}(n) = (n+2)\xi = m + \frac{1}{2} + \varepsilon,$$

or equivalently

$$n = \left(2 + \sqrt{2}\right)\left(m + \frac{1}{2} + \varepsilon\right). \tag{12}$$

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

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

Plugging in the value of n from (12) gives

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

The values of ε solving this equation are

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

The negative root is negative for all $m > 0$, 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

$$\varepsilon(m) = \frac{1}{8}\left(\sqrt{2} - 1\right)m^{-1} - \frac{1}{16}\left(\sqrt{2} - 1\right)m^{-2} + O(m^{-3}). \tag{13}$$

The windows of vulnerability thus start out small and shrink asymptotically to zero.

Now, we adopt a different tack to find out which multiples of ξ could possibly have fractional parts that fall within the windows of vulnerability. The next lemma gives the S-B expansion of ξ.

Lemma 8: The S-B expansion for ξ is $\mathrm{L(LLRR)}^{\infty}$.

Proof: It is clear that the expansion begins with L, since $\xi < 1$. Let those fractions that 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 that 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 Fig. 6). 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:

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

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

which yield

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

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

Similarly, we get

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

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

Let us define

$$\Delta_k = \xi n_k - m_k \tag{14}$$

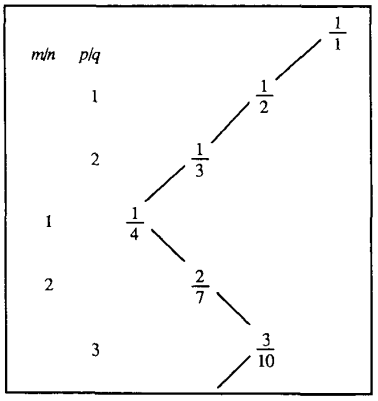


Fig. 6. Stern–Brocot expansion of $1-\sqrt{2}/2$. The table at the left indicates which subscript of which variables corresponds to each fraction of the tree. For example, $p_2/q_2 = 1/3$

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 9: The fractions m_k/n_k converge monotonically to ξ.

Proof: It should be noted that monotonic convergence is not an automatic property of S-B expansions of numbers. Let Δ_k be as in (14). Let δ_k be defined by $\delta_k = \xi - m_k/n_k$. Then $\delta_k = \Delta_k/n_k$. We get

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

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

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

This ratio 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 b_k for which the fractional part $b_k\xi$ approaches $1/2$ from the top more closely than $n\xi$ for any other $n < b_k$. It is easy to see from the generating functions that when k is odd, m_k is odd and n_k is even. If we define $\Delta_k = \xi n_k - m_k$, then

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

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

$$b_k = \frac{n_{2k-1}}{2} = \frac{1}{4}\left(\left(4+3\sqrt{2}\right)\left(3+2\sqrt{2}\right)^{k-1} + \left(4-3\sqrt{2}\right)\cdot\left(3-2\sqrt{2}\right)^{k-1}\right), \quad \text{for } k \geq 1, \tag{15}$$

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

Lemma 10: Let us define

$$\chi(n) = \text{Frac}(\xi n - 1/2), \tag{16}$$

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 $1/2$). The values of b_k defined in (15) 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 that 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 $n_j \leq 2n$, that is at least as good as $(2m+1)/2n$. Also, $2n < 2b_k = n_{2k-1}$, 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 9. Thus, the assumption that there was such a value of n was incorrect, and the lemma is proved. ■

The only remaining fact needed to prove the theorem is 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 11: 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 10 that

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

Plugging into (11), we get

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

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

$$\begin{aligned}\varepsilon(m(b_k)) &= \frac{1}{4}\left(\sqrt{2}-1\right)\left(\left(1+\sqrt{2}\right)^{2k-1} - \left(1-\sqrt{2}\right)^{2k-1}\right. \\ &\quad \left. - \left(1+\sqrt{2}\right)^{2k-1} - \left(1-\sqrt{2}\right)^{2k-1}\right) \\ &= \frac{1}{2}\left(3-2\sqrt{2}\right)^{k}.\end{aligned}$$

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

C. The Theorem

We are now ready to prove the theorem.

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

Proof: The values of $\tilde{g}_{\min}$ comprise all the multiples of an irrational number, ξ. The nearest integer function R will give the wrong answer only if the fractional part of that multiple of ξ is sufficiently close to $1/2$ that R will round it one way, but the minimum would occur by rounding it the other. Lemma 6 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 7 computes the window of vulnerability $\varepsilon(m)$, that is, the maximum amount by which the fractional part of a failing value of ξn can exceed $1/2$. This window of vulnerability is a monotonically decreasing function of m. Let N be the set of all n for which $R(\tilde{g}_{\min}(n)) \neq g_{\min}(n)$. Then if $N \neq \varnothing$, it has a smallest element n'. Define $\chi(n)$ as in (16). For any $n'' < n'$, $\chi(n') < \chi(n'')$ 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 10 proved the explicit form of those numbers b_k. Finally, Lemma 11 demonstrated that each b_k falls exactly at the edge of a window of vulnerability, which means that

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

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

D. The Significance of this Result

Proving an exact closed form for the optimal number of generations to compute is not strictly necessary to show that the architecture meets the lower bound asymptotically to within a constant factor. However, from a practical standpoint, it is nice to have a closed-form formula that tells how many generations to compute for a particular value of n. Furthermore, the fact that there exists a closed-form equation is quite unexpected, as we hope to show in this subsection. The technique of using a Stern–Brocot tree to prove such an exact result is novel.

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 (13)

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

Taking logs, we get

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

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.

V. Conclusions

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

Do the asymptotic differences exhibited in this paper have any practical significance? Table I indicates values of ψ for three of our schemes. The multigeneration sweep architecture is noticeably superior even when n is relatively small. A value of 31 is achievable with current technology, resulting in an improvement of almost a factor of 6 in I/O performance; using wafer technology, a chip with 100^2 processors is conceivable, in which case there is more than a factor of 18 improvement in the I/O overhead of the multigeneration sweep architecture over 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 proposed by other researchers is almost independent of n. In a problem like this, for which the limiting factor is I/O, we have the undesirable result that packing more processors onto a chip does not help much.

We conclude that a VLSI implementation of a two-dimensional lattice computation does not need to be severely

TABLE I
I/O OVERHEAD FOR VARIOUS ARCHITECTURES

n	array sweep ψ	$n/2$-gen. ψ	multi-generation g_{min}	multi-generation 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
31	2.06	2.12	10	0.35
101	2.02	2.04	30	0.11
1001	2.00	2.00	294	0.01

restricted by its I/O performance so long as viewing the results after each generation is unnecessary. In this case, the increased complexity of designing a chip to use the multigeneration sweep architecture may be more than offset by its I/O efficiency.

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

REFERENCES

[1] S. A. Cook, "An observation on time-storage tradeoffs," in *Proc. 5th Annu. ACM Symp. Theory Comput.*, May 1973, pp. 29–33.

[2] A. K. Dewdney, "Computer recreations," *Scientif. Amer.*, vol. 252, no. 5, pp. 18–30, May 1985.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics.* Reading, MA: Addison-Wesley, 1989, ch. 4.

[4] J. W. Hong and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, May 1981, pp. 326–333.

[5] S. D. Kugelmass, R. Squier, and K. Steiglitz, "Performance of VLSI engines for lattice computations," *Complex Syst.*, vol. 1, no. 5, pp. 939–965, Oct. 1987.

[6] S. Manohar, "Superconducting with VLSI," Ph.D. dissertation, Brown Univ., 1988.

[7] N. Margolus and T. Toffoli, "Cellular automata machines," *Complex Syst.*, vol. 1, no. 5, pp. 967–993, Oct. 1987.

[8] W. Poundstone, *The Recursive Universe.* Chicago, IL: Contemporary Books, 1985.

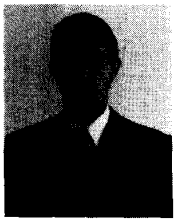
[9] J. E. Savage and J. S. Vitter, "Parallelism in space-time trade-offs," *Advances Comput. Res.*, vol. 4, pp. 117–146, 1987.

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

Mark H. Nodine received the B.A. degree in mathematics (magna cum laude with departmental honors), the B.S. degree in chemistry and physics (magna cum laude with departmental honors in chemistry) from Tulane University, New Orleans, LA, in 1978, and the S.M. degree in chemistry from the Massachusetts Institute of Technology, Cambridge, in 1982.

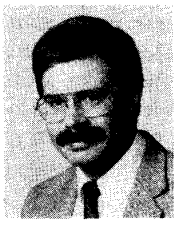
He spent 1982 to 1983 with Schlumberger working on computer-aided-design software and from 1983 to 1988 at Bolt Beranek and Newman working on authoring and network monitoring systems. He received the S.M. degree in computer science from Harvard University, Cambridge, MA in 1986, and again from Brown University, Providence, RI, in 1988. He is currently a candidate for the degree of Ph.D. at Brown University, Providence, RI. His research interests include the design and analysis of combinatorial algorithms; efficient I/O algorithms for external sorting, database applications, computational geometry, and neural networks; and computer graphics.

Mr. Nodine has been a student member of the Association for Computing Machinery since 1989.



Daniel P. Lopresti (M'87) received the A.B. degree in mathematics from Dartmouth College, Hanover, NH, in 1982, and the M.A. and Ph.D. degrees in computer science from Princeton University, Princeton, NJ, in 1984 and 1987, respectively.

He is an Assistant Professor in the Department of Computer Science, Brown University, Providence, RI. His research interests include parallel architectures, VLSI CAD, and computational aspects of molecular biology. He is also a consultant for the Supercomputing Research Center, Bowie, MD.



Jeffrey S. Vitter (S'80–M'81) was born in New Orleans, LA, on November 13, 1955. He received the B.S. degree in mathematics with highest honors from the University of Notre Dame in 1977, and the Ph.D. degree in computer science from Stanford University in 1980.

He joined the faculty of Brown University in 1980, where he is currently Professor of Computer Science. Prior to finishing graduate school, he worked as a Computer Performance Analyst at Standard Oil Co. of California and as a Research Assistant and teaching fellow in the Department of Computer Science at Stanford University. He was on sabbatical in 1986 as a member of the Mathematical Sciences Research Institute in Berkeley and in 1986–1987 as a member of INRIA in Rocquencourt, France, and as a Visiting Professor at Ecole Normale Supérieure in Paris. He is currently an associate member of the Center of Excellence in Space Data and Information Sciences. His research interests include mathematical analysis of algorithms, computational complexity, parallel algorithms, I/O efficiency, machine learning, computational geometry, and incremental computation. He has written numerous articles and has been a frequent lecturer, guest editor, conference program committee member, and consultant. He has coauthored the book *Design and Analysis of Coalesced Hashing* (Oxford University Press, 1987) and is coholder of a patent in the area of external sorting.

Dr. Vitter is an IBM Faculty Development Awardee, an NSF Presidential Young Investigator, and a Guggenheim Fellow. He serves on the editorial board of *SIAM Journal on Computing*, *Communications of the ACM*, and *Mathematical Systems Theory: An International Journal on Mathematical Computing Theory*. His professional memberships include IEEE Computer Society, The Association for Computing Machinery, and Sigma Xi.