POLYMORPHIC ARRAYS: A NOVEL VLSI LAYOUT FOR SYSTOLIC COMPUTERS

Amos Fiat and Adi Shamir

Applied Mathematics Department
Weizmann Institute of Science
Rehovot, Israel

(Extended Abstract)

ABSTRACT

This paper proposes a novel architecture for massively parallel systolic computers, which is based on results from lattice theory. In the proposed architecture, each processor is connected to four other processors via constant-length wires in a regular borderless pattern. The mapping of processes to processors is continuous, and the architecture guarantees exceptional load uniformity for rectangular process arrays of arbitrary sizes. In addition, no timesharing is ever required when the ratio of processes to processors is smaller than 1/\sqrt{5}.

1. Introduction

The declining cost of VLSI components and the recent advances in wafer-scale integration make it technically possible and economically feasible to build computers with hundreds of thousands of processors. Since each processor can be physically connected only to a limited number of other processors, the choice of interconnection pattern is a crucial architectural decision. In this paper we propose a novel pattern based on results from the geometry of numbers, which is efficient, easy to use, and optimally versatile (in a sense which will be made precise later).

The interconnection pattern is designed to support systolic algorithms, which are implemented on a rectilinear grid of synchronized processors. A large number of such algorithms have been proposed in the literature, and they represent a very attractive and cost-effective solution to the problem of massive parallelism. Most algorithms are based on linear vectors, fixed width strips, or hexagonal arrays (which are just tilted versions of rectangular arrays), and our interconnection pattern is optimized specifically for such shapes.

The main problem in the design of a general purpose systolic computer is that the hard-wired array must have fixed dimensions which may differ from the size and shape required to solve a particular problem. Consider, for example, the case of a single wafer computer with 10,000 processors arranged in a 100 x 100 square grid. To evaluate a polynomial on this computer, we need a long vector of processes, which has to be bent into a space-filling curve such as a serpentine or a spiral; for a frequent change of direction complicates the programming for different problem sizes. To maintain a priority queue, we need a long strip of width 2; the bends introduce discontinuities (where logically adjacent processes are mapped into physically non-adjacent processors) which disrupt the perfect synchronization. To multiply large matrices on this computer, we may need a 120 x 120 array; when this process array is folded over the 100 x 100 processor array, some processors have to work four times harder than other processors by timesharing the execution of four processes. In addition, some data structure algorithms require dynamically changing arrays in which existing processes die or spawn new processes at runtime; it is desirable to accommodate the extra processes without relocating the existing ones.

In this paper we propose a borderless interconnection pattern for $N$ processors into which rectangular arrays of arbitrary shapes and sizes can be efficiently mapped. The special structure of this pattern guarantees that for any rectangle with up to $N/\sqrt{5}$ processes, at most one process will be mapped to each processor, and for arbitrarily large rectangles the number of processes mapped to any two processors can differ by at most $O(\log N)$ (note that this is an additive constant which does not depend on the number of processes). In all the other known systolic architectures (such as Hewitt's Apriary Network [1980], Martin's Torus [1981], and Sequin's Doubly Twisted Torus [1981]), a process overlap can occur even for arrays with less than $N/\sqrt{5}$ processes. In these architectures the load imbalance for arbitrary process rectangles is either unbounded (the Apriary network) or can be as high as $\sqrt{N}$.

The problem of dynamically changing systolic arrays can be easily handled: As long as the intermediate sizes of the arrays remain at most $N/\sqrt{5}$, the array can change its shape (e.g., from a horizontal vector to a square to a vertical vector) and spawn new processes in any direction without creat-
ing overlaps and without relocating the existing processors. Similarly, the array can grow explosively, change direction, etc., and yet the difference between the number of processes executing on the most heavily loaded processor and the lightest loaded processor will be at most $O(\log(M))$.

The emphasis in this paper is on the choice of the interconnection pattern rather than on the (extensively investigated) problem of optimally embedding one given structure (such as a tree) in another given structure (such as a grid). In fact, the embedding problem is trivial in our architecture since each processor has exactly four links which are permanently labelled by "up", "down", "left" and "right". The bottom left process in any array is mapped to an arbitrary processor, and all the other processes are mapped continuously from there by following the up and right links in the natural order.

We call the proposed structure a polymorphic array since it can assume many different shapes at the same time. A typical example of a polymorphic array is obtained by superimposing the edges in Figures 1 and 2 and the reverse of the edges in Figures 1 and 2. These four sets of edges represent the up, right, down and left links of each processor, respectively. Each set of edges forms a Hamiltonian cycle, and the local topology is equivalent to a grid (i.e., any two directions commute and the (up, down) and (right, left) pairs of directions cancel each other out.) The global topology resembles a torus on which the four Hamiltonian cycles are diagonally wound as interlaced helixes.

By experimenting with this 55 processor computer, it is easy to verify that any rectangular systolic array with at most 35 processes (e.g., 35 x 1, 17 x 2, 11 x 3, 8 x 4 and 7 x 5) is mapped into the computer without overlaps. The natural embedding of a 6 x 6 array, on the other hand, maps processes at opposite corners into the same processor. Note that 35 is a uniform upper bound, and in fact larger arrays (such as a 55 x 1 vector or a 5 x 8 rectangular) can be embedded without overlaps.

When a million process problem is mapped into this 55 processor computer, the average load is 18,182 processes per processor. However, it is the maximal load which determines the effective clock rate and the average processor utilization. In this example, the maximal load is at most 18,184 and the loads on any two processors can differ by at most 3, regardless of the shape of the problem. This simplifies the synchronization and guarantees essentially optimal performance in spite of the extensive timesharing.

An important advantage of the proposed structure is that the length of all the links is constant (at most $\sqrt{6}$). The short wires, the regularity of the design, and the bounded number of wires and crossovers per unit area make it possible to implement large polymorphic arrays efficiently in silicon. In addition, the symmetry of the interconnection pattern makes it fault tolerant to any single failure since it is possible to embed any rectangular array with at most $N/\sqrt{6}$ processes without using any particular processor and without creating overlaps by shifting the origin of the embedding.

2. Definitions and Results

Our interconnection pattern is based on a $N$ by $L$ rectangular grid (with $M=V \times L$ processors). The grid is labelled by a non-standard coordinate system: Along each axis, the even locations are numbered in a forward direction and then the odd locations are numbered in a backward direction (see Fig. 1). This guarantees that along each axis cyclically successive numbers such as 0 and 1, 2 and 3, ... and 10 and 0 are at most two locations apart.

Each processor has a processor number $0 \leq k < M$ and a processor location which is defined as $(k \mod W, k \mod L)$ in this modified coordinate system. When $W$ and $L$ are relatively prime, the Chinese remainder theorem guarantees that the mapping from numbers to locations is one-to-one and onto. We can thus use either numbers or locations to refer to particular processors.

The up, down, left and right links of the processor located at $(i, j)$ are defined as:

- **up** $\rightarrow (i+1 \mod W, j+1 \mod L)$
- **down** $\rightarrow (i-1 \mod W, j-1 \mod L)$
- **left** $\rightarrow (i+1 \mod W, j-1 \mod L)$
- **right** $\rightarrow (i-1 \mod W, j+1 \mod L)$

Note that the up link connects processor number $k$ to processor number $k+1 \mod M$, and the down link connects processor number $k$ to processor number $k-1 \mod M$. The other directions are harder to describe in terms of processor numbers, since they treat the two coordinates differently.

When a path from $(i, j)$ contains $u$ up links, $d$ down links, $l$ left links and $r$ right links, its final destination is $(i+u-d+1-r \mod W, j+u-d-1+r \mod L)$.

This destination does not depend on the order in which the links are traversed, since modular addition is commutative and associative.

We call this interconnection scheme a diagonally connected torus. The length of each link is at most $\sqrt{6}$, since the horizontal and vertical separation between the link's endpoints is at most 2. This locality minimizes the communication delays and simplifies the synchronization between the various processors. The number of crossovers per unit area is a constant which does not depend on the number of processors, and thus the graph is easy to embed in silicon in spite of its non-planarity.

**Definition:** A polymorphic array of order $n$ (for $n$ not divisible by 3) is a diagonally connected torus with $W=F_n$ and $L=L_n$, where $(F_i)$ is the Fibonacci sequence:

$F_1=1, \quad F_2=1, \quad F_i=F_{i-1}+F_{i-2}$ for $i > 2$, and $\{L_i\}$ is the Lucas sequence:

$L_1=1, \quad L_2=3, \quad L_i=L_{i-1}+L_{i-2}$ for $i > 2$. 

38
The optimality of our polymorphic arrays with respect to property 1 follows from one of the theorems proved in [CLNG], which states that the constant \( \sqrt{5} \) cannot be replaced by a smaller constant in any two dimensional grid-like architecture.

A complete list of the first 13 non-trivial polymorphic arrays and their characteristics is given in Table 1. The constants represent the best known results for these small arrays, and are usually better than the general bounds proved in this paper.

For each order of magnitude there is at least one possible array, and thus one can choose a convenient size for any budget and technology. The table also illustrates the difference between the recommended sizes of our processor arrays \((M = 21, 55, 137, 987, \ldots)\) and the recommended sizes of the [CLR] memory arrays \((M = 5, 13, 34, 89, 233, 610, \ldots)\). In fact, four of the six [CLR] values above are prime, and thus they cannot correspond to any rectangular \( M = N \times L \) decomposition.

3. The Mathematical Model

The purpose of this section is to establish the connection between polymorphic arrays and Fibonacci lattices. Once this is done, we can exploit the rich structure and many known properties of these lattices to prove our results.

**Definition:** A Fibonacci lattice of order \( n \), \( T_n \), is the two dimensional lattice spanned by the basis vectors \((0,F_n)\) and \((1,1)\).

Some of the useful properties of Fibonacci and Lucas numbers are:

**Lemma 1:** 1. For any \( n \) which is not divisible by 3, \( F_n \) and \( L_n \) are both even and thus the mapping from processor numbers to locations is not one-to-one.

When a rectangular array of processes is mapped into a polymorphic array, each processor has a load defined as the number of processes assigned to it. If the load is larger than \( 1 \) (a situation called process overlap), the processor must timeshare the execution of all these processes. It is advantageous to avoid the timesharing whenever possible, and to distribute the loads as evenly as possible in all other cases. This makes the following properties of polymorphic arrays particularly interesting:

1. The overlap result: When a rectangular array of arbitrary shape with at most \( M/\sqrt{5} \) processes is mapped into a polymorphic array with \( M \) processors, the loads on all the processors are at most 1.

2. The uniformity result: When a rectangular array with arbitrarily many processes is mapped into a polymorphic array with \( M \) processors, the loads on any two processors differ by at most \((3/2)\log M\).

**Remark:** An exhaustive computer search proved that for all \( n \leq 7 \), the maximal load imbalance in a polymorphic array of order \( n \) is only \( 1/3)\log M \). This may be the case for all \( n \) but we were unable to prove this tighter bound.

Property 1 is related to a result proved by Chor, Leiserson and Rivest [CLR] in connection with the organization of raster graphics memories. Our polymorphic arrays and their memory organizations are based on similar mathematical concepts from lattice theory, even though they differ in their details due to the extra constraints imposed in our application by the short wires and regular layout. The other property of polymorphic arrays was not discussed at all in the [CLR] paper, even though it applies equally well to their memory organization and can make it even more attractive from an engineering point of view.

The optimality of our polymorphic arrays with respect to property 1 follows from one of the theorems proved in [CLNG], which states that the constant \( \sqrt{5} \) cannot be replaced by a smaller constant in any two dimensional grid-like architecture.

We next define a natural labeling of the infinite two dimensional grid of integers which is an "unwound" version of the polymorphic array:

**Definition:** The processor number associated with grid location \((i,j)\) is the processor reached from processor 0 by traversing \( i \) right links and \( j \) up links in the polymorphic array.

It is important not to confuse processor locations (which correspond to the physical layout of the processor array) and grid locations (which correspond to the logical links between the processors). In the grid representation, each processor is linked directly to its four rectilinear neighbours, while in the physical layout each processor is linked to four diagonally non-adjacent neighbours. Note that in the grid representation \( i \) and \( j \) can be arbitrarily large positive or negative integers, and that processor numbers are mapped to the infinite grid in a repetitive pattern due to the cycle structure of the polymorphic array. An example of a grid which is labeled by 13 processor numbers appears in Fig. 3 where the indices represent column numbers.
To determine the mapping of processes to processors, we simply place the process array on the grid representation so that the array's lower left corner is at the grid's origin. The load on each processor is the number of occurrences of its number within the rectangle, and the absence of overlaps is indicated by the disjointness of all the processor numbers within the rectangle. A typical example appears in Fig. 3, where a $5 \times 3$ process array is placed on a 13-processor grid. Since labels 2 and 7 occur twice within the rectangle, these processors must timeshare the execution of two processes, while all the other processors are assigned exactly one process.

To analyse the structure of the labels on the grid, we first consider processor 0. This processor is associated with the origin (0,0) and with any other grid location (i,j) with the property that a right link followed by a left shift forms a (not necessarily simple) cycle in the polymorphic array. These locations are related to Fibonacci lattices in the following way:

**Theorem 2**: The grid locations (i,j) with which processor 0 is associated in a polymorphic array of order $n$ are $T_n$ for even $n$ and its mirror image $T_{2n}$ (with respect to the x-axis) for odd $n$.

**Proof**: By the definition of the right and up links, the processor associated with grid location (i,j) is the processor which is located at $(j-i \mod W, j+i \mod L)$ in the polymorphic array. Processor 0 is thus associated with any grid location (i,j) such that $j-i=0 \mod W$ and $j+i=0 \mod L$. Since these equations are linear and homogeneous, any integral linear combination of solutions is also a solution and thus the set of solutions forms a lattice in the two dimensional grid of integers.

To prove that $(0,(-1)^n T_{2n})$ and $(1,(-1)^n T_{2n}-1)$ is a basis for this lattice, we use the following facts:

1. $(0,(-1)^n T_{2n})$ belongs to the lattice since by Lemma 1 $T_{2n} = F_{n+1} T_n$ and thus $T_{2n} = 0 \mod F_n$ and $T_{2n} = 0 \mod L$.
2. $(1,(-1)^n T_{2n}-1)$ belongs to the lattice since by Lemma 1 $T_{2n-1} = (-1)^n (mod F_n)$ and $T_{2n-1} = (-1)^n (mod L)$.
3. $(0,(-1)^n T_{2n})$ is the shortest lattice vector of the form $(0,a)$ since the up links form a Hamiltonian cycle and thus it is impossible to return to processor 0 by following fewer than $T_{2n}$ up links.
4. The basic paralleloiped defined by these two vectors is contained in the unit-width strip $0 \leq x \leq 1$ and thus cannot contain any other integral lattice points in its interior. Since $(0,(-1)^n T_{2n})$ is the shortest vertical lattice vector, the boundary of the paralleloiped cannot contain lattice points other than its corners. The absence of lattice points in this paralleloiped proves that these two vectors span the whole lattice. Q.E.D.

**Theorem 3**: For any $0 \leq k < F_{2n}$, the grid locations with which processor $k$ is associated in a polymorphic array of order $n$ are $T_{2n} + (0,k)$ for even $n$ and $T_{2n} + (0,k)$ for odd $n$ (i.e., upshifts by $k$ units of the corresponding lattices).

**Proof**: Processor $k$ is associated with all the grid locations $(i,j)$ for which $j-i-k \mod W$ and $j+i-k \mod L$. The general solution of these homogeneous linear equations is the general solution of the homogeneous equations (characterized by Theorem 2) shifted by the particular solution $(0,k)$ of the inhomogeneous equations. Q.E.D.

The union of all the upshifted Fibonacci lattices (or their mirror images) for $0 \leq k < F_{2n}$ is a complete labeling of the infinite two dimensional grid which we call a Fibonacci labeling. Our goal in the rest of this paper is to analyze the distribution of labels in rectangles placed on Fibonacci labeled grids.

4. The Overlap Result

In this section we prove that in a Fibonacci labeled grid with $F_n$ labels, the smallest rectangle that contains a repeated occurrence of some label has at least $F_{n+2}/5$ integral points. Since all the labels share the same lattice structure and the property is invariant under mirroring, it suffices to prove this result for unmirrored Fibonacci lattices and the particular label 0. We can further assume that the rectangle is placed in the first quadrant with its origin on the grid (the other three cases are handled analogously).

For any integral vector $v=(i,j)$ in the first quadrant, let $B(v)$ be the rectangle whose opposite corners are at the origin and at $v$, and let $I(v)$ be the number of integral points in it (i.e., $I(v) = (i+j)(i+j+1)/2$). For any subset $S$ of the grid, we define $P(S)$ to be the minimum size of a rectangle whose intersection with $S$ contains at least two points:

$$P(S) = \min \{ I(v) \mid |B(v) \cap S| \geq 2 \}.$$

By definition, any rectangle with fewer than $P(S)$ integral points can contain at most one representative from $S$. Since the occurrences of label 0 form a Fibonacci lattice in our grid, our goal is to bound $P(T_{2n})$ from below.

**Theorem 4**: $P(T_{2n}) > P_{2n}/5$.

**Proof**: The proof given here applies to our lattices as well as to the lattices analyzed in [CLR], and it is considerably simpler than their proof.

Let $\{v_k\}$ be a sequence of vectors defined as:

$$v_0 = (0,F_{2n})$$
$$v_1 = (1,F_{2n}-1)$$
$$v_k = v_{k-2} - v_{k-1} \quad \text{for } k \geq 2.$$

An explicit expression for these vectors is:

$$v_k = (F_{k-1} F_{n-k}) - ((-1)^{k+1} F_{k} F_{n-k}).$$

These vectors alternate between the first and second quadrant (see Fig. 4). All these vectors belong to $T_{2n}$ and any pair of vectors of the form $(F_{k}, F_{n-k})$
or \( \{v_1, v_{k+2}\} \) is a basis for this lattice (since the original basis vectors \( v_0, v_1 \) can be recovered from any such pair by inverting the recurrence).

Let us consider now the sequence of vectors \( v_0, v_1, v_3, v_5, \ldots \). These vectors are sorted around the origin in the first quadrant (see Fig. 4), and any successive pair of vectors in the sequence forms a basis for \( T_n \).

Consider now the polygonal area \( A \) bounded between the positive \( x \) and \( y \) axis and the polygon line that connects the \( v_k \) points. It is the union of the triangles formed between any two successive vectors in the sequence. By the basic property of bases in lattices, these triangles cannot contain other lattice points and thus their union \( A \) cannot contain other lattice points from \( T_n \).

To complete the proof that \( P(T_n) > P_n/\sqrt{5} \), it is sufficient to show that the area \( H \) bounded between the positive \( x \) and \( y \) axis and the hyperbola \((x+1)(y+1) - P_n/\sqrt{5} \) is strictly contained within \( A \), and thus \( H \) cannot contain any lattice points from \( T_n \) (except the origin). Consequently, any rectangle \( R(v) \) which contains another occurrence of label 0 satisfies \( I(v) = (x+1)(y+1) > P_n/\sqrt{5} \).

By using convexity arguments, we can show that \( H \) is contained in \( A \) by proving that all the corners of \( A \) (except the origin) are outside \( H \). By using convexity arguments, we can show that \( H \) is contained in \( A \) by proving that all the corners of \( A \) (except the origin) are outside \( H \). By using convexity arguments, we can show that \( H \) is contained in \( A \) by proving that all the corners of \( A \) (except the origin) are outside \( H \). By using convexity arguments, we can show that \( H \) is contained in \( A \) by proving that all the corners of \( A \) (except the origin) are outside \( H \).

**Theorem 5:** \( \max_v D_n(v) \leq (3/2)n \).

**Proof (Sketch):** We use a cut-and-paste argument to show that \( \max_v D_n(v) \leq \max_v D_{n-1}(v) + 1 \).

The desired result follows from this inequality by induction.

**Lemma 6:** There is a \( v = (i,j) \) with \( 0 \leq i, j \leq n-1 \) which achieves the maximal load difference for any \( R(v) \).

**Proof:** Let \( R(v) \) be an arbitrary rectangle. Since all the labels occur exactly once along any horizontal or vertical segment of length \( F_n \), we can eliminate arbitrarily many such segments without changing the load difference between processors in the remaining shape. We can thus reduce \( i \) and \( j \) mod \( F_n \) and obtain \( 0 \leq i, j < F_n \). To reduce this range further, we note that by replacing \( i \) by \( F_{n-1} \) (or \( j \) by \( F_{n-2} \)), we change the direction of the load difference but not its absolute value. We can thus assume without loss of generality that \( 0 \leq i, j < F_{n}/2 < F_{n-1} \).

In the 13 processor example in Fig. 5, the maximal load difference already occurs for rectangles which are at most 6 x 6 in size. However, the inductive proof is made simpler by considering rectangles which are at most 8 x 8 in size.

We next consider the possible load difference between processors which are horizontally adjacent (such as 0 and 5, or 7 and 12 in our example):

**Lemma 3:** For any \( R(v) \) with \( 0 \leq i, j < F_n \), the load difference between horizontally adjacent processors is at most 1.

**Proof:** Since any occurrence of one label in \( R(v) \) is immediately followed by an occurrence of the other label, the difference can only be caused by a vertical boundary of \( R(v) \) which separates the two labels. By the condition on \( j \), this can happen at most once along each boundary since labels can repeat only after \( F_{n} \) vertical steps.

**Definition:** The bridge set \( S_n \) is the set of \( F_{n-2} \) labels defined by \( F_{n-2} \leq k < F_{n-1} \).

**Example:** The bridge set in our 13 processor example is \( \{3, 4, 5, 6, 7\} \).

**Lemma 8:** Any label \( 0 \leq k < F_n \) is either in \( S_n \) or has a left neighbour in \( S_n \) or has a right neighbour in \( S_n \).

**Proof:** It is easy to show that the right neighbour of label \( k \) in a Fibonacci labelled grid with \( F_n \) labels is \( k + F_{n-2} \) (mod \( F_n \)) and that the left neighbour of \( k \) is \( k - F_{n-2} \) (mod \( F_n \)). For any \( 0 \leq k < F_{n-3} \), the right neighbour of \( k \) is in \( S_n \) since \( 0 + F_{n-2} = F_{n-2} \) and \( F_{n-3} + F_{n-2} = F_{n-1} \). For any \( F_{n-3} \leq k < F_{n-1} \), \( k \) is in \( S_n \) by definition. For any \( F_{n-1} \leq k < F_n \), the left neighbour of \( k \) is in \( S_n \) since \( F_{n-1} \) and \( F_{n-2} - F_{n-3} = F_{n-4} \) for any \( F_{n-3} \leq k < F_{n-1} \).

Q.E.D.
Lemma 9: The load difference between any two processors cannot exceed the load difference between any two processors in the bridge set $S_n$ plus 2.

Proof: An immediate consequence of the previous two lemmas. Q.E.D.

Example: The load difference between processors 1 and 9 is bounded by the load difference between their neighbours 6 and 4 (which are in the bridge set) plus 2.

To study the possible loads on the $F_{n-2}$ labels in the bridge set $S_n$ for rectangles $R(v)$ with $0 \leq i, j < F_{n-1}$, we erase all the other labels in the square. The partially filled columns which result from this step are illustrated in Fig. 6.

The next step in the proof is to squeeze all the labels to the left as much as possible. In our example the result is depicted in Fig. 7, which demonstrates that columns can be joined but need never be broken (the labels retain their original column indices to facilitate inspection). By subtracting 3 from all these labels, we get Fig. 8 whose label structure corresponds to a Fibonacci labelled grid with 5 labels. The trickiest part of our proof is to show that this is the case for all $n$.

Lemma 10: When all the labels except those in $S_n$ are erased in $0 \leq i, j < F_{n-1}$, the labels are squeezed to the left, and $F_{n-3}$ is subtracted from their values, the region becomes part of a Fibonacci labelled grid with $F_{n-2}$ labels and the relationship between column indices is preserved.

Proof: The proof is based on an elaborate case analysis which is omitted in this extended abstract.

Lemma 11: The maximal load difference between processors from $S_n$ in a Fibonacci labelled grid with $F_n$ labels is at most $\max_{v} D_{n-2}(v) + 1$.

Proof: Consider any rectangle $R(v)$ with $0 \leq i, j < F_{n-1}$ placed on the grid of labels from $S_n$ before they are squeezed. The set of labels in $R(v)$ is squeezed into the shape of a rectangle whose right boundary may contain a jog of width 1. We can eliminate this jog by dropping the protruding labels or adding the missing labels in the last column, whichever affects fewer labels. Since the load on each label in $S_n$ can change by at most 1 when the shape is changed into a rectangle, the original load difference between labels in $S_n$ cannot exceed $\max_{v} D_{n-2}(v) + 1$. Q.E.D.

To conclude the proof of the theorem, we combine Lemmas 9 and 11 to obtain

$$\max_{v} D_{n}(v) \leq \max_{v} D_{n-2} + 2 + \text{which yields the desired result.}$$

Q.E.D.

Corollary 12: The maximal load imbalance in a polyomorphically array with $N$ processors is at most $(3/2) \log M$.

Proof: $\max_{v} D_{n}(v) \leq 3n = (3/2) \log F_{2n} = (3/2) \log M$. Q.E.D.

6. Other Applications

In this section we briefly describe another application of Fibonacci lattices which is motivated by computer vision. Given a large array of $A$ pixels, we would like to locate all the objects whose area exceeds a certain threshold $B$ (i.e., those objects which are not likely to be noise). Each pixel can be either white or black, and objects are defined as (untitled) rectangular black regions on a white background. The basic operation in this model is probing, which determines the color of the probed pixel. The goal is to locate all the objects in a minimum of probes.

A random probing pattern is likely to locate an object of size $B$ in about $A/B$ probes, but in the worst case the number of probes can be as high as $A-B+1$. In this section we describe a deterministic probing pattern which is guaranteed to find the object in at most $O((A/B) \log B)$ probes.

The probing pattern is based on the fact that any sufficiently large rectangle in a Fibonacci labelled grid must contain at least one occurrence of each label. This result is essentially a converse of the overlap theorem, whose proof is based on the uniformity result:

Theorem 13: If $v = (i,j)$ satisfies $(i+j)(j+1) > (3n/2)F_n$ in a Fibonacci labelled grid of order $n$, then $Q_n(v,k) \geq 1$ for all $k$.

Proof: Since the number of distinct labels is $F_n$ while $R(v)$ contains more than $(3n/2)F_n$ points, $Q_n(v,k) > 3n/2$ for at least one $k$. By Theorem 5, the difference between the number of occurrences of any two labels is at most $3n/2$, and thus $Q_n(v,k) \geq 1$ for all $k$. Q.E.D.

Corollary 14: Any rectangle with more than $(3n/2)F_n$ points (placed not necessarily at the origin) contains at least one representative from $T_n$.

To use this result in our application, we choose the maximal $n$ such that $(3n/2)F_n \leq B$ and probe the array at all the pixels which are in $T_n$. The number of probes is approximately $A/F_n = (3n/2)A/B$. Since $n$ is at most $\log B$, the number of probes is $O((A/B) \log B)$.

This probing technique is applicable even when the shape of the region is not a rectangle. A particularly useful case is when the region is a horizontal or vertical elliptic "blob". Since any such region contains an untilted rectangle whose area is $\pi/2$ times smaller, we can locate the "blob" by locating the inscribed rectangle in essentially the same number of probes.

Acknowledgements: We would like to thank Ehud Shapiro for motivating this research and Benny Chor, Charles Leiserson and Ron Rivest for introducing us to the fascinating world of Fibonacci lattices.
REFERENCES


<table>
<thead>
<tr>
<th>Index</th>
<th>Width</th>
<th>Length</th>
<th>No of Processors - M</th>
<th>No Overlap size - O</th>
<th>O/M</th>
<th>Max Imbalance</th>
<th>Conjec. Max Imbalance</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
<td>21</td>
<td>15</td>
<td>0.714286</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>11</td>
<td>55</td>
<td>35</td>
<td>0.630364</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>29</td>
<td>377</td>
<td>195</td>
<td>0.517241</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>47</td>
<td>987</td>
<td>493</td>
<td>0.489362</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>123</td>
<td>6765</td>
<td>3135</td>
<td>0.463415</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>89</td>
<td>190</td>
<td>17711</td>
<td>8089</td>
<td>0.457290</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>233</td>
<td>521</td>
<td>121393</td>
<td>54755</td>
<td>0.451058</td>
<td>22</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>377</td>
<td>945</td>
<td>317811</td>
<td>142883</td>
<td>0.449858</td>
<td>35</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>987</td>
<td>2207</td>
<td>2178309</td>
<td>970143</td>
<td>0.449320</td>
<td>31</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>1597</td>
<td>3571</td>
<td>5702887</td>
<td>2553603</td>
<td>0.441774</td>
<td>34</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>4181</td>
<td>9349</td>
<td>39088109</td>
<td>17489123</td>
<td>0.441428</td>
<td>40</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>6765</td>
<td>15127</td>
<td>102334156</td>
<td>46778755</td>
<td>0.447340</td>
<td>43</td>
<td>13</td>
</tr>
<tr>
<td>22</td>
<td>1771</td>
<td>3903</td>
<td>701408733</td>
<td>313714943</td>
<td>0.447264</td>
<td>49</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 1: Polymorphic Arrays
Figure 1

Figure 2

Figure 3