EVALUATING THE COST-EFFECTIVENESS OF SWITCHES IN PROCESSOR ARRAY ARCHITECTURES

Haim Mizrahi and *Israel Koren*

Dept. of Electrical Engineering
Technion - Israel Institute of Technology
Haifa 32000, Israel

ABSTRACT

This paper examines the cost-effectiveness of switches in processor array architectures, where cost is measured by the area needed for implementing the communication grid inherent to these architectures. The CHIP architecture suggested in [1] was chosen as an example for processing arrays containing switching elements interconnecting the processors.

Several variations of the CHIP architecture are considered. We estimate for each the silicon area required and present schemes for mapping certain structures (like binary trees) onto it. We then compare the efficiency of embedding binary trees in CHIP arrays to that in arrays of processors without switches.

In the last part of the paper the fault-tolerance capabilities of CHIP arrays are analyzed and evaluated. First, strategies for overcoming various combinations of faults are suggested and some qualitative results are given. Quantitative comparison is then presented, using as criteria reliability, computational availability and area utilization.

Index Terms - Processor arrays, switching elements, embedding efficiency, area utilization, reliability, computational availability, fault-tolerance.

1. INTRODUCTION

Several architectures for processing arrays which include switching elements connecting the processors have been suggested, e.g., [1], [2], and [3]. The incorporation of switches in a processing array architecture offers a high degree of flexibility, enabling various topologies to be efficiently embedded into the same physical array. In addition, these architectures have dynamic reconfiguration and fault-tolerance capabilities, which allows a high degree of utilization of fault-free processors in the presence of faulty elements.

However, the inclusion of switches in an array increases its area, thus allowing a smaller number of processors to fit into a given silicon area. Our objective in this work is to evaluate the cost-effectiveness of switches in processors arrays. For this evaluation, we have chosen Snyder's "CHIP" (Configurable Highly Parallel Computer)[1], as a representative for architectures of processor arrays with switches, and for comparison, the representative of arrays of processors without switches is that presented in [4].

In the first part of this paper (Section 2), we estimate the area required for the communication grid in several variations of the CHIP architecture. The size of the processors and the width of the internal data busses are the parameters in this evaluation. The area estimation is based on a detailed examination of the logic and topological design of the switches, attempting to optimally integrate them into the data busses.

Special attention is given to the control section of the switch whose contribution to the total area might be substantial. A novel approach suggesting the control of the switches indirectly by the adjacent processors is also introduced and analyzed. In addition, an original CHIP architecture with enhanced communication flexibility is suggested and its advantages in area utilization and fault-tolerance are discussed.

In Section 3, algorithms for embedding binary trees in CHIP arrays are examined and a novel systematic mapping scheme is introduced. Based on the area estimation, a comparison of the efficiency of binary tree embeddings is then made in Section 4, between CHIP arrays and arrays of processors without switches.

In Section 5, the fault-tolerance capability of CHIP arrays is analyzed and evaluated. Strategies for overcoming faults in both the processors and the switches are suggested and some qualitative results are given. Quantitative comparison based on the Markov model suggested in [5], is then presented. The criteria used are reliability, computational availability and area utilization.

2. ESTIMATION OF AREA REQUIREMENTS FOR THE COMMUNICATION NETWORK

The CHIP communication network has two components: data busses and switches (see Figure 2.2). The major part of the area of this network is devoted to the data busses, but the switch size is in fact the dominant factor in the determination of the area. Therefore, we concentrate on the logical and physical structure of various switches and in particular, on the way they are integrated into the busses. The physical width of these busses is clearly a function of m - the number of bits in a bus.

Our layouts follow Mead and Conway [6] NMOS design rules for a process with one metal layer. Data busses are implemented with metal lines, keeping the conservative design rules, i.e., 3λ for both line width, and metal to metal separation. Each line includes short sections of diffusion for the purpose of switching.

The switches in our implementation are passive, i.e., pass transistors which require minimum area. However, delay considerations place a limit on the number of such pass transistors that can be cascaded without an active driver. Adding such buffers not only requires some extra area, but also turns the data lines into unidirectional ones.

To meet the inter-processor communication needs, either full duplex or half duplex protocols may be employed. In the following analysis we assume half duplex busses. If full duplex busses (two separate unidirectional links) are selected, the calculated width of the busses should be doubled.

Data busses run along the sides of the processing elements (PEs). The area for each segment of a bus is

0190-3918/85/0000/0480$01.00 © 1985 IEEE

approximately the length of the PE side times the bus width. For a given size of PE, the way of controlling the area of the communication network is by keeping the bus width minimal. This width is directly determined by the switches' cross section, as the busses run through them.

We next derive a formula for the silicon area required for a CHIP array. The area of the bus sections along the two sides of each PE is added to the area of the PE thus forming a structure which is a geometric "cell". This cell includes a PE, two bus sections and a corner switching area. Denoting by a the PE size, and by b the bus width, we have for the cell area:

$$S_{cell} = (a + b)^2 = a^2 (1 + \frac{b}{a})^2$$

The increase in area due to communication component is proportional to b/a, where b is determined by the width of the cross section of the corner switch. Therefore, we concentrate on estimating b.

This estimation is based upon the topological design and layout considerations for the principle parts of the lattice: data lines, switching zones and the control section of the switch. The switching area is treated first and a lower bound for the bus width is formulated. Next, the area needed for the control section, which depends on the control strategy, is calculated.

2.1. CHIP 4-1 LATTICE

We follow the terminology suggested by Snyder and use the notation $d-g$ for a switch having d incident busses (d is called the degree of the switch), and cross-over capacity of g links. Thus, a switch that can form only one link between the four busses connected to it is called a 4-1 switch, and the grid containing this type of switches is called CHIP 4-1.

As the crossover capacity g of the 4-1 switch is 1, there are only $\binom{4}{2} = 6$ possible links. The logic design approach is to let each of the busses have a controlled access to an internal "link" through an on-off switch. We have therefore, four groups of pass transistors, controlled by four lines: *left*, *right*, *up* and *down*. Figure 2.1 shows a schematic implementation of this switch. Each bus has m bits implemented as metal lines. At the switching zone, these lines are routed in the diffusion layer, to allow the formation of pass transistors. The internal link is the area in which the busses meet. Contacts are formed between the metal and diffusion lines, on the diagonal of the crossing points, thus minimizing the bus cross section.

Figure 2.2 shows the location of switches within the full lattice. We can distinguish between the corner switches (located on bus intersections), and side switches which control the access of PEs to the communication busses. It can be seen that when the latter are eliminated, all the logic capabilities of the lattice are still met, as long as all the PEs are fault-free.

From these figures we conclude that the bus should be only slightly widened, lines separation is now 3.5λ. For an m line bus, the minimal width is therefore,

$$b_{[4-1]}(m) = (6.5 * m + 3)\ \lambda \qquad (2.1)$$

Next, we include the effect of driver implementation. The active stages need additional area for the pull-up transistors. The size of these depend on the required speed and the parasitic capacitors on the metal lines, and is estimated to be 15λ. The active transistors need also power and ground lines. The output "low" voltage places a limit on the diffusion resistance, i.e., the length of the connection to ground. Therefore, power and ground lines are added between every group of 16 data lines. The overall bus width for m lines including pull-ups, Vcc and GND lines is given by:

$$b_{[4-1]}(m) = (6.5 * m + 18 + 12 * \lceil m/16 \rceil)\ \lambda \quad (2\text{-}2)$$

A typical value for a 16-bit bus is 134 λ.

2.2. CHIP 8-1 LATTICE

The logic structure of this lattice which is used in [1] for embedding hexagonal structures, is based on the 8-1 switch. It consists of 8 pass switches (one for every incident bus), controlling the access to the internal link area.

The schematic layout of the 8-1 switch consists of four bus-overlapping zones, each identical in area to the 4-1 switch. Its cross-section is therefore, approximately double that of a 4-1 switch. Hence, for CHIP 8-1 we have

$$b_{[8-1]}(m) = (13 * m + 18 + 12 * \lceil m/16 \rceil)\ \lambda \quad (2\text{-}3)$$

2.3. CHIP 8-4-1 LATTICE

This lattice is an extension to the original CHIP lattices. It differs from them in making a distinction between the degree of switches and that of processors. The degree of the corner switch has a major effect on the area needed for the communication network. The degree of processors on the other hand determines the ease of mapping various structures onto the CHIP lattice.

In the proposed CHIP 8-4-1 lattice, the corner switches used are the same as in the 4-1 lattice, while the degree of the PE is 8, i.e., each PE has 8 ports, 2 on each side. This way an 8-4-1 lattice has linking capabilities that are equivalent to those of CHIP 8-1. Figure 2.3 shows a possible way of implementing this grid. On the bus along the PE side there are additional pass-transistors which can cut the bus into two separate independent links. As the switches are integrated into the metal lines of the data bus, the area needed for the communication in CHIP 4-1 and in CHIP 8-4-1 are essentially the same. However, the fact that each processor has 8 ports instead of 4 might cause some increase in the area of the processor.

2.4. OTHER SWITCHES AND LATTICES

In the same manner, we may derive expressions for b in other CHIP lattices. Consider for example, CHIP 4-2 in which the switch characteristics are $d = 4$ (incident busses) and $g = 2$ (concurrent links). This switch has nine bus-overlapping zones and the bus width is therefore, three times that of a 4-1 grid. Thus,

$$b_{[4-2]}(m) = (19.5 * m + 18 + 12 * \lceil m/16 \rceil)\ \lambda \quad (2\text{-}4)$$

Similar equations for 8-2 and 8-4 lattices can be derived.

2.5. SWITCH CONTROL

The switches in the CHIP lattice as presented in [1] are controlled by an external host. In it, the mapping algorithms are run and the setting of the switches is determined. A different scheme is distributed control of the switches. Here, the mapping algorithms are executed in the PEs and they determine the setting of the switches.

However, for the evaluation of the area of CHIP lattices only the exact location of the switch control circuitry is important. There are two possibilities for this:

(1) The switch control is internal to the switch. Setting instructions are received directly by the switch which decodes them and activates the the pass transistor control lines. We call this control scheme Direct Switch Control (DSC).

(2) Switches are activated indirectly through PEs. The setting instructions are received by the PEs which decode them to control the adjacent switches. We call this scheme Indirect Switch Control (ISC).

The first was suggested in [1] and the second is a novel idea introduced here.

2.5.1. DIRECT SWITCH CONTROL

The principal parts of the switch control are the communication sub-section and the switch settings memory. DSC switches receive their setting instructions through a dedicated global bus which is assumed to be a serial one in order to minimize the area requirements. Each switch has a shift register and some control logic for implementing the communication protocol. A detailed design was carried out [7] and as a result the area for this logic was estimated at $76 \times 60\ \lambda^2$. The memory for the switch settings has c addresses [1] and its estimated overall size with $c = 2$ and $d = 4$ (four busses), is $60\text{x}24\ \lambda^2$.

Different layout schemes for the switch control have been checked and 76 λ is the minimal addition to the width of the busses (to account for the inclusion of the control section in the switches) that was obtained. For a 16-bit bus and 1000 λ processors, the addition of this to the 134 λ required for the data lines in CHIP 4-1 lattice, increases the bus width by 57%. The total increase in area due to the communication grid is in this case 46.4%.

2.5.2. INDIRECT CONTROL VIA PEs

Including the control logic in the switches requires substantial addition to the bus area. Letting the PEs to control the switches eliminates this area penalty without adding much to the PEs' area which have communication capabilities anyway. Moreover, the data busses can be used for the configuration control, eliminating the additional global bus. There are however, some questions concerning the organization of such a control since instructions have to be transmitted through the lines controlled by them. A mechanism assuring the transfer of control data (switch settings), must be provided. Also, the reconfiguration should be possible even in the event of faulty PEs.

It is insufficient to assign a unique PE to every switch, and pass the switch setting messages through it, as a fault in this PE would mean inability to control the switch in order to bypass the faulty processor. A simple way is to OR the four control lines from the PEs surrounding the switch.

In the configuration phase, the data lines are operated as contiguous vertical (or horizontal) busses. All the PEs listen to the configuration messages transmitted on the nearby bus by the host computer . Switch setting for this phase is initiated directly by the host computer, using a mechanism for cascading the switches along straight lines, (which is realized in the switch's hardware), without the intervention of the PEs. This way, the access to each PE is assured as long as the switches and data busses are fault-free, since alternative paths to each PE exist.

A detailed design of these switches is presented in [7]. As an example, we outline here the principles of ISC implementation in a CHIP 8-4-1 lattice. The allowed forms of links, can be divided into two types: Links between adjacent PEs, and links in which a cascade of switches is formed. The formation of the latter is initiated by the processor at one end of the bus segment while the PE at the other side "cuts" the segment and stops the cascading process. The switches that are not on either edge of this link, are not controlled by PEs, but rather by control lines coming from the preceding switches in the bus segment.

A large area is available beneath the metal lines of the busses and therefore, almost all the amount of random logic needed here can be implemented in the diffusion and polysilicon layers under these lines. Based on a detailed logical design of the switch, we have concluded that the addition to the bus width is that of a single inverter, i.e., 12 λ. For the same example as before, i.e., a 16-bit bus and 1000 λ processors, the width of the bus with ISC switches is increased by less than 10% compared to the 57% for DSC switches. The total increase in cell area is here 31.3% compared to 46.4% if DSC switches are used.

3. MAPPING LOGICAL STRUCTURES ONTO CHIP ARRAYS

In this section we discuss the mapping of various logical structures onto CHIP arrays. These include linear array, loop, square mesh, hexagonal array, and binary tree. The efficiency of the mappings is evaluated by calculating the size of the physical array required for a given size of a logical array. Goals of such mappings may be:

(1) Regularity - uniform mapping strategy for each of the mapped cells.
(2) Locality - short links, preferably to nearest neighbors.
(3) Simplicity - of the algorithms.
(4) Modularity - expansion capabilities to any size of the logical structure.

3.1. MAPPING NON-TREE TOPOLOGIES ON CHIP ARRAYS

Linear arrays, loops and square arrays do not require more than four connections per PE. Hence, they can be mapped on a switchless and CHIP arrays in a straightforward manner. When mapping hexagonal arrays, each PE needs 6 ports. Since PEs in CHIP 4-1 have only 4 ports, the CHIP 8-1 lattice has been suggested for this purpose. In this case, all the PEs and switches are utilized. However, in the 8-1 lattice, the area for communication is almost double than that of 4-1 lattice.

To map the hexagonal array on CHIP 4-1 lattice or mesh arrays without switches we may adopt the idea proposed in [1] to increase the degree of a PE by combining adjacent processors into pairs. This way, the number of ports of a pair of processors is $2*4-2=6$ as needed for a PE in the hexagonal array. The major drawback in both cases is the low PE utilization; half of the processors are operated as communication elements (CEs).

Hexagonal arrays may be mapped more efficiently on a CHIP 8-4-1 lattice by using the corner switches for the diagonal links. This shows a significant advantage of the 8-4-1 lattice for hexagonal array mappings. It retains 100% PE utilization as 8-1 lattices do, while requiring only half the area for communication. Comparing it to the 4-1 lattice, it needs the same area for communication as a 4-1 lattice, which achieves only 50% PE utilization.

3.2. MAPPING TREES ON ARRAYS WITHOUT SWITCHES

Algorithms for mapping trees on square and hexagonal switchless grids appear in [4] and [8], respectively. The mapping is systematic, modular and simple but it uses many PEs for communications purposes. PE utilization varies with tree level and approaches 50% in square grids. The mapping on an hexagonal grid is more efficient, but requires a more complex algorithm. Improvements in PE utilization and in the mean length of a path at a factor of 1.4 as compared to the square array, can be achieved [8]. The mean length of a path also decreases at a similar factor.

3.3. MAPPING TREES ON CHIP ARRAYS

In [1] a mapping scheme for binary trees that achieves 100% PE utilization, is suggested. The basic idea is to use two kinds of building blocks, one with its "spare" PE located at its corner, and the other at the center of its side. The root in each block is located in its center. The process of building higher level trees from these blocks is by assigning the new root to one of the spare PEs. Using transformations on the basic building blocks, we can position the new "spare" either at the center of the new side or at the new corner, enabling the process to proceed.

This recursive algorithm will function provided that some basic cell, satisfying our assumption on root and spare positioning is found, and that we can assure the creation of links between the new root and the previous ones. Closely checking the example given in [1], (mapping an eight level tree on a 16x16 CHIP 4-1 lattice), we find that the basic cell consists of 16 PEs which host a level 4 sub-tree. However, 12 different configurations of the basic cell are needed. The reason for this is the inability to locate the spare exactly at the center of the cell side (as required by the algorithm). The 12 configurations can be grouped into 3 types denoted here A, B, and C types. Figure 3.1 shows the internal structure of the basic cell C after it was slightly modified to introduce fault-tolerance capabilities (Section 5). These types of cells appear in all possible mirror transformations, and we denote them according to the axis of symmetry: A_x, A_y, A_{xy}, and similarly for B and C cells.

The schematic diagram in Figure 3.2 shows how a ten level tree can be formed using the same approach as the 8 level one in Snyder's example. The basic rules governing this mapping are:

(1) A-type cells host tree nodes of level 5 .
(2) B-type cells host tree nodes of level 6 .
(3) C-type cells host all the higher level nodes.

Due to limitations of the CHIP 4-1 lattice, not all the details in Figure 3.2 were derived from the above mentioned mapping algorithm. For example, the position of the level 9 node has moved a level 5 node from its "natural" place [7]. In general, there is a difficulty in systematically extending this mapping algorithm to higher level trees. When forming the links to the root of a new (higher level) tree, replacement of several cells in the already existing sub-trees is required.

The modified scheme in the next paragraph, achieves a systematic mapping at the expense of PE utilization.

3.4. SYSTEMATIC TREE MAPPING ON CHIP 8-4-1 ARRAYS

The next algorithm is conceptually based on the H-spanning scheme (e.g., [6]) with some necessary modifications for CHIP architecture. The idea is to leave the four corners of some basic structure as spares to be used for mapping higher level nodes. This will ensure that in all possible transformations, the corner required by the mapping algorithm is available.

Choosing the size of the basic structure affects the efficiency of the mapping. The one presented here (Fig 3.3) consists of four basic cells of size 3x3. Each cell has nine PEs, out of which seven are utilized for a sub-tree of level 3, one is used as a node at level 4 or higher, and one as a "spare". Thus, a PE utilization of 8/9 is achieved. A structure of 6x6 PEs, formed by four of these basic cell, has spares available at its four corners. Our algorithm maps tree nodes of level 7 and higher onto these spares. Thus, 32 out of every 36 PEs are utilized. Figure 3.3 is the output of a PASCAL program which implements this algorithm for any tree level.

The following equations give the number of rows and columns required for mapping a k level tree :

$$n_{row} = 3 * 2^{\lfloor (k-4)/2 \rfloor} \tag{3.1}$$

$$n_{col.} = 3 * 2^{\lfloor (k-3)/2 \rfloor} \tag{3.2}$$

For example, for k=8 and k=9, arrays of 24x12 and 24x24 are needed, respectively. The maximum tree level that can be mapped on a $n_{row} * n_{col.}$ array is given by

$$k = \begin{cases} \min(a_{row}, a_{col.}) + 1 & \text{if } |a_{row} - a_{col.}| > 1 \\ (a_{row} + a_{col.})/2 & \text{otherwise} \end{cases} \tag{3.3}$$

$$a_{row} = \lfloor 2 * \log_2(n_{row} / 3) \rfloor + 3 \tag{3.4}$$

$$a_{col.} = \lfloor 2 * \log_2(n_{col.} / 3) \rfloor + 3 \tag{3.5}$$

The algorithm calculates at each step (level) the length of the next link denoted by δ which is halved at even numbered steps. Its initial value is calculated from,

$$\delta(k) = 3 * 2^{\lfloor (k-6)/2 \rfloor} \tag{3.6}$$

4. EFFICIENCY OF TREE EMBEDDINGS

In this section we use the results of the previous paragraphs to evaluate the embedding efficiency when tree structures are embedded in CHIP lattices, compared to that in arrays without switches.

The common approach quoting the number of PEs required for a logical structure does not consider the extra area needed for the busses and switches in CHIP lattices, and is therefore not adequate. We use instead the maximal size of the logical structure that can be mapped onto a given physical chip area. This criterion is affected by the PE's size; the bigger the PE, the more worthwhile it is to invest in the communication sub-system to get better PE utilization. The size of the PE is dictated by the computational capacity needed, and therefore is left as a parameter in the evaluation. The other parameters are those of the communication sub-system: the width of data busses, and the switch structure.

In the comparison we assume that the 8 ports per PE in the 8-1 and 8-4-1 lattices (compared to 4 in the 4-1 lattice) do not add a significant amount af area to the PE. We also ignore in the comparison the common elements which appear in any array such as power and clock lines.

The number of tree levels that can be embedded in a switchless square array is given in [4]. Using those formulas, the chip area needed for a given tree has been calculated, with the PE size as a parameter. For example, the area required for an 8 level tree is 529 M λ^2 for a=1000 λ.

Based on the results of the preceding paragraphs, the size of the CHIP array required for trees of any level can be calculated. It is a function of the specific algorithm, and the two mapping schemes which are considered here are Snyder's suggestion for CHIP 4-1 arrays with 4x4 cells and full PE utilization, and the algorithm presented above for CHIP 8-4-1 arrays with 3x3 cells and 8/9 utilization.

Figure 4.1 gives a "rule-of-preference" in the plane of (a, m), i.e., the combination of PE size and bus width. The lower line shows the break-even in embedding efficiency between switchless mesh and CHIP 4-1 arrays, and the upper one shows the border line between switchless mesh and CHIP 8-4-1. For 1000 λ PEs and 16-bit busses, CHIP lattices offer a higher efficiency of tree embeddings while for smaller PEs switchless arrays are preferable.

5. FAULT-TOLERANCE IN PROCESSOR ARRAYS

5.1. MAPPING OF LINEAR ARRAYS

An algorithm for mapping a linear structure onto a square array of $n * n$ processors without switches is presented in [4]. For a single faulty PE, an average number of $(n-1)$ processors are turned into CEs.

As can be seen from Figure 5.1, the existence of switched busses in CHIP lattices enables routing around any faulty PE, thus achieving full utilization of all the fault-free PEs. It can be shown that this is true for any combination of faulty PEs in the array.

Faults in communication links, need a more careful attention. Single or double link faults in CHIP 4-1, even when all occur in links adjacent to a single PE, can be bypassed without the need for CEs, while a triple fault may block the access to a good PE. In CHIP 8-4-1 there are two possible links between any two adjacent PEs, providing redundancy for any non-operative link.

5.2. MAPPING OF SQUARE AND HEXAGONAL ARRAYS

In [4] a straightforward algorithm is presented for the mapping of rectangular structures on an array of processors without switches. The use of a faulty PE is avoided by turning a row *and* a column of processors into CEs. In the case of faults in the communication links, turning a row *or* a column of PEs into CEs is sufficient.

Following the same approach, it is possible to overcome PE faults in CHIP 4-1 array at half the "price", i.e., not utilizing a single row *or* a column of operational processors in the event of a PE fault. Switch and link faults can be handled in the same way, resulting in a row *or* a column of CEs. Only in the case of faults in all the four switches surrounding a PE, both the row and the column of PEs are turned into CEs.

Figure 5.2 shows that the additional flexibility of CHIP 8-4-1 grids can be used for the recovery from a faulty PE at the "price" of only one side of the appropriate row or column of PEs. Thus, the average number of unutilized fault-free PEs per a single fault is half a row *or* column. There are however, some combinations of faults for which this strategy is not applicable. For example, successive faults along a diagonal cannot be treated in this manner.

The same approach applies for the reconfiguration in the event of faults in hexagonal arrays.

5.3. EMBEDDINGS OF BINARY TREES

Two different approaches for embedding binary trees in CHIP arrays have been presented. Snyder's suggestion which achieves full PEs utilization in CHIP 4-1 array, and the algorithm presented in this paper which achieves 8/9 utilization of the PEs in a CHIP 8-4-1 lattice. In both cases, fault-tolerance can only be achieved by having some redundant components in the array.

There are two possible ways to distribute the spare PEs in the array:

(1) The global approach - spare rows and columns are added to the initial array. An appropriate fault-tolerance strategy was presented above for square arrays, and can also be applied to mapping binary trees in the presence of faults.

(2) The local approach - spare PEs are spread over the entire array, one per each basic cell, offering local redundancy, and enabling fault-tolerance by reconfiguring only the cell in which a fault in a processor occurs. Other parts of the array remain unchanged. A similar idea was presented in [9] for wafer-scale integration of CHIP lattices.

5.3.1. FAULT-TOLERANCE IN TREE MAPPING ON CHIP 4-1

As was shown earlier, the mapping of binary trees onto CHIP 4-1 as suggested in [1] is based on three types of basic cells. The following algorithm is suggested to overcome any fault in the PEs;

(1) All the PEs in the same *row* of the faulty processor, are not utilized.

(2) All the switches between processors in this row, are set to form a vertical link.

(3) The tree is re-mapped on the modified array, according to the original algorithm, with the number of rows reduced by one.

Here the PEs cannot be turned into CEs, as it is certain that at least one of them is faulty. Note that in step (1) we give up the use of PEs in a *row* and not in a column, since there are more active horizontal links than vertical ones.

It is sufficient to check the reconfiguration of a single transformation of every type of the basic cells, for all the possible locations of a fault within that cell. Some examples are illustrated in Figure 5.3. The number of switches available for bypassing faulty PEs (step (2) above) within each cell is three. This is insufficient for PE faults in the original structure of type C cell (as suggested by Snyder), and we present here therefore, a slightly modified structure as shown in Figure 3.1 (c). In the case of faulty links, for various locations of these faults either a row or a column of PEs is not utilized, and the use of some of the PEs as CEs is also required in some cases [7].

5.3.2. FAULT-TOLERANCE IN TREE MAPPING ON CHIP 8-4-1

Here, processor faults within a basic cell are handled differently from those in tree nodes of level six or higher. As can be seen in Figure 3.3, there are spare processors adjacent to every node of level 6 or higher. Therefore, faults occurring in these nodes are overcome by replacing the faulty PE by a nearby operational processor. A single PE fault within the basic cell can be handled by reconfiguring the cell without affecting the routing anywhere else in the array. Figure 5.4 depicts an example of reconfigurations of the basic cells. Although a general procedure cannot be stated for these structures, the use of a table seems practical due to the limited number of possibilities.

5.4. QUANTITATIVE EVALUATION OF FAULT-TOLERANCE

The following is based on the work in [5]. The behavior of the processor array is represented by a Markov model, in which at state (i,j) the array is operational in the presence of i faulty PEs, and j faulty communication links. F is the system failure state, i.e., any combination of faults that prohibits successful recovery.

We denote by $P_{i,j}(t)$ the probability that the array will be at state (i,j) at time t. Also, $\alpha_{i,j}$, $\alpha_{i,j}^{i+1,j}$, and $\alpha_{i,j}^{i,j+1}$ denote the transition rates from state (i,j) to state F, state $(i+1,j)$ and state $(i,j+1)$, respectively.

The general solution of this Markov model is given in [5]. Based on this solution we may calculate several performance measures like, *Reliability* $R(t)$ (probability that the system operates correctly in [0,t]), normalized *Computational Availability* $A_c(t)$ (the number of processing elements that are operational at time t, divided by the initial number of processors in the array), and *Area Utilization* $U(t)$ ($A_c(t)$ / Total area of the chip).

5.4.1. SQUARE ARRAYS EMBEDDED IN CHIP LATTICES

Let $n * n$ be the size of the square array to be mapped onto a CHIP 4-1 lattice. The following equations were derived for the transition rates for the corresponding Markov model:

$$\alpha_{i,j}^{i+1,j} = (n - \frac{i+j}{2})^2 * \lambda_o \quad (5\text{-}1)$$

$$\alpha_{i,j}^{i,j+1} = (2n*(n-1) - j*(n-1) + i)*\lambda_c + j*n*\lambda_o \quad (5\text{-}2)$$

and,

$$c_{i,j} = (1 - \frac{i+j}{2n})^2 \quad (5\text{-}3)$$

where λ_o and λ_c are the failure rates of a PE and a link, respectively and $c_{i,j}$ is the number of operational processors in state (i,j) [5].

The first equation reflects the symmetry in rows and columns when PE faults are considered. Each fault causes a loss of a row or a column of PEs. In the second one, the number of operational links is calculated by subtracting $n-1$ links from the initial number of links (which is $2n * (n-1)$) for each of the j faulty links, and adding i links which bypass faulty PEs.

In order to compare the results with those obtained for the switchless array, the same example from [5] was used. In this example, the initial array is of size 10x10 and faults are tolerated as long as the size of the array after reconfiguration is at least 7x7.

A CHIP 4-1 array requires, in comparison to the switchless array, an additional area. The area increase is for example, 31.3% for PE size of 1000 λ and 16-bit wide communication links. Consequently, in the area of a 10x10 switchless array, a smaller CHIP 4-1 array of size 9x8 approximately can be implemented. For simplicity however, the calculations were carried out on the symmetrical array of 9x9 PEs. In this case, the maximal number of faults that can be tolerated is 4, either PE faults or link faults.

An alternative approach was also examined. According to it the initial CHIP 4-1 array is 10x10 and the total area is enlarged by 31.6%, thus affecting the initial value of the area utilization. In this case, six faults of either type can be tolerated, resulting in a higher reliability.

For CHIP 8-4-1 lattice we derive the following:

$$\alpha_{i,j}^{i+1,j} = (n - (\frac{i}{2} + j) / 2)^2 * \lambda_o \qquad (5\text{-}4)$$

$$\alpha_{i,j}^{i,j+1} = i * \lambda_c \qquad (5\text{-}5)$$

$$c_{i,j} = (1 - (\frac{i}{2} + j) / 2n)^2 \qquad (5\text{-}6)$$

The area-enlargement factor for this grid, under the same assumptions is 41.6%. Keeping the same physical area as the 10x10 switchless array, allows the formation of a 8x8 array in which any mix of 4 faults can be tolerated. If the initial array is of size 10x10, we achieve a higher reliability at the price of larger chips (41.6%) since any mix of 12 faults may be tolerated now.

Figure 5.6 summarizes the numerical results obtained. From the graphs, the trade-offs between the performance measures become clear. Higher reliability is obtained when a larger part of the array is devoted to redundancy, which causes lower utilization. An important observation is that even though the rate at which the area utilization decreases over time is considerably lower for CHIP lattices, they do not reach (within the time interval of practical importance) the utilization offered by the switchless array, due to the initial difference. Thus, we conclude that under the assumptions made in this example, switchless arrays are preferable.

5.4.2. BINARY TREES EMBEDDED IN CHIP LATTICES

The example we consider here is the mapping of a seven level tree. In the switchless grid, a 15x15 array is required, and if two faults are to be tolerated, the initial array has to be 17x17. This is a 28% increase in the area requirement.

The minimal physical array needed for embedding the same tree in CHIP 4-1 lattice is 8x16. To tolerate two faulty PEs, we need to add two rows, i.e., an initial 10x16 array which has a smaller physical size compared to the 17x17 switchless array, yet yields comparable reliability. Using the area enlargement factor of 1.316, an array containing 19x11 PEs is still physically smaller than the switchless array, yet offers the possibility of tolerating three faults, and hence higher reliability.

The results given in Figure 5.7 are for the case when $\lambda_c = 0$, i.e., communications links are assumed to be fault-free. This simplifies the calculations considerably, and is believed to suite the purpose of the comparison being done here. In this case,

$$\alpha_i^{i+1} = n_{col.} * (n_{row} - 1) * \lambda_o \qquad (5\text{-}7)$$

Using basic cells of 3x3 PEs, the physical array needed for the seven level tree contains 12x12 processors. According to the fault-tolerance strategy for this lattice, as presented earlier, a seven level binary tree can be embedded in this array, in the presence of up to sixteen PE faults, as long as a double fault does not occur in any of the basic cells. For this case we have, when $\lambda_c = 0$,

$$\alpha_i^{i+1} = (n_{row} * n_{col.} - 9i) * \lambda_o \qquad (5\text{-}8)$$

$$\alpha_i^{F} = 8i * \lambda_o \qquad (5\text{-}9)$$

The area required for this array is approximately 30% less than that of the 17x17 switchless array, and yet it achieves better reliability as can be seen from Figure 5.7. Thus we conclude, that under the assumptions made in this example, the local approach for distributing redundancy when employed on a highly flexible reconfigurable lattice, such as the CHIP 8-4-1 lattice, yields better performance than that of the global approach.

6. SUMMARY

In this paper we have estimated the area requirements of different CHIP lattices. Area was estimated based on a detailed design of the switching element with processor size and communication bus width as parameters. For a typical case, when the PE size is 1000x1000 λ^2 and 16-bit busses, the increase in area is estimated to be 31.3%.

For the purpose of evaluating the cost-effectiveness of switches in the architecture, a comparison was made between CHIP lattices and a switchless array of processors. The size of the physical array required to embed a given binary tree in the various processor arrays served as basis for this comparison.

The fault-tolerance capabilities of these arrays were analyzed both qualitatively and quantitatively. It was shown that CHIP lattices offer lower area utilization when mapping linear and square structures, but achieve higher reliability and area utilization when mapping binary trees.

Although only CHIP architectures have been considered in this paper, it is believed that the method of analysis presented here is applicable to other similar architectures.

7. REFERENCES

[1] L. Snyder, "Introduction to the Configurable Highly Parallel Computer," *Computer*, Vol. 15, January 1982, pp. 47-56.

[2] A. L. Rosenberg, "The Diogenes Approach to Testable Fault-Tolerant Arrays of Processors," *IEEE Trans. on Computers*, Vol. C-32, Oct. 1983, pp. 902-910.

[3] W. H. Lee and M. Malek, "MOPAC- A Partitionable and reconfigurable Multiprocessor Array", *Proc. of the 1983 Interml. Conf. on Parallel Processing*, August 1983, pp. 506-509.

[4] I. Koren, "A Reconfigurable and Fault-tolerant VLSI Multiprocessor Array," *Proc. of the 8th Annual Symposium on Computer Architecture*, May 1981, pp. 425-441.

[5] I. Koren and M.A. Breuer, "On Area and Yield Considerations for Fault- Tolerant VLSI Processor Arrays," *IEEE Trans. on Computers*, Vol. C-33, Jan. 1984, pp. 21-27.

[6] C. Mead and L. Conway, "*Introduction to VLSI Systems*," Addison-Wesley, 1980.

[7] H. E. Mizrahi, "Evaluation of the Cost-effectiveness of Switches in Architectures of Processor Arrays," *M.Sc. Thesis*, (in Hebrew), Dept. of Electrical Engineering, Technion, Haifa, Israel, October 1984.

[8] D. Gordon, I. Koren, and G. M. Silberman, "Embedding Tree Structures in VLSI Hexagonal Arrays," *IEEE Trans. on Computers*, Vol. C-33, Jan. 1984, pp.104-107.

[9] K. Hedlund and L. Snyder, "Wafer Scale Integration of Configurable, Highly Parallel (Chip) Processor," *Proc. of the 1982 Interml. Conf. on Parallel Processing*, August 1982, pp. 262-265.

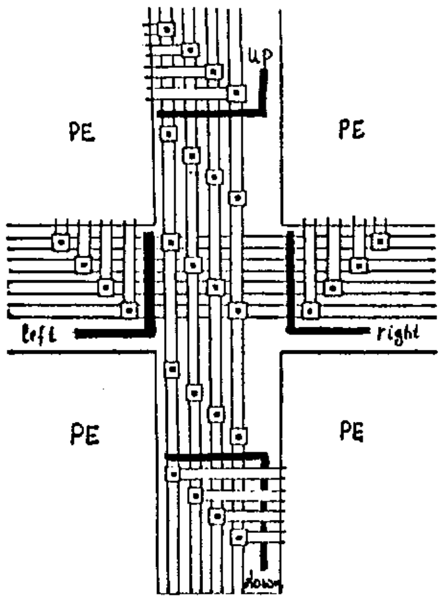

Fig. 2.1: Busses and pass transistors within a 4-1 switch.

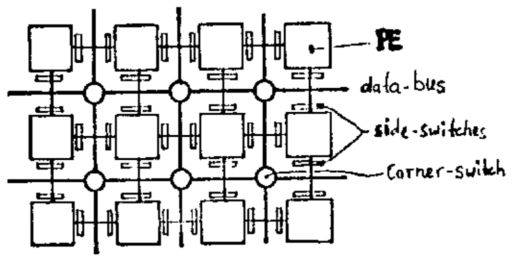

Fig. 2.2: A CHIP 4-1 lattice.

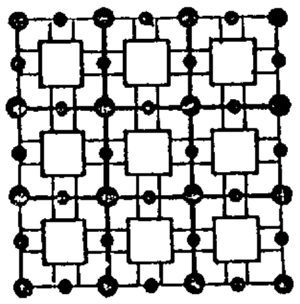

Fig. 2.3: A CHIP 8-4-1 lattice.

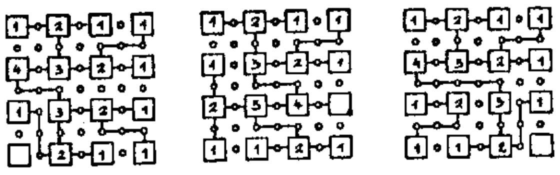

Fig. 3.1: Basic cells of type A,B and C (modified).

A	B	By	Ay	A	B	By	Ay
Ax	Cx	Cy	Axy	Ax	C	Cxy	Axy
A	C	A	Cx	A	C	By	Ay
Ax	Bx	Bxy	Axy	Ax	Bx	Ax	C
A	B	By	Ay	Cxy	Ay	By	Ay
Ax	Cx	Ax	C	Ax	Bx	Cxy	Axy
A	C	Cxy	Ay	A	Cx	Cy	Ay
Ax	Bx	Bxy	Axy	Ax	Bx	Bxy	Axy

Fig. 3.2: Embedding a 10 level binary tree in a CHIP 4-1 array.

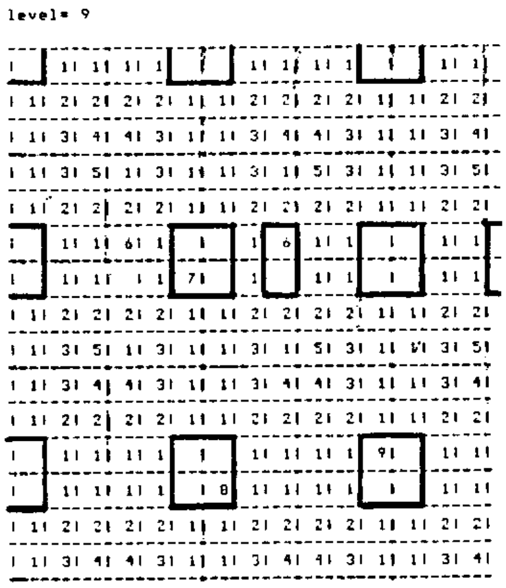

Fig. 3.3: Systematic embedding of a binary tree in CHIP 8-4-1.

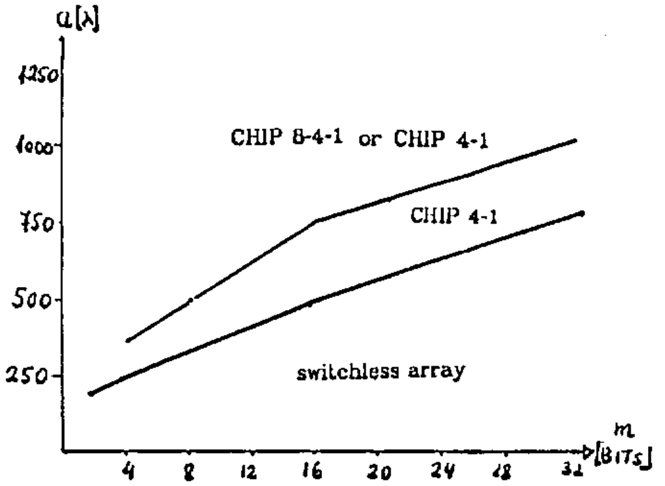

Fig. 4.1: The preferred lattice for tree embeddings, shown as a function of processor size and bus width.

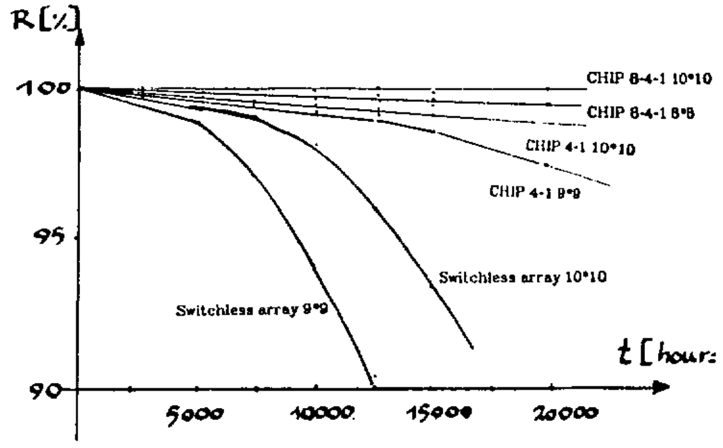

Fig. 5.4: Fault-tolerance in the basic cells, when embedding trees in a CHIP 8-4-1 lattice.

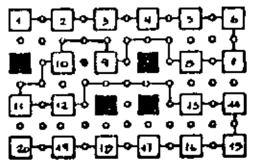

Fig. 5.1: A linear array mapped onto CHIP 4-1 lattice in the presence of faults.

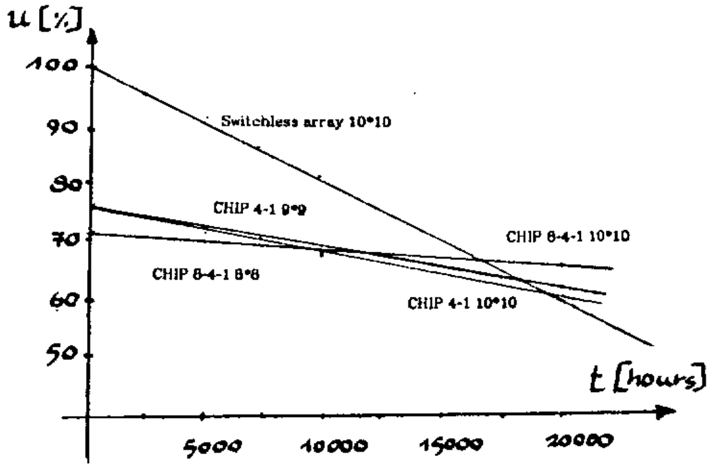

Fig. 5.5: Performance measures when mapping square arrays onto different lattices.

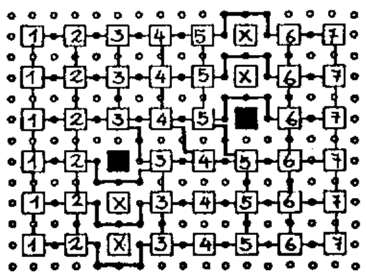

Fig. 5.2: Mapping a square array onto a CHIP 8-4-1 lattice in the presence of faults.

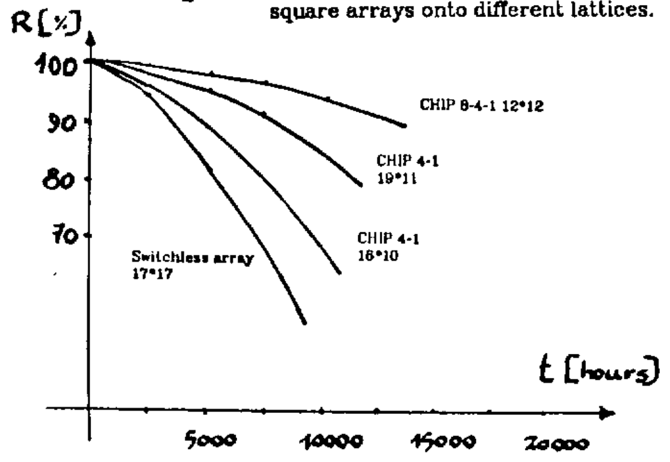

Fig. 5.6: The reliability when mapping binary trees onto different lattices.

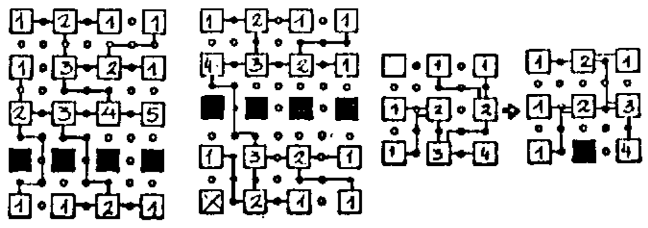

Fig. 5.3: Fault-tolerance in the basic cells, when embedding trees in a CHIP lattice.