LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-302

A SURVEY OF ALGORITHMS FOR INTEGRATING WAFER-SCALE SYSTOLIC ARRAYS

TOM LEIGHTON
CHARLES LEISERSON

MAY 1986

545 TECHNOLOGY SQUARE. CAMBRIDGE. MASSACHUSETTS 02139

A Survey of Algorithms for Integrating Wafer-Scale Systolic Arrays

Tom Leighton

Mathematics Department and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Charles E. Leiserson

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract— VLSI technologists are fast developing *wafer-scale integration.* Rather than partitioning a silicon wafer into chips as is usually done, the idea behind wafer-scale integration is to assemble an entire system (or network of chips) on a single wafer, thus avoiding the costs and performance loss associated with individual packaging of chips. A major problem with assembling a large system of microprocessors on a single wafer, however, is that some of the processors, or *cells,* on the wafer are likely to be defective. This paper surveys practical procedures for integrating "around" such faults. The procedures are designed to minimize the length of the longest wire in the system, thus minimizing the communication time between cells. Although the underlying network problems are NP-complete, all the procedures can be proved reliable by assuming a probabilistic model of cell failure.

Key Words: channel width, fault-tolerant systems, matching, probabilistic analysis, spanning tree, systolic arrays, travelling salesman problem, tree of meshes, VLSI, wafer-scale integration, wire length.

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80 C 0622 and in part by the Air Force under Contracts OSR-82 0326 and OSR 86 0076. Tom Leighton and Charles Leiserson are both supported in part by NSF Presidential Young Investigator Awards. Portions of this work are based on the paper "Wafer-scale integration of systolic arrays" by Leighton and Leiserson, which appeared in *IEEE Transactions on Computers,* Vol. C-34, No. 5, pp. 448-461, May 1985. (C) 1985 IEEE.

1. Introduction

VLSI technologists are fast developing *wafer-scale integration* [25]. Rather than partitioning a silicon wafer into chips as is usually done, the idea behind wafer-scale integration is to assemble an entire system (or network of chips) on a single wafer, thus avoiding the costs and performance loss associated with individual packaging of chips. A major problem with assembling a large system of microprocessors on a single wafer, however, is that some of the processors, or *cells*, on the wafer are likely to be defective, or *dead*. In this paper, we survey algorithms for constructing systolic arrays from the *live* cells of a silicon wafer.

Laser-programming the interconnect of a wafer is one promising means of achieving wafer-scale integration. This technology was pioneered at IBM [21] and pursued in the direction of wafer-scale integration at MIT Lincoln Laboratory [25]. Figure 1 shows a scanning electron microscope photograph of a portion of a wafer with programmable interconnect. Laser welds can be made between two layers of metal, and by using the beam at somewhat higher power, wires can be cut. Defective components can thus be avoided by programming connections between only the good components.

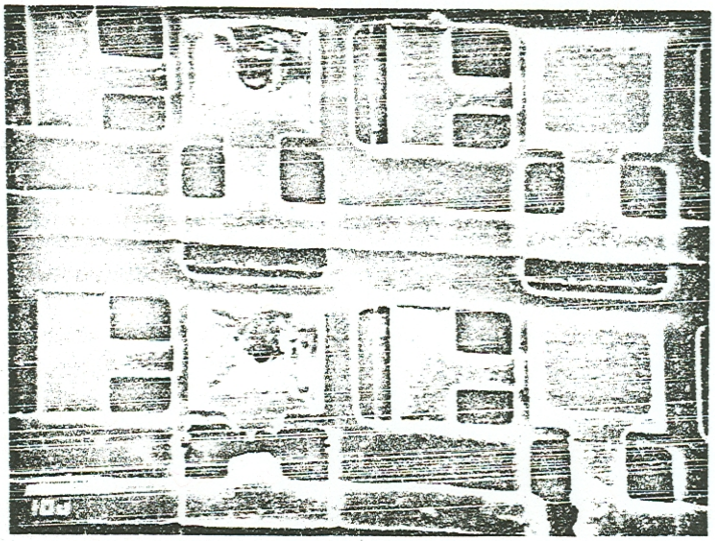

Figure 1. A close-up of laser-programmable interconnect.

Figure 2 shows a typical organization of a wafer-scale system with programmable interconnections. The components are organized as a matrix of cells, and between the cells are channels through which the interconnect runs. Figure 3 is a close-up of the channel structure. At the intersection of a horizontal and vertical channel, laser-programmable connections can make a horizontal and a vertical wire electrically equivalent. Between two cells, connections can be made from the wires in the channel to the inputs and outputs of the two cells. Given that the interconnect is programmable, we shall adopt a usage of the term "wire" to mean an electrically equivalent portion of the programmable interconnect.

Systolic arrays [12, 13, 20] are a desirable architecture for VLSI because all communication is between nearest neighbors. A realization of a systolic array as a wafer-scale system may lose this advantage if all nearest neighbors of a processor are dead, however, because a long wire may

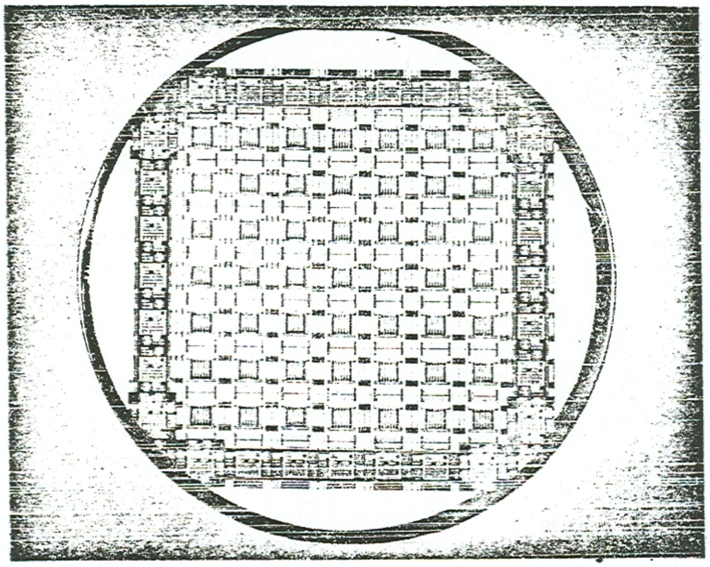

Figure 2. A wafer-scale system of cells and programmable interconnect.

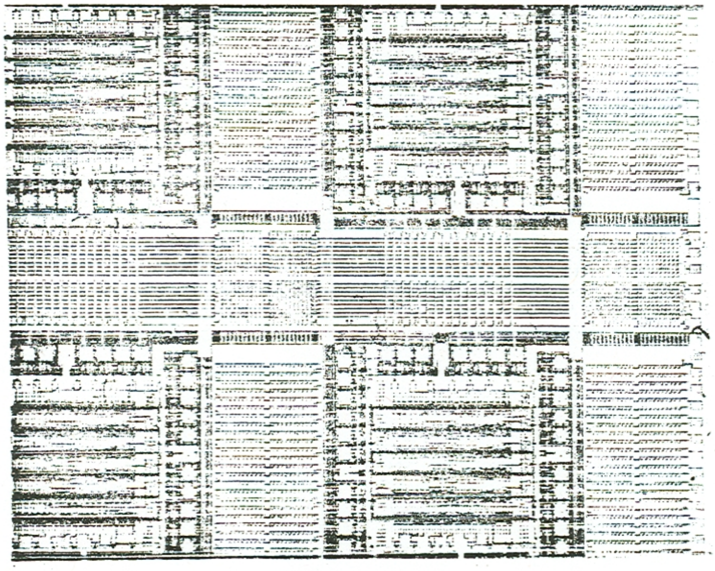

Figure 3. The channel structure of a wafer-scale system.

be needed to connect electrically-adjacent processors. In general, the longest interconnection between processors is the communication bottleneck of the system. Of the many possible ways in which the *live* cells on a wafer can be connected to form a systolic array, therefore, the one that minimizes the length of the longest wire is most desirable.

To illustrate the subtleties inherent in configuring systolic arrays, consider the problem of constructing a linear (i.e., one-dimensional) array using all of the live cells in an N-cell wafer. Unfortunately, if we wish to minimize the length of the longest wire, the problem is NP-complete [10]. Even more discouraging is that there are some arrangements of live and dead cells for which even the optimal linear array has unacceptably long wires. Thus optimal solutions—even if they could be found quickly—are not always practical.

By assuming a probabilistic model of cell failure, however, many positive results can be proved. For example, Figure 4 illustrates a possible solution to the problem of connecting the live cells of a wafer into a linear systolic array. The live cells, which are denoted by small squares, are connected together, one after another, in a snake-like pattern. Dead cells, denoted by X's, are skipped over. With probability at least $1 - O(1/N)$, the length of the longest wire is $O(\lg N)$, where N is the number of cells in the wafer and where each cell independently has a 50 percent chance of failure.*

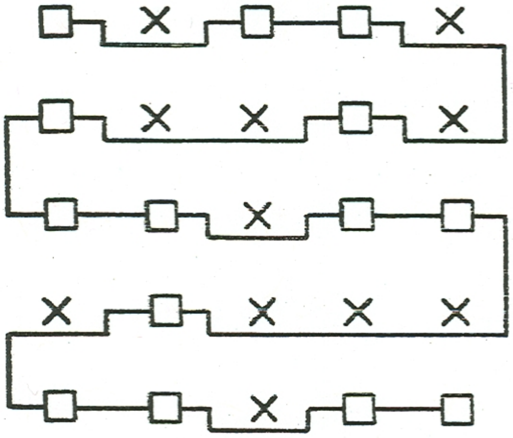

Figure 4. A simple means of constructing a linear systolic array from the live cells on a wafer.

This bound comes from the observation that the length of the longest wire that connects two cells in the array is just the length of the longest sequence of dead cells in the snake-like string. For a given set of k cells, the probability that all are dead is $1/2^k$, and thus the probability that any set of $2\lg N$ cells are dead is $1/N^2$. Since there are less than N sets of $2\lg N$ consecutive cells, the chances are thus less than one in N of having to skip more than $2\lg N$ cells in the entire snake-like path of length N. Hence the maximum wire length is $O(\lg N)$ with probability at least $1 - O(1/N)$.

To say that "with probability $1 - O(1/N)$ the maximum wire length is $O(\lg N)$," is a substantially stronger statement than saying that the expected maximum wire length is $O(\lg N)$. This is because no wire can ever have length greater than $O(\sqrt{N})$, even in the worst case. Hence the expected maximum wire length is at most

$$(1 - O(1/N)) \cdot O(\lg N) + O(1/N) \cdot O(\sqrt{N}) = O(\lg N) \quad .$$

Moreover, the chances that the maximum wire length is much greater than $O(\lg N)$ are miniscule. In particular, the probability of having to skip more than $k\lg N$ dead cells at a fixed point in the snake-like path is less than one in N^k. Hence, every wire has length at most $k\lg N$ with probability at least $1 - 1/N^{k-1}$.

*Here and throughout the paper, we use $O(f(N))$ to denote a function that is bounded above by $cf(N)$ for a fixed constant c and all sufficiently large N. We also use $\Omega(f(N))$ to denote a function that is bounded below by $cf(N)$, and $\Theta(f(N))$ to denote a function that is bounded above by $c_1 f(N)$ and below by $c_2 f(N)$ for some fixed constants c, c_1 and c_2, and all sufficiently large N. We also use $\lg N$ to denote $\log_2 N$, $\lg^2 N$ to denote $(\lg N)^2$, and $\lg\lg^2 N$ to denote $(\lg\lg N)^2$. Lastly, $\lfloor x \rfloor$ denotes the largest integer less than or equal to x, and $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

This paper presents a survey of algorithms for realizing one- and two-dimensional systolic arrays as wafer-scale systems. Unlike many of the heuristics in the literature, the algorithms here have all been theoretically analyzed, and bounds on their quality have been mathematically proved. The analyses make the assumption that each cell fails independently with probability p, and for simplicity, we assume here that $p = \frac{1}{2}$. We also assume for ease of explication and analysis that the width of a cell and the width of a wire are each unity. A more complete discussion of the assumptions and their generalizations can be found in [17].

The algorithms are organized to aid an engineer in picking an algorithm for implementation. We try to present enough mathematics to aid his intuition, but we do not, for the most part, include the detailed combinatorial arguments appearing in the literature that substantiate the effectiveness of the algorithms. Since programming involves many more "real-world" constraints than can be considered in an algorithmic analysis, we expect that the engineer might choose a less effective algorithm, for example, if it is easier to code. The algorithms here constitute a menu of possibilities to stimulate an intelligent design decision.

The remainder of the paper is divided into four sections. Section 2 contains basic combinatorial facts underlying the probabilistic analyses used in the literature. Section 3 gives two algorithms for integrating linear arrays. The first algorithm connects all the live cells on a wafer, and the second achieves somewhat shorter maximum wire length by connecting only a large constant fraction of the live cells. Section 4 gives five algorithms for integrating two-dimensional arrays, and includes both worst-case and probabilistic bounds. Section 5 discusses provides a summary of the material covered in the paper and mentions some related work.

2. Combinatorial facts

In the introduction, we showed that with probability at least $1 - O(1/N)$, a sequence of N cells on a wafer contains no more than $O(\lg N)$ dead cells in a row. This kind of high probability analysis underlies most of the algorithms in this paper. We shall use the term *"high probability"* to mean *"with probability at least* $1 - O(1/N)$*,"* where N is the number of cells on a wafer. We now present some basic facts used in high probability analyses.

The first fact is the standard definition of independence.

Fact 1. *Let A and B be independent random variables. Then*

$$\Pr\{A \cap B\} = \Pr\{A\}\Pr\{B\}\,. \quad \blacksquare$$

The second fact bounds the probability of the union of two random events, even if the events are not independent.

Fact 2. *Let A and B be random variables. Then*

$$\Pr\{A \cup B\} \leq \Pr\{A\} + \Pr\{B\}\,.$$

Proof. This fact follows from the principle of inclusion and exclusion. We always have

$$\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\},$$

and since $\Pr\{A \cap B\} \geq 0$, the result follows. ∎

Fact 2 provides a weak bound if the probabilities involved are large. For example, if the probability of the individual events are each greater than $1/2$, the bound on their union is trivial. When the probabilities are small, however, the bound can be useful.

The next fact bounds a linear function with an exponential. It is most useful when x is near zero.

Fact 3. *For all x in the range $-\infty < x < \infty$, we have*

$$1 + x \leq e^x . \quad ∎$$

We now turn to combinatorial theorems that deal more directly with the statistics of faults on wafers. As was mentioned in the introduction, we shall typically assume that each cell on the wafer fails independently with probability $1/2$.

Fact 4. *With high probability, a given rectangular pattern of live and dead cells of size $2\lg N$ never appears on an N-cell wafer.*

Proof. The proof follows the analysis for the snake-like scheme in the introduction, which relies on Fact 2. The generalization from one- to two-dimensional regions is straightforward, as is the generalization from a pattern consisting solely of dead cells to an arbitrary pattern. ∎

Of course, Fact 4 does not imply that no pattern will occur, only that the probability that a *given* pattern occurs is low. It's like the lottery: somebody will win, but probably not you.

Remarkably, patterns of slightly less than half the size almost always appear on a wafer.

Fact 5. *With high probability, a given rectangular pattern of live and dead cells of size $\lg N - 2\lg\lg N$ appears somewhere on an N-cell wafer.*

Proof. Partition the wafer into $N/(\lg N - 2\lg\lg N)$ rectangular regions of size $\lg N - 2\lg\lg N$. The probability that a given one of the regions realizes the pattern is

$$2^{-\lg N + 2\lg\lg N} = 1 - \frac{\lg^2 N}{N} .$$

The probability that every region avoids the pattern is therefore

$$\begin{aligned} \left(1 - \frac{\lg^2 N}{N}\right)^{\frac{N}{\lg N - 2\lg\lg N}} &\leq e^{\left(-\frac{\lg^2 N}{N}\right)\left(\frac{N}{\lg N - 2\lg\lg N}\right)} \\ &= e^{-\frac{\lg^2 N}{\lg N - 2\lg\lg N}} \\ &\leq e^{-\lg N} \\ &\leq \frac{1}{N} , \end{aligned}$$

using Facts 1 and 3. ∎

In a region of m cells on a wafer, the expected number of live cells is $\frac{1}{2}m$. The actual number will vary, however. The next fact gives tight bounds on the expected deviation.

Fact 6. *Let X be the random variable indicating the number of live cells in a region with m cells. Then the expectation of the deviation is*

$$E(\left|X - \frac{1}{2}m\right|) = \Theta(\sqrt{m}) . \quad \blacksquare$$

Fact 6 tells us that the expected deviation from the mean is $\Theta(\sqrt{m})$. We shall occasionally need to bound the actual probability of some given deviation. The next fact provides such a bound.

Fact 7. *Let X be the random variable indicating the number of live cells in a region with m cells. Then for $r \geq 0$, the probability that the deviation exceeds $r\sqrt{m}$ is*

$$\Pr\left\{\left|X - \frac{1}{2}m\right| > r\sqrt{m}\right\} = O(e^{-2r^2}) . \quad \blacksquare$$

We can use Fact 7 to prove a lower bound on the number of live cells in each of a collection of sufficiently large regions. The next fact shows that if each region contains $c \lg N$ cells, for some sufficiently large constant c, then with high probability, there are a substantial number of live cells in the each of the regions.

Fact 8. *For any $c > 4$, and for any particular collection of N regions on an N-cell wafer, each with at least $c \lg N$ cells, the probability is at least $1 - O(1/N)$ that every region contains $\frac{1}{2}c \lg N - \sqrt{c} \lg N$ live cells.*

Proof. The probability that a given region does not contain at least $\frac{1}{2}c \lg N - \sqrt{c} \lg N$ live cells is $O(e^{-2 \lg N}) = O(1/N^2)$ by Fact 7. By Fact 2, the probability that all the N regions on the wafer, overlapping or not, fail to contain at least $\frac{1}{2}c \lg N - \sqrt{c} \lg N$ cells is at most $N \cdot O(1/N^2) = O(1/N)$. $\blacksquare$

3. Integrating one-dimensional arrays

With high probability, the snake-like scheme described in the introduction connects all the live cells on an N-cell wafer into a linear array with wires of length at most $O(\lg N)$. This section gives two procedures that substantially improve and generalize this bound. The first connects all the live cells on a wafer with wires of length $O(\sqrt{\lg N})$, and the second connects most of the live cells with wires of constant length.

Before presenting the algorithms, we first observe that with high probability, wires of length $\Omega(\sqrt{\lg N})$ are required to connect all the live cells on a wafer. The idea is that somewhere on the wafer, there is a live cell in the center of a square region of $\Omega(\lg N)$ dead cells, an observation that follows directly from Fact 5. (An example of such a region is shown in Figure 5.) Therefore, a wire of length $\Omega(\sqrt{\lg N})$ is required to link the isolated live cell to any other live cell.

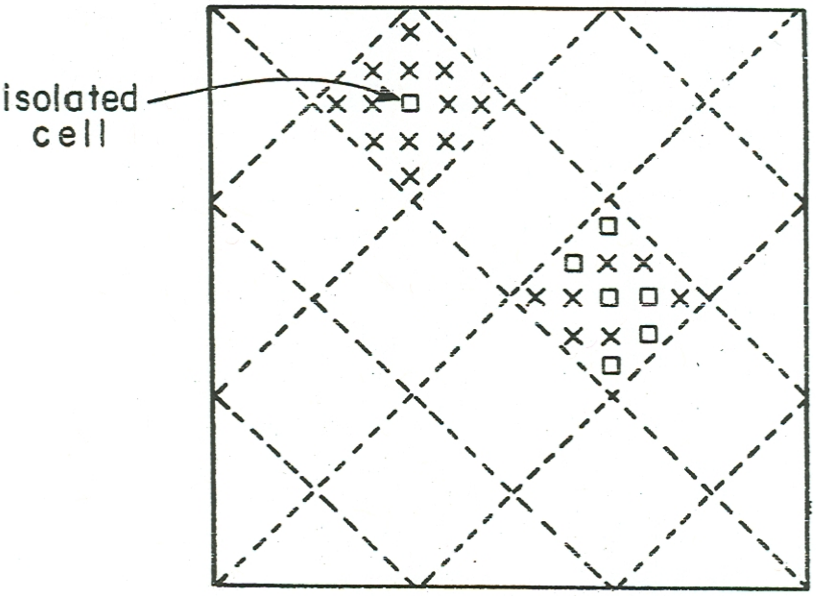

Figure 5. An example of an isolated cell.

3.1. The patching method

The first algorithm for integrating a linear systolic array achieves the lower bound of $\Omega(\sqrt{\lg N})$ by partitioning the wafer into squares, forming linear arrays within each square, and then patching together the ends of the small linear arrays to yield a single linear array consisting of all the live cells on the wafer.

More precisely, the method is as follows. Partition the wafer into square regions containing $2 \lg N$ cells each, as is shown by the dashed lines in Figure 6. The probability that each of the

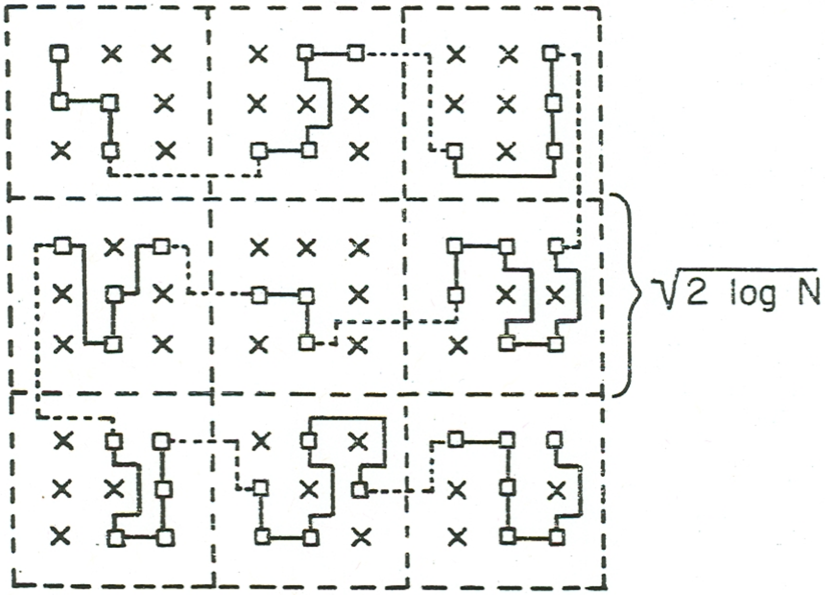

Figure 6. A scheme for constructing linear arrays from all live cells on a wafer with wires of length $O(\sqrt{\lg N})$ and constant channel widths.

$2 \lg N$ cells are dead in one or more of the squares is less than $1/N$ by Fact 4. Thus, with high probability, each of the squares contains at least one live cell.

Construct a linear array out of the live cells in each square using a snake-like scheme on the columns of the square, except that when an empty column is encountered, skip over it. Figure 6 shows these connections with solid lines. Since any pair of cells in the same square can be linked with a wire of length at most $2\sqrt{2 \lg N}$, the wires in each array have length $O(\sqrt{\lg N})$. Next, add wires, shown by dotted lines in the figure, to connect the small arrays into one large array. Because each region contains at least one live cell, these connections can be made with wires of length at most $3\sqrt{2 \lg N}$. Thus, every wire in the completed linear array has length $O(\sqrt{\lg N})$ with high probability.

3.2. The tree method

If all the cells are incorporated in a linear array using the patching method, then the maximum wire length is $\Theta(\sqrt{\lg N})$ with high probability. But the proof of the lower bound suggests that isolated cells induce the long wires. Instead of insisting that *all* live cells be incorporated in the linear array, suppose we only require that *most* of the live cells be included. This section describes a procedure that can construct a linear array from almost all of the live cells with constant-length wires.

The procedure relies on the fact that most live cells on the wafer are near each other. More specifically, it has been proved [17] that there exists a positive constant c such that for any d, with probability $1 - O(1/N)$, at least $1 - O(2^{-cd^2})$ of the live cells on an N-cell wafer can be connected in a tree using wires of length at most d. Up to constant factors, this is the best possible bound.

The algorithm consists of two parts. First, a tree T of live cells is constructed with wires of length at most d, and then the tree is transformed into a linear array with wires of length at most $6d$. (The constant 6 is due in part to our assumption that the width of a wire equals the width of a cell. If wire widths are substantially smaller, the constant shrinks closer to 3.)

The tree T can be constructed by any of the algorithms that compute the minimum spanning tree of a graph. In particular, Prim's method [1, 5, 24] can be modified to compute the spanning tree in linear time.

The construction of the linear array from the tree depends on a result by Sekanina [29] which states that the cube of a nontrivial connected graph always has a Hamiltonian circuit. Specifically, we now show that, without regard for wire widths, the linear array can be constructed using wires of length $3d$ by tracing over wires in the tree T no more than twice each. Since every wire is traced over at most twice, the channel widths could (at worst) double in the resulting wiring, thereby increasing the maximum wire length from $3d$ to $6d$ when wire widths are accounted for.

Choose a node v to be the root of T, and let $T_1, T_2, \ldots, T_m$ be the subtrees of v as is shown in Figure 7. (Degenerate cases not like Figure 7 are easily handled, but we do not include the details here.) Recursively construct linear arrays on the nodes of $T_1, T_2, \ldots, T_m$ such that no wire has length greater than $3L$, and so that the end points of the array in T_i are v_i and u_{i1} for $1 \leq i \leq m$. Then join the arrays in the subtrees by adding the following wires: (v, u_{11}), $(v_1, u_{21}), (v_2, u_{31}), \ldots, (v_{m-1}, u_{m1})$. (These wires are shown as dashed lines in Figure 7.) Each of these wires has length at most $3L$, and the resulting network is a linear array on the nodes of T with endpoints v and v_m. For completeness, we remark that the boundary conditions of the recursion are easily handled.

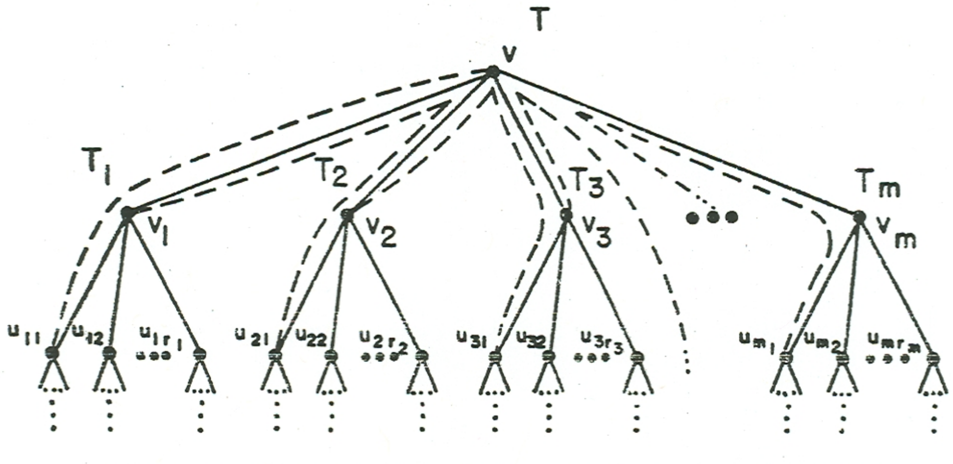

Figure 7. Constructing a linear array from a spanning tree.

4. Integrating two-dimensional arrays

The problem of linking the live cells on a wafer to form a square two-dimensional systolic array is substantially more difficult than the corresponding problem for linear arrays. The main difficulty with constructing two-dimensional arrays is that constant length wires no longer suffice even if we throw away some of the live cells [8]. In fact, it has been shown [17] that with high probability, every realization of an M-cell two-dimensional array on an N-cell wafer has a wire of length $\Omega(\sqrt{\lg M})$, for all $M = \Omega(\lg^2 N)$. This result means, for example, that wires of length $\Omega(\sqrt{\lg N})$ are required to connect just one percent of the live cells.

In order for an algorithm to be effective in realizing a two-dimensional array, it must respect the two-dimensional constraints inherent in the problem. For example, consider the following naive algorithm for realizing an M-cell square two-dimensional array from all the live cells of an N-cell wafer. We assume for convenience that $M \approx N/2$ is a perfect square.

Take the top $\sqrt{M}$ live cells on the wafer, breaking ties randomly. These cells, in order left to right, make the first row of the array. Take the top $\sqrt{M}$ cells of the remainder as the second row, in order left to right, and continue similarly to make each row of the array. With high probability no row of the array contains cells from more than three rows of the wafer because Fact 8 guarantees that every row contains nearly $\frac{1}{2}\sqrt{N} \approx 0.7\sqrt{M}$ live cells.

At first, this method does not seem so bad because (Fact 5) the horizontal connections among the cells of the array have length $\Theta(\lg N)$. The vertical connections are much worse, however. Consider a vertical line which divides the wafer into left and right halves. Fact 6 says that we can expect that the number of cells in a given row on one side of the dividing line is at least $\Omega(\sqrt{\sqrt{M}}) = \Omega(N^{1/4})$ larger than the number on the other side. Thus, with constant probability, the midpoint of the row is at least $\Omega(N^{1/4})$ cells away from the dividing line. Two consecutive rows have their midpoints on opposite sides of the dividing line half the time, and thus, with constant probability, a wire connecting the two midpoints has length $\Omega(N^{1/4})$. Since there are $\sqrt{M}$ rows, there is a wire of length $\Omega(N^{1/4})$ between two of them with high probability. A bound of $\Theta(N^{1/4}\sqrt{\lg N})$ for the maximum wire length in the resulting array can be shown with more detailed analysis.

4.1. The tree-of-meshes method

This section presents an algorithm which can constuct a two-dimensional array from all the live cells of an N-cell wafer if the channels have width $\Omega(\lg N)$. All possible configurations of live and dead cells, however unlikely, can be handled by this technique, but the wire length bounds are not good. This result will be used as a subroutine in the divide-and-conquer and patching methods to achieve better bounds for wire length on *average*-case wafers.

We first show how an N-cell wafer with channels of width $\Theta(\lg N)$ can be viewed as an N-leaf tree of meshes [2, 14, 15, 16]. The tree of meshes is constructed from a complete binary tree by replacing nodes of the tree with meshes and single edges of the tree with bundles of edges linking the meshes. Figure 8 shows a 16-leaf tree of meshes. The root of an N-leaf tree of meshes is a $\sqrt{N}$-by-$\sqrt{N}$ mesh. (We assume for simplicity that $\sqrt{N}$ is a power of 2.) The nodes at the second level are $\sqrt{N}/2$-by-$\sqrt{N}$ meshes, those at the third level are $\sqrt{N}/2$-by-$\sqrt{N}/2$ meshes, and so on until the leaves are replaced by 1-by-1 meshes.

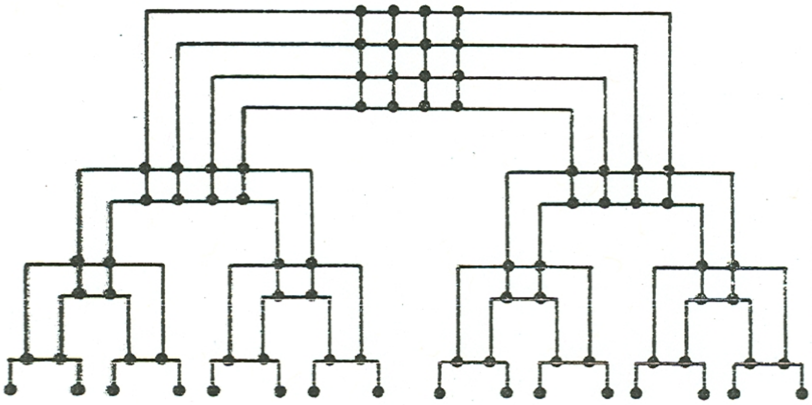

Figure 8. The 16-leaf tree of meshes.

The correspondence between the N-cell wafer and the N-leaf tree of meshes is established as follows. The first step is to construct a $\lg N$-layer three-dimensional layout [18, 26] of the tree of meshes. Fold the connections between the root of the tree of meshes and each of its two children so that the children fit naturally on a second layer over the root. Fold the connections to each of the grandchildren so that they fit naturally over the children on a third layer, and so forth. This procedure generates a $\lg N$-layer three-dimensional layout where each layer has area N. Next, project the three-dimensional layout onto a single layer in the manner of [31, pp. 36–38]. Locate cells of the wafer at the leaves of the tree of meshes. The crosspoints of the meshes become programmable switches, and the wires of the meshes become the wires in $\lg N$-width channels.

We now wish to make a two-dimensional array from the $M \approx N/2$ live leaves of the tree of meshes. (In general, an exact square array is not possible, and thus we shall assume the array to be formed is missing some border cells, as is shown in Figure 9.) We first use divide-and-conquer to assign each cell a number from 1 to M. We chop the M-cell array in half vertically into two subarrays with $\lfloor M/2 \rfloor$ and $\lceil M/2 \rceil$ cells. We recursively assign numbers from 1 to $\lfloor M/2 \rfloor$ to the first subarray and numbers from $\lceil M/2 \rceil$ to M to the second subarray, alternating the orientation of the cut between horizontal and vertical at each recursive step.

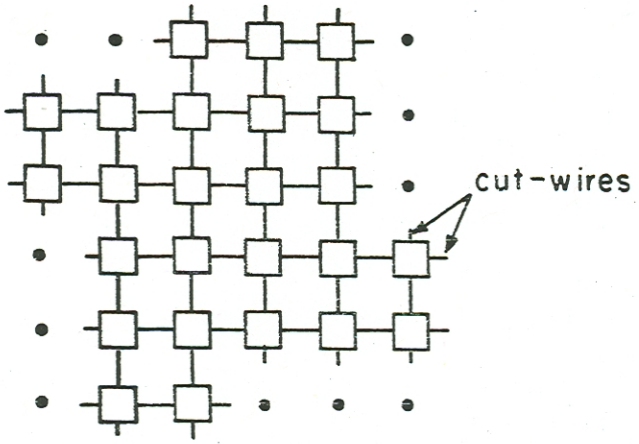

Figure 9. A 6-by6 array that is missing some border cells.

The assignment is now simple. The ith cell of the array is mapped to the ith live leaf of the tree of meshes counting from left to right. After swelling the channel capacities by a small constant factor to accommodate the wires, adjacent cells can be connected by routing wires through the unique path in the underlying complete binary tree. Routing through the meshes can be done by treating them as crosspoint switches. The wire lengths are $O(\sqrt{N} \lg N)$ since we need to route across $O(\sqrt{N})$ channels of width $\Theta(\lg N)$.

As a practical matter, the tree of meshes need not be used directly for routing wires. The assignment algorithm can be used to establish the correspondence between the two-dimensional array and the live cells of the wafer, and then the wires can be routed using a standard gate-array routing program. In the case when $\sqrt{M}$ is an exact power of 2, the assignment is particularly simple. The kth live cell corresponds to the (i, j) position of the array, where i is obtained by concatenating the even bits of the binary representation of k, and j is obtained by concatenating the odd bits.

4.2. The divide-and-conquer method

The tree-of-meshes algorithm works as well as might be expected in the worst case, and thus it is natural to wonder how well it works on average. Unfortunately, the algorithm works poorly in a probabilistic model because the maximum wire length is nearly always large. This section presents a similar divide-and-conquer algorithm which works poorly in the worst case, but which can be proved to work extremely well on average. With high probability, the algorithm connects all the live cells of an N-cell wafer with channels of width $O(\lg \lg N)$ using wires of length $O(\lg N \lg \lg N)$.

The divide-and-conquer algorithm has two stages. In the first stage, the wafer is recursively bisected, and the number of live cells in each half is counted. Based on the count of live cells in each half of the wafer, the algorithm computes the dimensions of the two subarrays that must be constructed, and then recursively constructs the subarrays. The two subarrays are then linked together to form the complete array. The algorithm remains in the first stage as long as the distribution of cells within the current region of the wafer is good, which (with high probability) is until subproblems with $\Theta(\lg N)$ cells are encountered. Below this point, the distribution of cells can be arbitrarily bad, and thus the algorithm uses the tree-of-meshes technique to complete the wiring of a $\Theta(\lg N)$-cell subarray. The exact crossover point between the first and second stages can be set at subproblems of size $c \lg N$, where c is any constant sufficiently large to ensure that with high probability, every $c \lg N$-cell region contains $\Omega(\lg N)$ live cells. That such a c exists is a consequence of Fact 8.

Figures 10 through 13 illustrate the divide-and-conquer procedure. Figure 10a shows a 64-cell wafer which contains 36 live cells. In what follows, we step through the algorithm as it constructs a 6-by-6 array, which is identified as the "overall target" in Figure 10b.

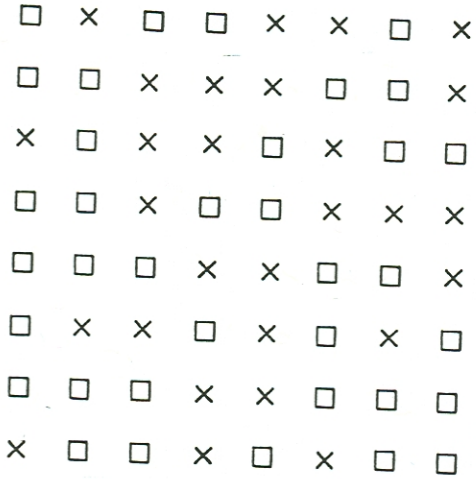

Figure 10a. A 64-cell wafer that contains 36 live cells.

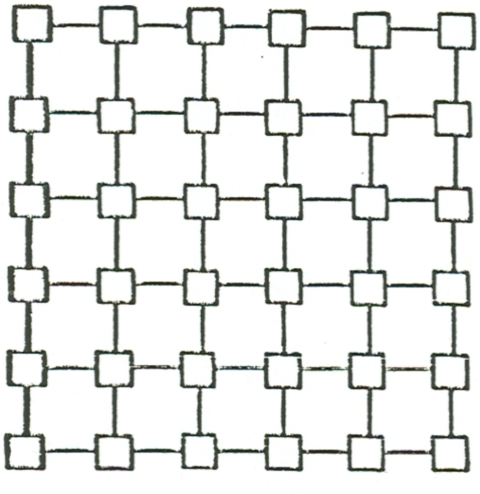

Figure 10b. The target: a 6-by-6 systolic array.

The first step is to bisect the wafer vertically, which gives 19 live cells in the left half and 17 in the right. We wish to construct a 19-cell subarray in the left half wafer and a 17-cell subarray in the right half wafer. Since we want the two subarrays to fit together nicely after they have been constructed, we choose the shapes of the two subarrays that are determined by the partition of the 6-by-6 array shown in Figure 11.

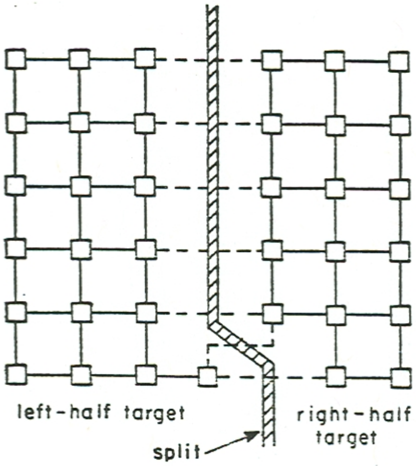

Figure 11. Partitioning the target..

We now invoke the procedure recursively on the two subarrays, but this time we bisect each of the halves horizontally. For example, when the left half wafer is bisected, the 19 live cells are divided into 9 cells above and 10 cells below, as displayed in Figure 12. The algorithm continues in this fashion, alternating between horizontal and vertical divisions, until the wafer and the target have been partitioned into $\Theta(\lg N)$-cell regions, at which point the algorithm proceeds to the second stage, and the tree-of-meshes technique is applied.

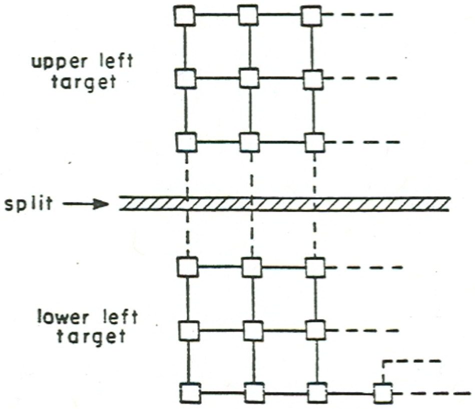

Figure 12. Partitoning the left target.

In this example the number of cells is small enough that the second stage construction can be performed by inspection. The inspection strategy can be used effectively in practice. Since the second stage operates on regions of size $\Theta(\lg N)$, the routings of this size can conceivably be precomputed. The second stage then consists of a single table lookup.

Figure 13 shows the final solution to the problem in Figure 10. For clarity the wires have not been routed within the channels of the wafer. Notice that each quadrant contains the specified targets for second level of recursion. The dashed lines represent wires that connect cells in different quadrants of the wafer.

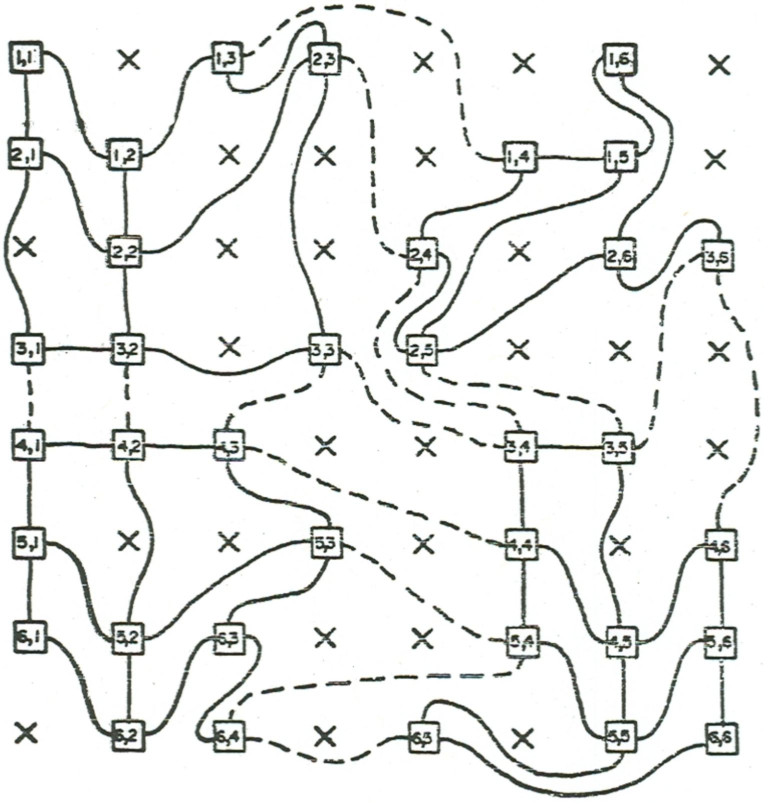

Figure 13. Completed cell assignment and wiring of the 6-by-6 array.

With probability $1 - O(1/N)$ the divide-and-conquer method can construct a two-dimensional array from all the live cells on an N-cell wafer using wires of length $O(\lg N \lg\lg N)$ and channels of width $O(\lg\lg N)$. It is not too difficult to see that these bounds hold with probability 1 for the regions of size less than $c \lg N$ that are connected by the tree-of-meshes procedure. Plugging in $c \lg N$ for N in the tree-of-meshes bound yields wires of length $O(\sqrt{\lg N} \lg\lg N)$ and channels of width $O(\lg\lg N)$.

The hard part is showing that the wiring in the upper levels of recursion satisfy the bounds. The analysis, which we briefly sketch, assumes that during the recursion, the channel dividing a subwafer with $m > c \lg N$ cells has width $\Theta\sqrt{\lg N} \lg m$. Uniform channel widths of $\lg\lg N$ across the entire wafer can later be obtained by distributing the wider channels across neighboring channels, which does not asymptotically increase the wire lengths in the subsequent analysis.

We begin at the first level of recursion. Consider the wires that link a cell in the left subarray to a cell in the right subarray, as is illustrated by the two examples in Figure 14. For the most

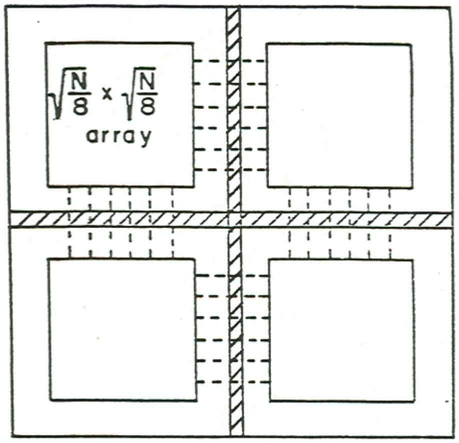

Figure 14a. A distribution of live cells which might allow a narrow center channel.

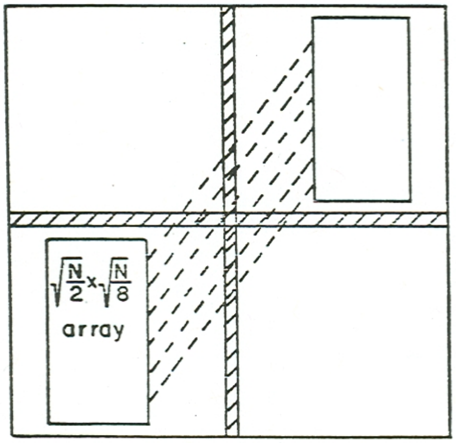

Figure 14b. A distribution of live cells which requires a wide center channel.

part, the connecting wires can be routed in the channel that separates the left and right halves of the wafer. The length of the longest wire in the channel is proportional to the longest vertical distance that a single wire must traverse, as is the width of the channel itself.

The length of the longest wire in the center channel depends on the distribution of cells in each quadrant. For example, if we are extremely lucky and the live cells are regularly spaced, the longest wire may have constant length, as in Figure 14a. But if we are very unlucky, half the live cells might occur in the upper right quadrant and the other half in the lower left quadrant (Figure 14b). To connect the two halves in this latter case, some wire must have length $\Omega(\sqrt{N})$.

The length of the longest wire in the center channel can also be influenced by the distribution of cells within a quadrant. For example, if the upper left quadrant contains $\sqrt{N/8}$ live cells (about the right number), but they are distributed as in Figure 15, then the center channel still contains a wire of length $\Omega(\sqrt{N})$.

Most often, we are not so unlucky that a wire in the center channel has length $\Omega(\sqrt{N})$, but neither are we lucky enough that all wires are constant length. With high probability, we are more lucky than unlucky because the length of the longest wire in the center is $O(\lg N)$. The idea is that the live cells are distributed so evenly that with high probability, the total vertical

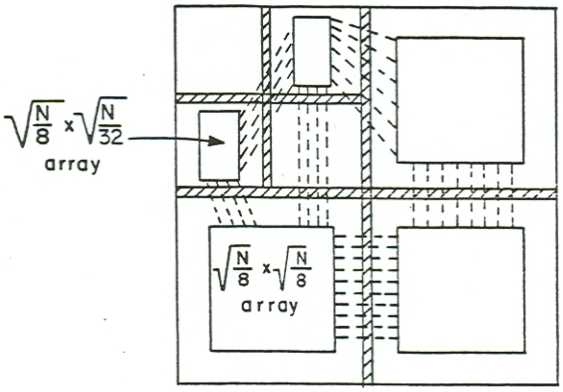
$\sqrt{\frac{N}{8}} \times \sqrt{\frac{N}{32}}$ array
$\sqrt{\frac{N}{8}} \times \sqrt{\frac{N}{8}}$ array

Figure 15. Another distribution of live cells which requires a wide center channel.

distortion of the wires in the center channel (over all subproblems of size $\Omega(\lg N)$) is $O(\lg N)$. For channels dividing a subwafer of size $m > c \lg N$, the vertical distortion is $O(\sqrt{\lg m \lg N})$. Thus, the channel width bounds assumed earlier suffice.

The wire length analysis of the divide-and-conquer *algorithm* is fairly tight. For example, the algorithm requires wires of length $\Omega(\lg N)$ with high probability. Thus, if the lower bound of $\Omega(\sqrt{\lg N})$ is to be achieved, a different algorithm must be discovered. It may be possible to improve the channel width bound, however. For example, any improvement in the worst-case bound given by the tree-of-meshes technique would lead directly to an improvement in the channel width bounds for the divide-and-conquer algorithm.

4.3. The patching method

Not surprisingly, we can improve the wire length bounds if we need only construct a two-dimensional array from *most* of the live cells on a wafer. In particular, we can use a scheme similar to the patching scheme from Section 3.1 to construct a two-dimensional array from any constant fraction (less than 1) of the live cells on an N-cell wafer using wires of length $O(\sqrt{\lg N}\ \lg\lg N)$ and channels of width $O(\lg\lg N)$. These bounds are also achieved with high probability.

The key idea is to partition the wafer into $N/c \lg N$ square regions, each containing $m = c \lg N$ cells. According to Fact 8, we can choose c sufficiently large such that with probability $1 - O(1/N)$, each of the regions contains at least $m' = \frac{1}{2} c \lg N - \sqrt{c} \lg N$ live cells. Using the tree-of-meshes technique, we can therefore construct an m'-cell two-dimensional array in each region using wires of length $O(\sqrt{m}\ \lg m) = O(\sqrt{\lg N}\ \lg\lg N)$ and channels of width $O(\lg m) = O(\lg\lg N)$. The $N/c \lg N$ two-dimensional arrays are then connected together into one large array with $\frac{1}{2} N(1 - 2/\sqrt{c})$ live cells. The added wires also have length at most $O(\sqrt{\lg N}\ \lg\lg N)$, and can easily fit into the $\Theta(\lg\lg N)$-width channels.

The patching method can be thought of as a refinement of the divide-and-conquer method that throws away a fraction of the cells at each level of the recursion. The actual decisions as to which cells at a given level are thrown away can be postponed until lower in the recursion, but it is important that at each level, every region of the wafer have exactly the same number of live cells.

4.4. Greene's method

The next method, due to Greene [7], also connects any constant fraction of the live cells on an N-cell wafer into a two-dimensional array. With high probability, it uses wires of length

$\Theta(\sqrt{\lg N})$ and channels of constant width, thus achieving the lower bound for integration of two-dimensional arrays. It is similar to the algorithm presented at the beginning of this section in that it creates rows of the array, but it is considerably more clever. The algorithm that determines the rows and columns of the array is based on network flow techniques, but we present it in a manner that does not require a knowledge of combinatorial optimization.

Greene's algorithm can construct a $(1-\epsilon)\sqrt{N}$-by-$\frac{1}{2}(1-\epsilon)\sqrt{N}$ array, for any constant $\epsilon > 0$. For any such ϵ, we require the N-cell wafer to have channels of width w, where w is a sufficiently large constant that depends on ϵ. The higher the percentage of cells we wish to integrate into an array, the wider we must make the channels.

Partition the wafer as shown in Figure 16 into blocks of size 1-by-$c_1\sqrt{\lg N}$ such that there are $\sqrt{N}$ rows of blocks and $\sqrt{N}/c_1\sqrt{\lg N}$ columns of blocks, where c_1 is a constant depending

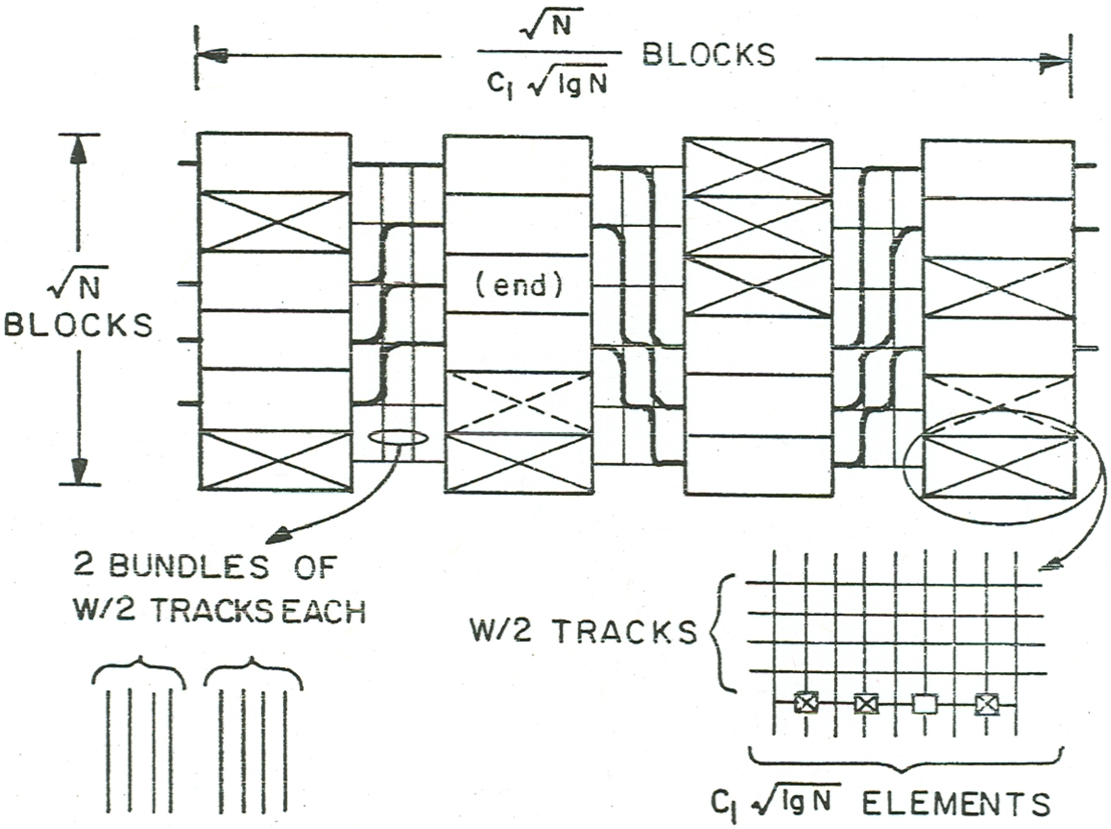
$\frac{\sqrt{N}}{C_1\sqrt{\lg N}}$ BLOCKS
$\sqrt{N}$ BLOCKS
(end)
2 BUNDLES OF W/2 TRACKS EACH
W/2 TRACKS
$C_1\sqrt{\lg N}$ ELEMENTS

Figure 16. Forming the tentative rows in Greene's method. Blocks containing fewer than t live cells are marked with solid X's. Blocks marked as bad during the scan are marked with dashed X's.

on ϵ. Mark a block as *bad* if it contains fewer than t live cells, and *good* otherwise, where t is also a constant depending on ϵ. For the exact values of constants, we refer the reader to [7].

The first part of the algorithm determines tentative rows for the array. We divide the w vertical tracks between blocks on the wafer into two *bundles*, each consisting of $w/2$ tracks. For this part of the algorithm, we will treat the two bundles as two routing tracks. Later, we will need to reexpand the capacity of the two tracks by $w/2$ each.

The algorithm first determines $(1-\epsilon)\sqrt{N}$ horizontally running chains from the left edge of the wafer to the right edge through the good blocks. The chains must satisfy the constraint that no wire is longer than $c_2\sqrt{\lg N}$, for some constant c_2 depending on ϵ. The algorithm determines the chains in the following manner. Scan the columns of blocks left to right. For each column, proceed through the blocks from top to bottom. At each point, if the current block is good, we attempt to connect it to a good block on the left. This connection is made to the uppermost good block within distance $c_2\sqrt{\lg N}$, up or down, from the current block that has not yet been connected to a block in the current column. It must also satisfy the constraint that the routing does not exceed the channel capacity of 2. If such a connection cannot be made, we mark the current block as bad. Block (5,2) in Figure 16 is marked bad for this reason. Some chains are terminated by this procedure—for example, the chain ending in block (3,2) of the figure. With high probability, however, this procedure establishes $(1-\epsilon)\sqrt{N}$ horizontal chains, each with $\sqrt{N}/c_1\sqrt{\lg N}$ blocks.

The horizontally running chains can be viewed conceptually as shown in Figure 17. We now expand the blocks in the chains to see their internal structure, as shown in Figure 18. The

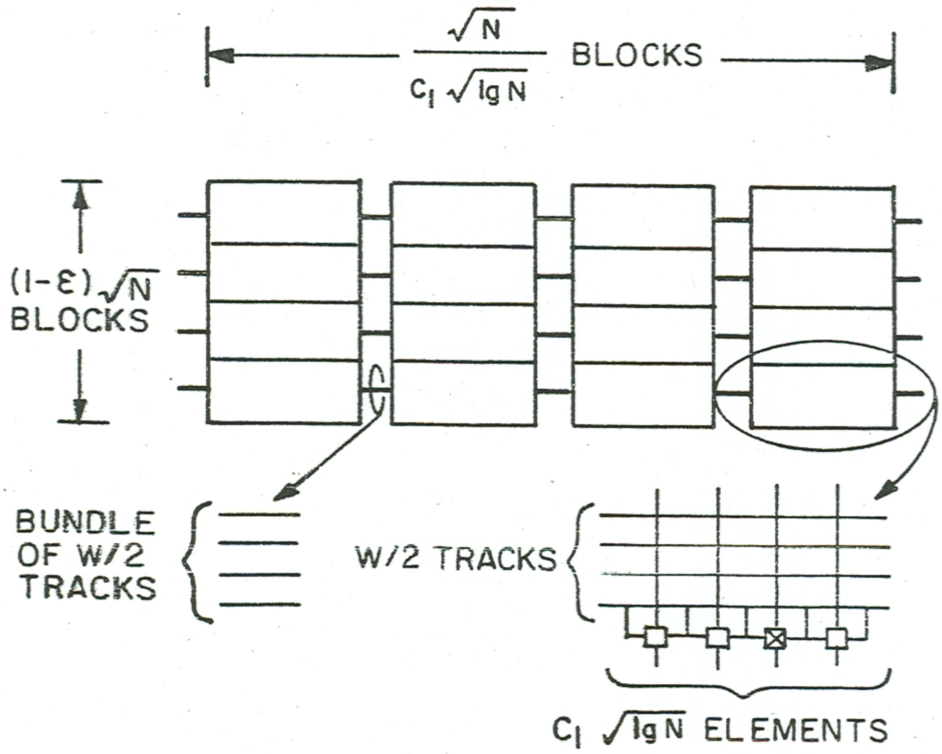
$\frac{\sqrt{N}}{C_1\sqrt{\lg N}}$ BLOCKS
$(1-\varepsilon)\sqrt{N}$ BLOCKS
BUNDLE OF W/2 TRACKS
W/2 TRACKS
$C_1\sqrt{\lg N}$ ELEMENTS

Figure 17. Normalized view of the rows of blocks.

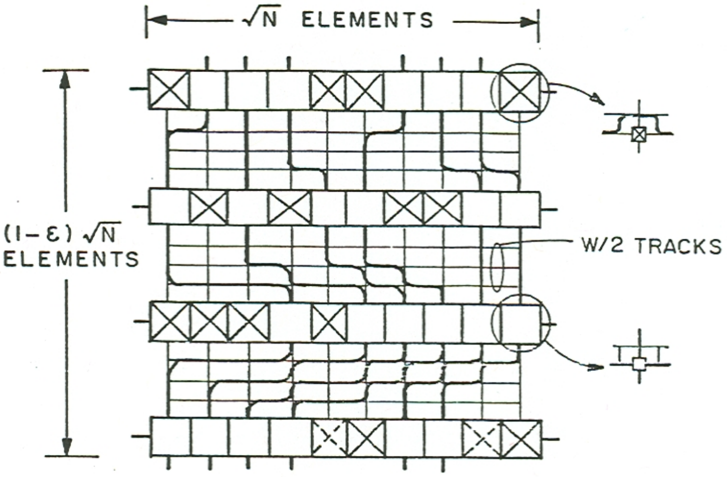

Figure 18 Forming the columns in Greene's method. Dead cells are marked with solid X's. Cells marked as bad during the scan are marked with dashed X's.

horizontal tracks in Figure 18 actually correspond to sections of both horizontal and vertical tracks in Figure 16 because the chains run both horizontally and vertically. The horizontal channels in Figure 18 have $w/2$ tracks, and thus the two vertical tracks between blocks in Figure 16 must each be expanded by $w/2$ to accommodate the wires we shall now route to make the vertical connections.

We establish the vertically running chains by essentially the same procedure as before, except we scan top to bottom and route through horizontal channels of width $w/2$. With high probability, the algorithm constructs $\frac{1}{2}(1-\epsilon)\sqrt{N}$ vertical chains. The horizontal chains are now modified to include only those cells used in the vertical chains, which completes construction of the $(1-\epsilon)\sqrt{N}$-by-$\frac{1}{2}(1-\epsilon)\sqrt{N}$ array. All channels are constant width w, and it turns out to be the case that all wire lengths are $O(\sqrt{\lg N})$.

Greene's method generates a rectangular array with aspect (length to width) ratio 2, but we may wish to realize a square array without throwing away half the cells. By embedding a $(1-\epsilon)\sqrt{N/2}$-by-$(1-\epsilon)\sqrt{N/2}$ square array into a $(1-\epsilon)\sqrt{N}$-by-$\frac{1}{2}(1-\epsilon)\sqrt{N}$ rectangular array so that adjacent cells of the square array are constant distance away in the rectangular array, we can use Greene's method directly. The first row of the square is embedded in the first two rows of the rectangle such that all the first row of the rectangle is used and an evenly spaced portion of the second row is used. We connect the cells of the first row of the square linearly left to right in the rectangle. The second row of the square is embedded linearly in the second and third rows of the rectangle using all the remaining cells in the second row and a uniformly spaced portion of cells in the third row. The third row of the square uses all the remaining cells in the third row of the rectangle, all the cells in the fourth row, and a uniformly spaced portion of cells from the fifth row. We continue in this fashion until the embedding is completed. Every adjacent pair of cells in the square array are within horizontal and vertical distances of four cells in the rectangular array. This procedure can be generalized to construct any rectangular array of any aspect ratio.

4.5. The matching method

We conclude with a method whose proven bounds are not as good as those presented thus far, but which is nevertheless interesting. In the case of widthless wires, this method, which is based on bipartite matching in a graph, can integrate all the cells on an N-cell wafer with wires of length $O(\lg^{3/4} N)$. When we consider the normal case of unit width wires, however, we could conceivably need channels of width $\Theta(\lg^{3/4} N)$, and because the wires would need to cross these channels, wires of length $O(\lg^{3/2} N)$. This algorithm is certainly worth considering when wire widths are small because the $O(\lg^{3/4} N)$ wire-length bound is better than the bound of $\Theta(\lg N)$ which the divide-and-conquer method yields for widthless wires. Moreover, the true performance of the matching method might be better than that suggested by the upper bound for unit-width wires. In comparison, the divide-and-conquer method has a hard lower bound of $\Theta(\lg N)$ even for widthless wires. In addition, the algorithm is easily tailored to handle the situation when we wish to integrate any constant fraction of the live cells, in which case the widthless wire bound shrinks to $\Theta(\sqrt{\lg N})$, which is optimal.

The first step of the matching method is to determine the number M of live cells on an N-cell wafer. Then we pick a target wire length d that we hope to achieve. The algorithm now determines the locations of points in a uniform $\sqrt{M}$-by-$\sqrt{M}$ grid superimposed on the wafer. It then constructs a bipartite graph between the grid points and the live cells of the wafer with an edge between a grid point and a live cell if the distance between them is at most d. Then, using a bipartite matching algorithm [5], the procedure determines whether every grid point can be matched one-to-one with a live cell. If a perfect matching exists, then we know a routing of the corresponding assignment with widthless wires has maximum edge length d.

It is possible to show [19, 30] that if $d = \Theta(\lg^{3/4} N)$, then the matching succeeds with high probability. As a practical matter, it is better to search for the smallest d that works for a given wafer using exponential search. Try $d = 1, 2, 4, 8, \ldots$ until a value of d is found that results in a perfect matching, and then binary search to find the exact value.

The same technique can be applied to construct a two-dimensional array from any number $m < M$ of the M live cells by using a $\sqrt{m}$-by-$\sqrt{m}$ grid. For the case when $m = (1 - \epsilon)M$, it can be shown that wires have length $O(\sqrt{\lg N})$ with high probability.

5. Summary and conclusions

The content of this paper is taken primarily from [17] and somewhat from [7] and [8]. The algorithms presented are summarized in Tables I and II. The literature contains many more techniques for integrating systolic arrays. Manning [22, 23], Hedlund and Snyder [9], Koren [11], and Fussell and Varman [6] look at the basic problem of constructing arrays from wafers containing faulty cells. Rosenberg [27, 28], Chung, Leighton, and Rosenberg [3, 4], and Bhatt and Leighton [2] have also investigated fault tolerance.

Table I
Bounds for One-Dimensional Arrays

Method	Portion of cells used	Maximum wire length	Maximum channel width
patching	all	$\Theta(\sqrt{\log N})$	$\Theta(1)$
optimal	all	$\Theta(\sqrt{\log N})$	$\Theta(1)$
tree	99%	$\Theta(1)$	$\Theta(1)$

Table IIa
Bounds for Two-Dimensional Arrays
(worst-case wafer, using all live cells)

Method	Maximum wire length for widthless wires	Maximum channel width	Maximum wire length for unit width wires
tree of meshes	$\Theta(\sqrt{N})$	$O(\log N)$	$O(\sqrt{N}\log N)$
optimal	$\Theta(\sqrt{N})$	$\Omega(1)$	$\Omega(\sqrt{N})$

Table IIb
Bounds for Two-Dimensional Arrays
(average-case wafer, using all live cells)

Method	Maximum wire length for widthless wires	Maximum channel width	Maximum wire length for unit width wires
divide & conquer	$\Theta(\log N)$	$O(\log\log N)$	$O(\log N \log\log N)$
matching	$O(\log^{3/4} N)$	$O(\log^{3/4} N)$	$O(\log^{3/2} N)$
optimal	$\Omega(\sqrt{\log N})$	$\Omega(1)$	$\Omega(\sqrt{\log N})$

Table IIc
Bounds for Two-Dimensional Arrays
(average-case wafer, using 99% of the live cells)

Method	Maximum wire length for widthless wires	Maximum channel width	Maximum wire length for unit width wires
patching	$\Theta(\sqrt{\log N})$	$O(\log\log N)$	$O(\sqrt{\log N}\log\log N)$
Greene	$\Theta(\sqrt{\log N})$	$\Theta(1)$	$\Theta(\sqrt{\log N})$
matching	$\Theta(\sqrt{\log N})$	$O(\sqrt{\log N})$	$O(\log N)$
optimal	$\Theta(\sqrt{\log N})$	$\Theta(1)$	$\Theta(\sqrt{\log N})$

Acknowledgments

We would like to thank the members of the MIT Lincoln Laboratory Restructurable VLSI project for acquainting us with the details of their work and for providing the photographs in Figures 1, 2, and 3.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms,* Addison-Wesley, Reading, Massachusetts, 1983.

[2] S. N. Bhatt and F. T. Leighton , "A framework for solving VLSI graph layout problems," *Journal of Computer and System Sciences*, Vol. 28, No. 2, April 1984, pp. 300–343.

[3] F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, "Diogenes: a methodology for designing fault-tolerant VLSI processor arrays," *Proceedings of the IEEE Symposium on Fault-Tolerant Computing*, June 1983, pp. 26-31.

[4] F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, "Embedding graphs in books: a layout problem with applications to VLSI design," MIT-VLSI Technical Memo, 1985.

[5] S. Even, *Graph Algorithms,* Computer Science Press, Rockville, Maryland, 1979.

[6] D. Fussell and P. Varman, "Fault-tolerant wafer-scale architectures for VLSI," *Proceedings of the 9th Annual IEEE/ACM Symposium on Computer Architecture*, April 1982, pp. 190–198.

[7] J. W. Greene, *Configuration of VLSI Arrays in the Presence of Defects,* Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, December 1983.

[8] J. W. Greene and A. El Gamal, "Configuration of VLSI arrays in the presence of defects," *JACM,* Vol. 31, No. 4, October 1984, pp. 694–717.

[9] K. Hedlund and L. Snyder, "Wafer-scale integration of configurable, highly parallel (CHiP) processors," *Proceedings of the IEEE International Conference on Parallel Processing*, 1982, pp. 262–264.

[10] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamiltonian paths in grid graphs," *SIAM Journal of Computing,* Vol. 11, No. 4, November 1982, pp. 676–686.

[11] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," *Proceedings of the 8th Annual IEEE/ACM Symposium on Computer Architecture*, May 1981, pp. 425–431.

[12] H. T. Kung, "Why systolic architectures," *Computer Magazine,* IEEE, January 1982, pp. 37–46.

[13] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," *Sparse Matrix Proceedings 1978,* edited by I. S. Duff and G. W. Stewart, Society for Industrial and Applied Mathematics, pp. 256–282.

[14] F. T. Leighton, *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*, MIT Press, 1983.

[15] F. T. Leighton, "New lower bound techniques for VLSI," *Math. Systems Theory,* Vol. 17, No. 1, April 1984, pp. 47–70.

[16] F. T. Leighton, "A layout strategy for VLSI which is provably good," *Proceedings of the Fourteenth ACM Symposium on Theory of Computing*, May 1982, pp. 85–98.

[17] F. T. Leighton and C. E. Leiserson, "Wafer-scale integration of systolic arrays," *IEEE Transactions on Computers*, Vol. C–34, No. 5, May 1985, pp. 448–461.

[18] F. T. Leighton and A. L. Rosenberg, "Three-dimensional circuit layouts," *SIAM Journal on Computing*, to appear.

[19] F. T. Leighton and P. W. Shor, "Tight bounds for grid matching with application to the average-case analysis of algorithms," in preparation.

[20] C. E. Leiserson, *Area-Efficient VLSI Computation*, MIT Press, Cambridge, Massachusetts, 1983.

[21] J. Logue, W. Kleinfelder, P. Lowy, J. Moulic, and W. Wu, "Techniques for improving engineering productivity of VLSI designs," *Proceedings of the IEEE International Conference on Circuits and Computers*, 1980.

[22] F. Manning, *Automatic Test, Configuration, and Repair of Cellular Arrays*, Ph.D. dissertation, MIT Project MAC, June 1975.

[23] F. Manning, "An approach to highly integrated, computer-maintained cellular arrays," *IEEE Transactions on Computers*, Vol. c-26, June 1977.

[24] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Tech. J.*, Vol. 36, 1957, pp. 1389–1401.

[25] J. I. Raffel, "On the use of nonvolatile program links for restructurable VLSI," *Proceedings of the Caltech Conference on VLSI*, January 1979, pp. 95–104.

[26] A. L. Rosenberg, "Three-dimensional integrated circuitry," *Proceedings of the CMU Conference on VLSI Systems and Computations*, edited by H. T. Kung, R. Sproull, and G. Steele, October 1981, pp. 69–80.

[27] A. L. Rosenberg, "The Diogenes approach to testable fault-tolerant networks of processors," Technical Report CS–1982–6.1, Department of Computer Science, Duke University, May 1982.

[28] A. L. Rosenberg, "On designing fault-tolerant arrays of processors," Duke University Technical Report CS–1982–14.

[29] M. Sekanina, "On an ordering of the set of vertices of a connected graph," *Publications of the Faculty of Science, University Brno*, Czechoslovakia, No. 412, 1960, pp. 137–142.

[30] P. W. Shor, *Average-case analyses for bin packing and planar matching problems*, Ph.D. dissertation, Mathematics Department, MIT, 1985.

[31] C. D. Thompson, *A Complexity Theory for VLSI*, Ph.D. dissertation, Department of Computer Science, Carnegie-Mellon University, 1980.