Layout optimisation for yield enhancement in on-chip-VLSI/WSI parallel processing

P. Mazumder

Indexing terms: Very large scale integration, Optimisation

Abstract: The paper investigates the layout optimisation problem for processor-array networks. If an appropriate shape geometry is selected for the processors, a specific interconnection network can be area-efficiently mapped on a VLSI/WSI chip to maximise the chip yield, operational reliability and circuit performance. A formal technique of cellular layout by polyomino tiles is proposed, with application to mapping a variety of processor geometries onto specific array networks. The layout algorithms are expressed in a new notational language, which is amenable to cellular layout in contrast to classical procedural languages. The layout technique is illustrated with both well known parallel-processing array networks and a new fault-tolerant square mesh with reconfigurable processors and interconnect. The square mesh with redundant processors provides high yield and operational reliability.

1 Introduction

Two-dimensional array networks in which communication between two processors occurs through their adjoining neighbours have been shown to be highly suitable for on-chip parallel processing in CMOS VLSI/WSI applications [13]. In the literature, a large number of important classes of computational problems (e.g. matrix multiplication, LU decomposition, sorting, FFT etc. [15, 23]) have been mapped into a VLSI chip utilising the various types of square- and hexagonal-array networks. This paper addresses the problems of optimisation of processor geometry and the associated floorplan design of such array networks within VLSI/WSI chips so that overall chip yield can be maximised. Specifically, it identifies a set of processor geometries which can be optimally laid out (in the sense that the total area occupied by the processor modules and the associated interprocessor links is minimum) into a VLSI/WSI chip to describe an array network. While looking into the floorplanning aspects of these array networks, a new fault-tolerant topology of the square arrays has been found. This topology has been shown to improve the yield and operational reliability of the chip.

Traditionally, in studies of layout techniques for parallel-processing networks, the individual processors have been assumed to be square in shape because this allows us to minimise the average delay by distributing the processors uniformly along both the horizontal and the vertical directions [6]. A number of postprocessing algorithms exist in computational-geometry literature to transform layout of one shape (say rectangle) into a square [4]. Such algorithms employ the notions of grid embedding where a rectangular grid is embedded on a square grid. Such techniques have potential practical limitations in the sense that grid embedding expands the area considerably, the dilatation (separation between two adjoining grid points) occurs due to embedding, and multiple crossover occurs to minimise the expansion and dilatation factors. This paper discusses a new layout style, called here cellular layout, that is constructed by hypercellular structures, commonly known as polyominoes,* and it identifies a number of array networks described by these layouts. The purpose of this work is to develop a formal technique of cellular layout by polyomino shaped modules and to demonstrate, given an array network, how to select a suitable processor geometry and how it can be mapped into a VLSI/WSI chip by using the simple layout algorithms discussed here. The layout algorithms are expressed in a new notational language which is more amenable to the layout descriptions than conventional procedural languages. Application is made to several parallel-processing-array networks, including a new fault-tolerant square mesh with reconfigurable processors and interconnect.

2 A formal framework for cellular layout

An alternative strategy is proposed in this Section to tackle the issue of layout transformation. Polyomino cellular structures, originally proposed by von Neumann in connection with cellular automata, are proposed as alternative processor geometries to the square shape. Since individual processing elements comprise a variety of functional blocks, such as CPU, memory, I/O buffer, etc., they can be organised in many different ways and are not always merely rectangular in shape. Also, like square blocks, the polyomino shapes of this section can be distributed uniformly to minimise the propagation delay.

2.1 A notational language for cellular layout

A notational language is introduced here for the construction of layout algorithms. The processors are described as polyomino-shaped modules. To maintain regularity, layout algorithms are designed by transforming modules on a Euclidean plane through linear transformations. Linear transformations have linear as

Paper 8419E (C2, E3), first received 12th October 1988 and in final revised form 5th August 1991

The author is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA

* A polyomino is a figure obtained by joining squares along their edges (i.e. rookwise) only.

well as rotational components. The simple linear component is not discussed here; however the modules are juxtaposed to align their outer edges so that they can be enclosed into a minimum square-chip area. In the resulting layout, a module is completely surrounded by a finite number of other modules and such a pattern is described in this paper as cellular layout, since each module looks like a von Neumann's cellular animal. Each cell on the chip plane is a unit square, as described subsequently. The rotational component of the applied linear transformation is defined as a placement operator and the modules are restricted to orient along a finite direction as permitted by the layout constraint of the design.

Let $\mathbf{E}^2$ be a two-dimensional Euclidean plane representing the chip surface, and $\{x, y\}$ be the basis on $\mathbf{E}^2$ such that the angle subtended by x and y at the point of intersection, called the origin O, is equal to $\pi/2$. Let $X = \{i \mid i = 0, 1, 2, \ldots\}$ and $Y = \{j \mid j = 0, 1, 2, \ldots\}$ be the set of points on the x axis and the y axis such that the distance between any two neighbouring points is unity. The quadratic lattice graph on $\mathbf{E}^2$ is defined by connecting all the ordered pairs of x and y in the Cartesian product $X \times Y$ such that the edges are parallel to the x axis or y axis (Fig. 1). The (i, j)th cell on $\mathbf{E}^2$ is the area

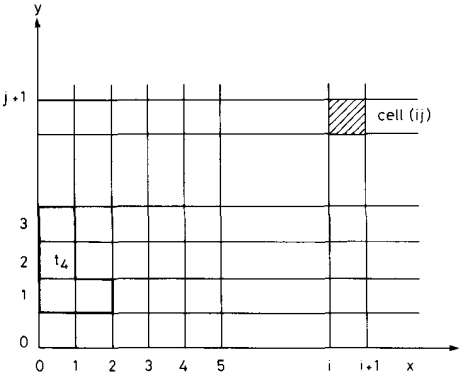

Fig. 1 *Quadratic lattice graph on E^2*

bounded by the edges connecting the ordered pairs $\{(i, j), (i + 1, j), (i, j + 1), (i + 1, j + 1)\}$ and is denoted by cell $(i, j) = x^i y^i$.

Definition 1: A processor module τ_k on a chip plane consists of k rookwise-connected unit squares (the connection scheme is analogous to valid movements of a rook in chess game) [7], called here as cells, and is called a k-omino or a k-cell animal.

Three processor modules are shown in Fig. 2. The first module consists of two cells and resembles the English alphabet I. This is denoted by I_2. Figs. 2*b* and 2*c* are similarly denoted by L_3 and P_5 since they consist of three and five unit cells, respectively. In general, if a processor module resembles an English letter in block capital, τ, then it is represented by τ_k, where k is the number of cells in the module. Each processor module and its orientation is represented by its shape polynomial [5], also known as rook polynomial [22]. If the lower-left corner of the processor module is on the origin, then the shape polynomials of the above three modules are given by $S(I_2) = 1 + x$, $S(L_3) = 1 + x + xy$, and $S(P_5) = 1 + y + xy + y^2 + xy^2$, respectively.

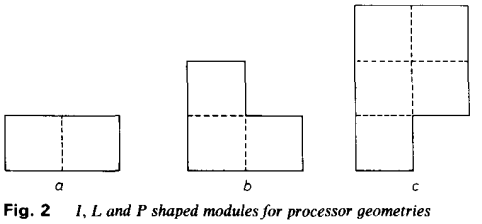

Fig. 2 *I, L and P shaped modules for processor geometries*
(*a*) I_2 (*b*) L_3 (*c*) P_5

Definition 2: The shape polynomial $S(\tau_k)$ of the processor module τ_k whose lower-left corner is on the origin O is given by

$$S(\tau_k) = \sum_{(i,j) \in X \times Y} \gamma_{i,j} x^i y^j$$

where $\gamma_{i,j} = 1$ if the cell $(i, j) \in \tau_k$, otherwise $\gamma_{i,j} = 0$.

The shape polynomial of the module P_5 in Fig. 2*c* is $S(P_5) = 1 + y + y^2 + xy + xy^2$, because $\gamma_{0,0} = \gamma_{0,1} = \gamma_{0,2} = \gamma_{1,1} = \gamma_{1,2} = 1$. It may be noted that x and y elements in a shape polynomial of a module can be interchanged to define a new shape polynomial, which will represent the same module in a different orientation. The new shape polynomial is called here as the conjugate of the original shape polynomial.

Definition 3: The conjugate of shape polynomial $S(\tau_k)$ is defined as a shape polynomial,

$$S^*(\tau_k) = \sum_{(i,j) \in X \times Y} \gamma_{i,j} x^j y^i$$

and is obtained by permuting x and y elements of $S(\tau_k)$.

The conjugate of $S(P_5)$ will be given by $S^*(P_5) = 1 + x + x^2 + xy + x^2y$. Since the x and y elements are permuted in the conjugate-shape polynomial, clearly the basis of chip plane is rotated by 90 degrees, and thereby the conjugation operation represents placement of the module in a new orientation. Normally, each processor module can be placed in a different orientation by employing a placement operator.

Definition 4: A placement operator Ψ_ζ rotates a processor module τ_k by an angle Ψ about one of its edges parallel to the ζ axis and the new shape polynomial of τ_k is given by $\Psi_\zeta \bullet S(\tau_k)$.

Properties of Ψ_ζ: If $\Psi_{\zeta 1}$ and $\Psi_{\zeta 2}$ are two placement operators applied on τ_k, then

$$(\Psi_{\zeta 1}\Psi_{\zeta 2}) \bullet S(\tau_k) = \Psi_{\zeta 1} \bullet \{\Psi_{\zeta 2} \bullet S(\tau_k)\}$$
$$= \Psi_{\zeta 2} \bullet \{\Psi_{\zeta 1} \bullet S(\tau_k)\}$$

Thus the placement operation is commutative.

If $\Psi = 2m\pi$ (where $m \in \{1, 2, \ldots\}$), then $(2m\pi)_\zeta \bullet S(\tau_k) = S(\tau_k)$ and $\iota = (2m\pi)_\zeta$ is called the identity operator. Intuitively, an identity operator applies a linear translation on a processor module, with no rotational transformation.

If $\Psi = \phi$, then $\phi \bullet S(\tau_k) = \phi$ and $\Psi = \phi$ is called the null operator.

Since the interconnections in a VLSI layout usually run either vertically or horizontally parallel to the edges of the chip, the modules are oriented such that their edges are always parallel to the basis $\{x, y\}$.

Definition 5: The permissible orientations of a processor module τ_k on $\mathbf{E}^2$ are due to the following placement operators applied in any arbitrary sequence: $(m\pi)_x$, $(m\pi)_y$, $(m\pi)_z$ and λ_z, where $\lambda_z \bullet S(\tau_k) = S^*(\tau_k)$ and z is orthogonal to chip plane.

The transformations due to $(m\pi)_x$ and $(m\pi)_y$ are called reflections or mirroring about the x axis and the y axis, respectively. The transformation due to $(m\pi)_z$ corresponds to a sequel of two reflections about x and y axes, or equivalently an anticlockwise rotation of 180° about z axis. The conjugation operation λ_z is a composite operation of anticlockwise rotation of the module by 90° about the z axis and then a reflection about a y axis, i.e. x and y elements in the shape polynomial $S(\tau_k)$ are swapped to yield its conjugate shape polynomial $S^*(\tau_k)$.

Note that a k-omino module τ_k has at most eight distinct shape polynomials corresponding to all permissible orientations. To justify this, it can be seen that, corresponding to a shape polynomial $S(\tau_k)$ of τ_k, three more shape polynomials can be generated by applying the placement operators π_x, π_y, and π_z on $S(\tau_k)$. By permuting the x and y elements, $S^*(\tau_k)$ can be derived and the above operators can be applied to get three more shape polynomials. Depending on the symmetry of τ_k about x, y or z axes, there will be eight, four, two or one distinct shape polynomials. Fig. 3 shows the shape polynomials of a module, designated as P_5, corresponding to all its permissible orientations. Clearly, L_3 has four distinct shape polynomials, I_2 has 2, and τ_1 that consists of only one cell is isotropic (i.e. one permissible orientation).

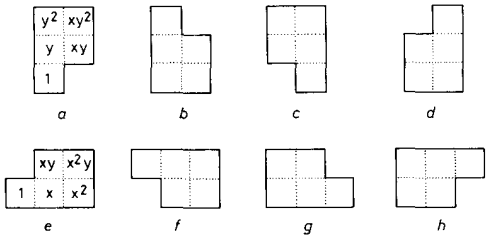

Fig. 3 *All permissible orientations of* P_5

a $S(P_5)$
b $\pi_x \bullet S(P_5)$
c $\pi_y \bullet S(P_5)$
d $\pi_z \bullet S(P_5)$
e $\lambda_z \bullet S(P_5) = S^*(P_5)$
f $\pi_x \bullet S^*(P_5)$
g $\pi_y \bullet S^*(P_5)$
h $\pi_z \bullet S^*(P_5)$

Using the above definitions, the layout described by polyomino-shaped modules can be easily described as discussed in the rest of this section. Since a base cell (say, τ_1) can have at most eight neighbours (Moore's neighbourhood) which physically adjoin it, each base cell should be described by its own shape polynomial and the shape polynomials of its eight neighbours.

Definition 6: The cellular layout of a set of modules $R = \bigcup_{i=1}^{n} R_i$ is its shape polynomial S, represented as a spatial distribution of the shape polynomials S_i (or the permissible orientations) of each of the constituent modules, R_i.

Assuming that two of the edges of R constitute the basis $\{x, y\}$ such that the lower-left corner of R is the origin, the spatial distributions of S_i can be expressed as a matrix, such that for all R_i and R_j, if R_j is adjacent to R_i horizontally, vertically or diagonally, S_i and S_j are also similarly adjacent in the matrix.

If $R = \bigcup_{i=1}^{9} R_i$ (Fig. 4) is a planar layout describing a Moore's neighbourhood structure [8] of nine modules,

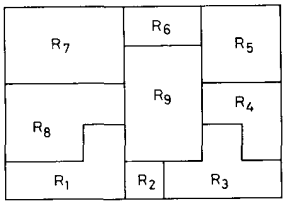

Fig. 4 *Planar layout representing a Moore's neighbourhood*

where the base module R_9 is surrounded by eight other modules, then

$$S = \begin{bmatrix} S_7 & S_6 & S_5 \\ S_8 & S_9 & S_4 \\ S_1 & S_2 & S_3 \end{bmatrix} \tag{1}$$

The right-hand side is called the layout matrix and eqn. 1 is called the layout rule, ε. The layout matrix denotes the spatial distribution of the processors in the layout, and the layout rule defines the adjacency relationship between the modules.

Operations on S

As discussed earlier, the placement operators rotate the modules in different permissible orientations. They can be applied repetitively to construct layout hierarchically. When an operator is applied on an layout matrix S all the modules within the layout are also rotated accordingly. This property of the placement operators is described here by the following four operations on S (where S is given by eqn. 1):

Reflection about x axis:

$$\pi_x \bullet S = \begin{bmatrix} \pi_x \bullet S_1 & \pi_x \bullet S_2 & \pi_x \bullet S_3 \\ \pi_x \bullet S_8 & \pi_x \bullet S_9 & \pi_x \bullet S_4 \\ \pi_x \bullet S_7 & \pi_x \bullet S_6 & \pi_x \bullet S_5 \end{bmatrix} \tag{2}$$

Since the operator π_x produces a mirror image of S about the x axis, the first and the third rows in eqn. 1 have been swapped in eqn. 2.

Reflection about y axis:

$$\pi_y \bullet S = \begin{bmatrix} \pi_y \bullet S_5 & \pi_y \bullet S_6 & \pi_y \bullet S_7 \\ \pi_y \bullet S_4 & \pi_y \bullet S_9 & \pi_y \bullet S_8 \\ \pi_y \bullet S_3 & \pi_y \bullet S_2 & \pi_y \bullet S_1 \end{bmatrix} \tag{3}$$

Since the operator π_y produces a mirror image of S about the y axis, the first and the third columns in eqn. 1 have been interchanged in eqn. 3.

Rotation about z axis:

$$\pi_z \bullet S = \begin{bmatrix} \pi_z \bullet S_3 & \pi_z \bullet S_2 & \pi_z \bullet S_1 \\ \pi_z \bullet S_4 & \pi_z \bullet S_9 & \pi_z \bullet S_8 \\ \pi_z \bullet S_5 & \pi_z \bullet S_6 & \pi_z \bullet S_7 \end{bmatrix} \tag{4}$$

Since the operator π_z rotates the module by 180°, the first and the third rows in eqn. 1 have been swapped, and then the first and the third columns in the modified matrix are interchanged in eqn. 4.

Conjugate operation:

$$\lambda_z \bullet \boldsymbol{S} = \begin{bmatrix} \lambda_z \bullet \boldsymbol{S}_5 & \lambda_z \bullet \boldsymbol{S}_4 & \lambda_z \bullet \boldsymbol{S}_3 \\ \lambda_z \bullet \boldsymbol{S}_6 & \lambda_z \bullet \boldsymbol{S}_9 & \lambda_z \bullet \boldsymbol{S}_2 \\ \lambda_z \bullet \boldsymbol{S}_7 & \lambda_z \bullet \boldsymbol{S}_8 & \lambda_z \bullet \boldsymbol{S}_1 \end{bmatrix} \quad (5)$$

Since the operator λ_z rotates the module by 90°, the first and the third columns in eqn. 1 have been swapped, and then the modified matrix is transposed in eqn. 4.

Fig. 5 shows the effect of application of above placement operators on the shape polynomial $\boldsymbol{S}$ of the

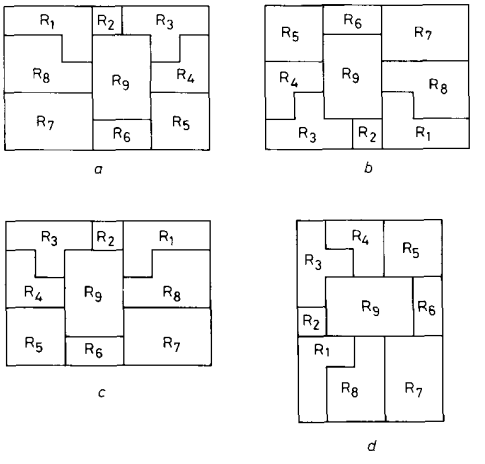

Fig. 5 *Application of placement operators on Moore's neighbourhood*
(a) $\pi_x \bullet \boldsymbol{S}$ *(b)* $\pi_y \bullet \boldsymbol{S}$ *(c)* $\pi_z \bullet \boldsymbol{S}$ *(d)* $\lambda_z \bullet \boldsymbol{S}$

Moore's neighbourhood in Fig. 4. Note that in Fig. 5*a* the bottom (top) row corresponds to the top (bottom) row of Fig. 4, and this corresponds to mirroring about x-axis. Similarly, Fig. 5*b*, 5*c* and 5*d* illustrate the other operations as expressed by eqns. 3, 4 and 5, respectively.

2.2 Mosaic layout constructed by polyomino modules

The cellular layout of a repetitive-module structure is called a mosaic. For a VLSI/WSI implementation, such regularly structured layouts, described by iterative logic arrays, cellular networks, systolic arrays etc. are very easily constructed. These layouts are constructed by replicating a single type module such that there is no gap between the successive modules. Formally, a mosaic layout can be defined as:

Definition 7: A layout of τ_k under the layout rule ε is called a mosaic $\boldsymbol{M}_\varepsilon(\tau_k)$ iff for every point within the layout, there exists an instance of the module τ_k denoted as τ_k^i, such that for all $j \neq i$, $\tau_k^i \cap \tau_k^j = \phi$.

A mosaic layout constructed by I_2 and L_3 will be discussed here. For other polyomino structures, the layout algorithms can be easily constructed as shown later. Other researchers, namely Wong and Liu [11], have also considered general-purpose floorplanning using L-shape modules.

Layout algorithms

Fig. 6 shows a mosaic layout made of I_2, commonly known as dominoes. The powerful array-interconnection network described by this layout is discussed in Section 3. The layout can be recursively formed by the following algorithm.

Algorithm 2: The layout of I_2 describes a mosaic $M_{\varepsilon 1}(I_2)$ under the following layout rule (say $\varepsilon 1$):

$$\boldsymbol{S}_n(I_2) = \begin{bmatrix} \phi & \boldsymbol{S}(I_{2n}) & \phi \\ \lambda_z \bullet \boldsymbol{S}(I_{2n-2}) & \boldsymbol{S}_{n-1}(I_2) & \lambda_z \bullet \boldsymbol{S}(I_{2n-2}) \\ \phi & \boldsymbol{S}(I_{2n}) & \phi \end{bmatrix} \quad (6)$$

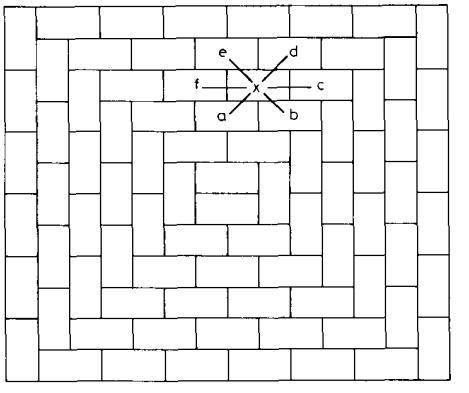

Fig. 6 *Mosaic layout described by* I_2

where $n \geq 1$ is the level of layout, $\boldsymbol{S}_0(I_2) = \phi$, and $I_{2n} = \sum_{i=0}^{2n-1} x^i$ denotes a linear concatenation of $2n$ cells.

In the above algorithm, the nth level of the layout consists of $4n - 2$ dominoes, n dominoes in the top row, n dominoes in the bottom row, $n - 1$ dominoes in the leftmost column, and $n - 1$ dominoes in the right most column. This describes an annular square region having an inner edge of length $n - 1$ and an outer edge of length n. It may be observed that for $n = 1$, $\boldsymbol{S}_1(I_2)$ will consist of two dominoes stacked horizontally. By simple induction, it can be seen that the algorithm describes the layout shown in Fig. 6, which shows $\boldsymbol{S}_6(I_2)$.

The next algorithm describes a mosaic layout which consists of L_3 modules, commonly known as trominoes. As it is pointed out below, hexomino modules having P_6 shape also describe a similar mosaic layout. The layout can be recursively constructed as the following:

Algorithm 3: The layout of L_3 describes a mosaic $\boldsymbol{M}_{\varepsilon 2}(L_3)$ under the following layout rule (say $\varepsilon 2$):

$$\boldsymbol{S}_n(L_3) = \begin{bmatrix} \pi_x \bullet \boldsymbol{S}_{n-1}(L_3) & \phi & \boldsymbol{S}_{n-1}(L_3) \\ \phi & \boldsymbol{S}(L_3) & \phi \\ \boldsymbol{S}_{n-1}(L_3) & \phi & \pi_y \bullet \boldsymbol{S}_{n-1}(L_3) \end{bmatrix} \quad (7)$$

where $n > 1$ is called the level of layout and $\boldsymbol{S}_1(L_3) = \boldsymbol{S}(L_3)$ is the shape polynomial of L_3.

Referring to Fig. 7, in order that a layout having the shape polynomial $\boldsymbol{S}_m(L_3)$ is a mosaic by layout rule $\varepsilon 2$, it is required that $h_2 = 2^m h(L_3) - 1$, $v_2 = 2^m v(L_3) - 1$, $h_3 = h(L_3)$ and $v_3 = v(L_3)$.

Thus for example if $n = 3$, then the mosaic layout $\boldsymbol{M}_{\varepsilon 2}(L_3)$ will be given by

$$\boldsymbol{S}_3(L_3) = \begin{bmatrix} \pi_x \bullet \boldsymbol{S}_2(L_3) & \phi & \boldsymbol{S}_2(L_3) \\ \phi & \boldsymbol{S}(L_3) & \phi \\ \boldsymbol{S}_2(L_3) & \phi & \pi_y \bullet \boldsymbol{S}_2(L_3) \end{bmatrix}$$

By applying eqns. 2, 3 and 4, the shape polynomial of the mosaic layout is given by

$$S_3(L_3) = \begin{bmatrix} \pi_x \bullet \begin{bmatrix} \pi_x & \phi & \iota \\ \phi & \iota & \phi \\ \iota & \phi & \pi_y \end{bmatrix} & \phi & \iota \bullet \begin{bmatrix} \pi_x & \phi & \iota \\ \phi & \iota & \phi \\ \iota & \phi & \pi_y \end{bmatrix} \\ \phi & \iota & \phi \\ \iota \bullet \begin{bmatrix} \pi_x & \phi & \iota \\ \phi & \iota & \phi \\ \iota & \pi_y & \pi_y \end{bmatrix} & \phi & \pi_y \bullet \begin{bmatrix} \pi_x & \phi & \iota \\ \phi & \iota & \phi \\ \iota & \phi & \pi_y \end{bmatrix} \end{bmatrix} \bullet S(L_3)$$

$$= \begin{bmatrix} \begin{bmatrix} \pi_x & \phi & \pi_x \\ \phi & \pi_x & \phi \\ \iota & \phi & \pi_x \end{bmatrix} & \phi & \begin{bmatrix} \pi_x & \phi & \iota \\ \phi & \iota & \phi \\ \iota & \phi & \pi_y \end{bmatrix} \\ \phi & \iota & \phi \\ \begin{bmatrix} \pi_x & \phi & \iota \\ \phi & \iota & \phi \\ \iota & \phi & \pi_y \end{bmatrix} & \phi & \begin{bmatrix} \pi_y & \phi & \pi_x \\ \phi & \pi_y & \phi \\ \iota & \phi & \pi_y \end{bmatrix} \end{bmatrix} \bullet S(L_3) \quad (8)$$

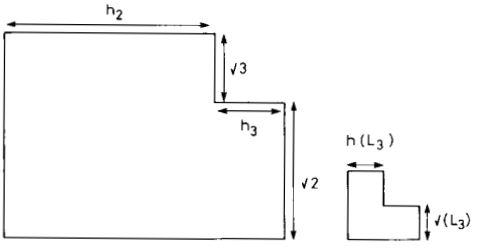

Fig. 7 *Mosaic layout described by L_3*

By applying the properties of the placement operators, the layout of $S_3(L_3)$ can be obtained from eqn. 8 as shown in Fig. 8. It can be seen that the diagonal (lower-left to upper-right) modules are placed without rotation by the identity operator ι, other modules are rotated by the corresponding placement operator in the matrix of eqn. 8, and the null operator ϕ indicates no occurrence of a module in the corresponding position.

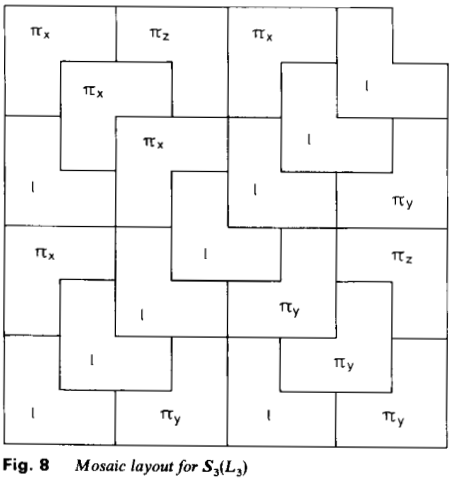

Fig. 8 *Mosaic layout for $S_3(L_3)$*

2.3 Computational power, chip area and layout cost

The computational power of a mosaic layout can be roughly measured by a parameter called structural complexity. Since each processor has equal and constant computational power, a mosaic has a total computational power which is proportional to an exponential or a polynomial function of the level of layout n. The structural complexity of the shape polynomial $S_n(\tau_k)$ is the total number of constituent k-ominoes and is given by $N_n(\tau_k)$. Structural complexity is denoted with respect to n, since the level of layout indicates the number of steps required to generate a layout of certain computational power. From the recursive definition of $\varepsilon 1$, $N_n(I_2) = 4n + N_{n-1}(I_2) - 2$, with $N_1(I_2) = 2$. Thus the structural complexity of $S_n(I_2)$ is given by $N_n(I_2) = 4\sum_{i=2}^{n} i + N_1(I_2) - 2(n-1) = 2n^2$. Since $N_n(L_3) = 4N_{n-1}(L_3) + 1$ with $N_1(L_3) = 1$, the structural complexity of $S_n(L_3)$ is given by $N_n(L_3) = (1/3)(4^n - 1)$.

The square hull of a mosaic is the smallest-size square in which the mosaic can be enclosed and is a measure of the minimum chip size required to fabricate the mosaic layout. In order that the mosaic can be efficiently laid out within a chip, the mosaic should be approximately square in shape. The mosaic $M_{\varepsilon 1}(I_2)$ has a square hull of side $\sqrt{(2)}n$. The mosaic $M_{\varepsilon 2}(L_3)$ has a square hull of side $2^n/\sqrt{(3)}$, assuming $|S(L_3)| = 1$ (i.e. the area of the tile L_3 is equal to unity).

2.4 Layout cost

The regularity of the layout allows us to make a mosaic arrangement of the processors by laying out fewer processors than the total number of processors in the mosaic. Since each processor has a bounded (usually less than 8) number of interprocessor links, the placement and the associated connection cost of each processor is $O(1)$. The overall layout cost of the mosaic is defined as the total number of processors actually laid out. In algorithm of $\varepsilon 1$, since each level of recursion of the layout involves adding the top and the bottom rows and the rightmost and the leftmost columns and costs $4 \times O(1)$, except when $n = 1$ that costs $2 \times O(1)$. Thus to construct $M_{\varepsilon 1}(I_2)$, it costs at most $(4n - 2) \times O(1) = O\sqrt{\{N_n(I_2)\}}$. In the algorithm of $\varepsilon 2$, at each level of recursion of the layout $\varepsilon 2$ costs $5 \times O(1)$, and to construct $M_{\varepsilon 2}(L_3)$, a cost of $5/2 \log (3N_n(L_3) + 1) = O\{\log N_n(L_3)\}$ is required.

3 Array interconnection networks

To evaluate the importance of the layout algorithms described above, it is necessary to identify the interconnection networks described by the mosaic layout. A communication link can be set up between two modules in the mosaic if they are adjacent to each other. The interconnection networks described by the layouts $M_{\varepsilon 1}(I_2)$ and $M_{\varepsilon 2}(L_3)$ are identified in this Section.

In Fig. 6, it can be seen that every non-peripheral domino is surrounded by six other dominoes except the innermost two dominoes corresponding to layout level $n = 1$. The graph due to $M_{\varepsilon 1}(I_2)$ is 6-regular (assuming that the innermost two dominoes as a single node) and corresponds to the communication geometries of hexagonal arrays as shown in Fig. 9. It can be seen in Fig. 6 that processor x is surrounded by processors *a–f*, and that the resulting graph is thereby a hexagonal array. These arrays are suitable for various computational problems like matrix multiplication, LU decomposition, 2-D convolution, FFT etc. [15].

In Fig. 8 it can be seen that every non-peripheral tromino is surrounded by four, five or six other modules depending on its position in the layout. The graph

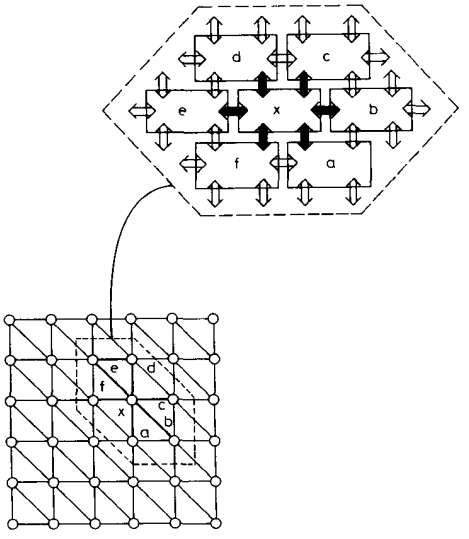

Fig. 9 *Hexagonal arrays described by* $M_{\ell 1}(I_2)$

described by $M_{\ell 2}(L_3)$ denotes communication geometries of square arrays with redundant cells as shown in Fig. 10. In the matrix of eqn. 8 the five non-peripheral diagonal placement operators (except ϕ) correspond to the loca-

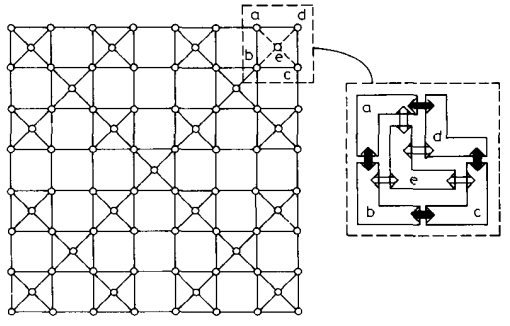

Fig. 10 *Fault-tolerant meshes described by* $M_{\ell 2}(L_3)$

tion of redundant cells. It may be noted that the communication graph in Fig. 10 corresponds to $S_4(L_3)$ and hence it has $5 * 4 + 1 = 21$ redundant cells corresponding to four $S_3(L_3)$ and the centre $S(L_3)$ in eqn. 7. This communication topology is highly suitable for fault-tolerant parallel processing and is particularly relevant for high-yield VLSI/WSI fabrication. A defective processor can be bypassed as shown in Fig. 11, where the central processor is utilised to reconstruct the square mesh.

The presence of redundant processors ameliorates the chip yield considerably in VLSI/WSI applications. Since the layout of the chip is recursively defined, the yield of $S_n(L_3)$ is defined by the recurrence relation

$$Y_n = Y_{n-1}^4 + 4Y_{n-1}^3 Y_1 C_{n-1}$$

where Y_P is the yield of a single processor and C_{n-1} is the possibility that $S_{n-1}(L_3)$ can be repaired by one spare processor which is added to the centre of $S_n(L_3)$. The values of C_k for different values of $k = 1, 2, 3, \ldots$ can be derived as follows.

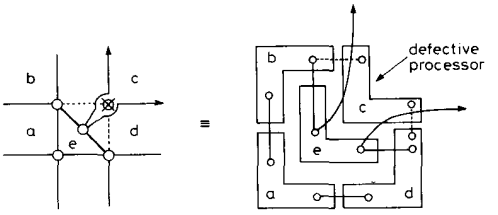

Fig. 11 *Reconfiguration of a defective processor*

From the above equation, it may be seen that $Y_2 = Y_P^4 + 4C_1 Y_P^4$, which is also equal to $Y_P^5 + 5Y_P^4(1 - Y_P)$ enumerating the different ways we can construct a square mesh with five processors. Thus we can find that $C_1 = 1 - Y_P$. To calculate C_2, we find the probability that $S_2(L_3)$ is faulty and can be repaired by a fault-free spare processor in the centre of $S_3(L_3)$. In this case, the corner processor in $S_2(L_3)$ that is connected to the centre processor of $S_3(L_3)$ should be faulty, and one of the four other processors in $S_2(L_3)$ is faulty. Therefore, we obtain $C_2 = (1 - Y_P)4Y_P^3(1 - Y_P)$. Similarly, we can show that $C_3 = (1 - Y_P)Y_2^3 Y_P 3Y_P^2(1 - Y_P) + (1 - Y_P)Y_2^3 C_2 + 3Y_2^2 Y_P C_2^2$, and $C_k \simeq 0$ for $k \geqslant 4$.

In Table 1, the yields of a square mesh with and without redundant processors for the different processor defect probabilities are shown. It may be noted that if the processor failure probability is 10^{-4} or less, the yield remains constant (about 100%) for the fault-tolerant mesh, but the yield of the mesh without redundancy deteriorates rapidly for large-size meshes (greater than 1 K). If the defect probability is greater than 10^{-3}, the fault-tolerant mesh is very effective in improving the yield even for meshes of small sizes. Fig. 12 shows the variation of yield of normal square meshes of different sizes, and Fig. 13 shows the same for fault-tolerant square meshes.

In addition to improving the chip yield, the presence of redundant processors can be utilised to detect and correct concurrent error, and thereby to improve the computational reliability. Since the redundant processors are symmetrically located in the mesh networks (Fig. 10),

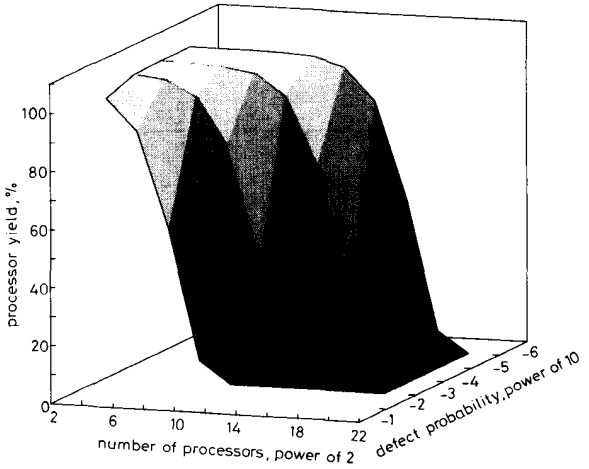

Fig. 12 *Yield against size of a normal square mesh*

they can check the computational correctness of the outer four processors in a round-robin fashion (similar to roving diagnosis [16]) and thereby improve the operational reliability of the mesh networks considerably.

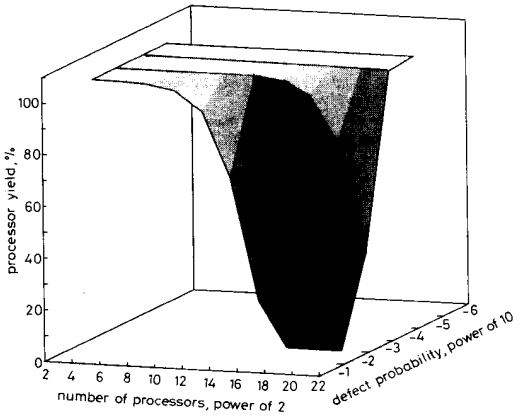

Fig. 13 *Yield against size of a fault-tolerant mesh*

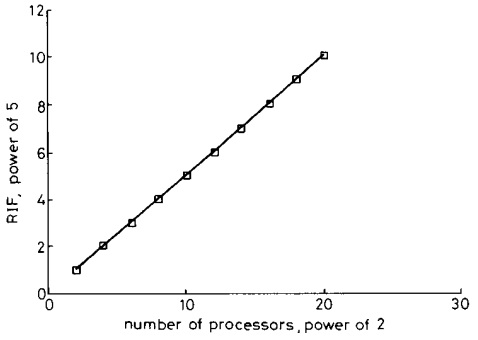

Fig. 14 *Improvement of RIF against mesh size*

Thus the mosaic layout $M_{\ell 2}(L_3)$ is topologically suitable for fault-tolerant computation. Regular mesh structure is not suitable for fault-tolerant computation. If $R_p(t) = e^{-\lambda t}$ is the reliability of each processor (assuming the operational lifetime of the processors is represented by a Poissonian process such that λ is the mean lifetime of the processors), then the reliability of an $(\sqrt{N}) \times (\sqrt{N})$ mesh network without any redundant processor is $R_M(t) = e^{-\lambda N t}$. The reliability of the redundant mesh network $R_{rM}(t) \simeq (5e^{-4\lambda t} - 4e^{-5\lambda t})^{\log_4 N}$. Thus the overall reliability improvement factor RIF is $\text{RIF} = R_{rM}/R_M = (5 - 4e^{-\lambda t})^{\log_4 N}$. For $\lambda t \gg 0$, $\text{RIF} \simeq N^{\log_4 5} \simeq N^{1.16}$, and it can be seen that for a large mesh network the reliability improves significantly as shown in Fig. 14. The various reconfiguration schemes for bypassing faulty processors have been described in Reference 20 and can be easily applied here.

Table 2: Module geometries for array networks

Module notation	Shape polynomial of module	Type of meshes
I_2	$1 + x$	Hexagon
L_3	$1 + x + y$	Square (fault-tolerant)
T_4	$1 + x + x^2 + xy$	Hexagon
Z_4	$1 + x + xy + x^2y$	Hexagon
F_5	$1 + x + xy + xy^2 + x^2y$	Hexagon
P_5	$1 + x + y + xy + y^2$	Hexagon
Y_5	$1 + x + x^2 + x^3 + x^2y$	Hexagon
L_6	$1 + x + x^2 + y + y^2$	Hexagon
P_6	$1 + y + y^2 + y^3 + x + xy$	Square (fault-tolerant)
H_7	$1 + x + x^2 + xy + y^2 + xy^2 + x^2y^2$	Hexagon

Table 2 identifies the layout geometries and shape polynomial of some other polyomino modules which also describe array networks that are highly suitable for

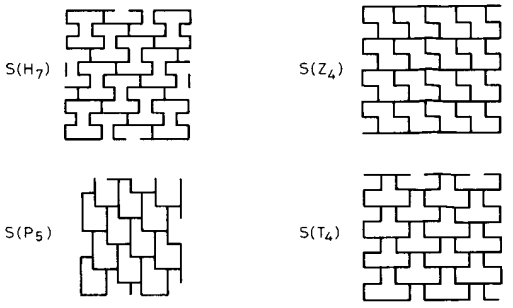

Fig. 15 *Mosaic layouts for hexagonal arrays*
a $S(H_7)$ *b* $S(Z_4)$ *c* $S(P_5)$ *d* $S(T_4)$

VLSI/WSI parallel processing. The planar covering properties of these modules have been pictorially represented by Fig. 15 and from these geometrical arrangements, the layout algorithms can be easily designed employing the concept developed in this Section. Here, only the polyomino modules have been identified, which can be utilised to describe different array networks. The choice of lower-order polyominoes has been restricted by the requirement of distributing the processors uniformly

Table 1: Yield against size of square mesh for different defect probabilities

Defect probability	10^{-2} (%)		10^{-3} (%)		10^{-4} (%)		10^{-5} (%)	
Number of processors	Y	Y_t	Y	Y_t	Y	Y_t	Y	Y_t
4	96.0	99.9	99.6	100	99.9	100	100	100
16	85.2	99.8	98.4	100	99.8	100	99.9	100
64	52.6	99.3	93.8	100	99.3	100	99.9	100
256	7.6	97.3	77.4	100	97.5	100	99.7	100
1024 (=1K)	0.0	89.7	35.9	99.9	90.3	100	99.0	100
4096 (=4K)	0.0	64.6	1.7	99.6	66.4	100	96.0	100
16384 (=16K)	0.0	17.5	0.0	98.3	19.4	100	84.9	100
65536 (=64K)	0.0	0.1	0.0	93.3	0.1	100	51.9	100
262144 (=256K)	0.0	0.0	0.0	75.7	0.0	99.7	7.3	100
1048576 (=1M)	0.0	0.0	0.0	32.8	0.0	98.9	0.0	100

Y = Yield for normal square mesh
Y_t = Yield for fault-tolerant square mesh

both horizontally and vertically so that delay is approximately the same along both directions. Only the square and hexagonal array networks are illustrated, because of their applicability to practical problems such as matrix multiplication, LU decomposition, 2-D convolution, FFT etc.

4 Conclusions

As gargantuan ULSI and GSI chips with several billion transistors are becoming a reality, massively large array processors will be ideal candidates for the exploitation of large silicon surfaces. Fast-parallel-processing topologies such as perfect shuffle networks [21], binary trees and its variants [10, 17], cube-connected cycles [18] and others have been shown to be inadequate for on-chip parallel processing [12]. Interconnection networks having short links, such as nearest-neighbour arrays, have several attractive features: high regularity, intrinsic fault-tolerance capabilities, area efficiency, high chip yield, and scalability of the architecture with the problem size. Several array machines have been experimentally built for general-purpose coprocessor applications (such as ILLIAC IV, HARTS and PAX [1, 19]) and also for special-purpose engines in CAD such as IBM's wire routing machine (WRM) [3], the University of Michigan's hexagonal array machine (HAM) [14], and NEC's torus machine [2]. The task granularity for such array processors varies from simple inner-step product computations as in systolic matrix multiplication and LU decomposition to customised reduced-instruction-set-computations (RISC) as in maze routing engines.

This paper identifies a number of polyomino module shapes for processor geometries which can be mapped cost-efficiently into a VLSI/WSI chip to describe an array network. The overall design strategy can be described as floorplanning for the on-chip-array networks so that the layout area for the processor modules and the associated interconnects is a minimum. A new method of defining the cellular layout has been proposed which uses the notion of linear transformation on the layout matrix. The layout algorithms for array networks are developed; chip area and the layout cost required by these algorithms are also analysed. Finally, a new topology for square meshes has been serendipitously discovered. The presence of redundant processors in the proposed topology has been shown to considerably improve the yield and operational reliability of the chip.

5 Acknowledgments

I would like to thank Profs. Kent Fuchs and Steve Kang at the Coordinated Science Laboratory, University of Illinois, for their criticisms and useful suggestions. This research was supported in part by US Army University Research Initiation programme under grant DAAL-03-87-K-0007, and by NSF under grants MIP 8808978 and MIP 9013092.

6 References

1 BARNES, G.H., *et al.*: 'The ILLIAC IV computer', *IEEE Trans.*, 1968, **C-17**, pp. 746–757
2 SUZUKI, K., *et al.*: 'A hardware maze router with application to interactive rip-up and reroute', *IEEE Trans.*, 1986, **5**, (10), pp. 466–476
3 NAIR, R., *et al.*: 'Global wiring on a wire routing machine'. Proceedings of Design Automation Conference, June 1982, pp. 224–231
4 ALELIUNAS, R., and ROSENBERG, A.L.: 'On embedding rectangular grids in square grids', *IEEE Trans.*, 1982, **C-31**, (9), pp. 907–913
5 BARNES, F.W.: 'Algebraic theory of brick packing 2', *Discrete Math.*, 1982, **42**, pp. 129–144
6 HWANG, K., and BRIGGS, F.: 'Computer architecture and parallel processing' (McGraw-Hill, New York, 1984)
7 GOLOMB, S.W.: 'Polyominoes' (Scribner, 1965)
8 HAYES, B.: 'Computer recreations', *Sci. Am.*, 1984, **250**, (3), pp. 12–21
9 HAYES, J.P.: 'Testing memories for single-cell pattern-sensitive faults', *IEEE Trans.*, 1980, **C-29**, (3), pp. 249–254
10 HOROWITZ, E., and ZORAT, A.: 'The binary tree as an interconnection network: applications to multiprocessor systems and VLSI' (*ibid.*, 1981, **C-30**, (4), pp. 247–253
11 WONG, W.F., and LIU, C.L.: 'A new algorithm for floorplan design'. Proceedings of 23rd Design Automation Conference, June 1986, pp. 101–107
12 MAZUMDER, P.: 'Evaluation of three interconnection networks for CMOS VLSI implementation'. Proceedings of 1986 International Parallel Processing Conference, 1986, pp. 200–207
13 MAZUMDER, P.: 'Evaluation of on-chip static interconnection networks', *IEEE Trans.*, 1987, **C-36**, pp. 365–369
14 VENKATESWARAN, R., and MAZUMDER, P.: 'A hexagonal array machine for multilayer wire routing', *IEEE Trans., Computer-Aided Design*, **9**, (10), pp. 1096–1112
15 MEAD, C.A., and CONWAY, L.A.: 'Introduction to VLSI systems' (Addison Wesley, 1980)
16 MEYER, G., and MASSON, G.: 'An efficient fault diagnosis algorithm for symmetric multiple processor architecture', *IEEE Trans.*, 1978, **C-27**, (11), pp. 1059–1063
17 PATTERSON, D., and DESPIAN, A.: 'X-tree: a structured multiprocessor computer architecture'. Proceedings of IEEE 6th Annual Symposium on Computer Architectures, April 1979, pp. 83–89
18 PREPARATA, F.P., and VUILLEMIN, J.: 'The cube connected cycles: a versatile network for parallel computation'. Proceedings of 20th Annual IEEE Symposium on Foundations of Computer Science, 1979, pp. 140–147
19 SHIN, K.G.: 'HARTS: a distributed real-time architecture', *IEEE Computer*, 1991, pp. 25–35
20 SHUTE, M.J.: 'Fifth generation wafer architecture' (Prentice Hall, London, 1988)
21 STONE, H.S.: 'Parallel processing with the perfect shuffle', *IEEE Trans.*, 1970, **C-20**, (2), pp. 153–161
22 TUCKER, A.: 'Applied combinatorics' (John Wiley & Sons, 1980)
23 ULLMAN, J.D.: 'Computational aspects of VLSI' (Computer Science Press, 1984)