[16] P. Montuschi and L. Ciminiera, "Fast radix-2 division and square root," Internal Rep., I. R. DAI/ARC 15/90, Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy.
[17] G. W. Reitwiesner, "Binary arithmetic," in *Advances in Computers*, F. L. Alt, Ed. New York: Academic, 1960.
[18] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 218–222, Sept. 1958.
[19] P. K.-G. Tu "On-line arithmetic algorithms for efficient implementation," Ph.D. dissertation, Comput. Sci. Dep., U.C.L.A., CSD-900029, Sept. 1990.
[20] T. E. Williams and M. A. Horovitz, "A 160nS 54bit CMOS division implementation using self-timing and symmetrically overlapped SRT stages," in *Proc. 10th IEEE Symp. Comput. Arithmetic*, Grenoble, France, June 1991, pp. 210–217
[21] J. B. Wilson and R. S. Ledley, "An algorithm for rapid binary division," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 662–670, Dec. 1961.
[22] S. Winograd, "On the time required for binary addition," *J. ACM*, vol. 12, pp. 277–285, 1965.
[23] J. H. P. Zurawski and J. B. Gosling, "Design of high-speed digital divider units," *IEEE Trans. Comput.*, vol. C-30, pp. 691–699, Sept. 1981.

A Cube-Connected Cycles Architecture with High Reliability and Improved Performance

Nian-Feng Tzeng

Abstract— **The cube-connected cycles (CCC) architecture is an attractive parallel computation network, because it is suitable for VLSI implementation while preserving all the desired features of hypercubes. However, the CCC tends to suffer from considerable performance degradation when a fault arises. In this paper, a fault-tolerant CCC which exhibits significantly enhanced reliability is proposed. Reconfiguration in response to an operational fault in this fault-tolerant CCC is simple and can be performed in a distributed manner. When compared with the CCC, the proposed design in the absence of faults gets performance improvement as a result of faster broadcasting and PE-to-PE communication. The layout of this structure is discussed, and its area overhead is found to be moderate if the PE size is much larger than the link/switch size. Therefore, this design approach is particularly useful for situations where the PE is relatively complex.**

Index Terms—**Cube-connected cycles, fault-tolerance, message broadcasting and routing, reliability analysis, VLSI layout.**

I. Introduction

While the natural decomposition of many numerical algorithms creates subtasks whose communication patterns match the hypercube topology [1], the hypercube topology has one major disadvantage: the number of links to each PE grows with the hypercube dimension, which renders the hypercube ill-suited for VLSI/WSI implementation. Preparata and Vuillemin [1] considered a substitute of the hypercube, known as the cube-connected cycles (CCC) architecture, which not only preserves all the attractive features of the hypercube but also has a more compact and regular layout, making it ideal for VLSI/WSI implementation. By combining the principles of pipelining and parallelism, the CCC can effectively emulate the hypercube machine.

As the system size increases, the probability of getting failed components becomes higher, and hence a fault-tolerant scheme has to be incorporated in the system design to guard against failures. In the presence of faults, a fault-tolerant system reconfigures itself to exclude the faulty elements from the system so that proper operation can resume. Normally, it is common for a system upon reconfiguration to encompass all the healthy PE's whenever possible. A system so reconfigured may or may not change its topology. If the available system changes its topology after reconfiguration, it may no longer be able to support the original task partition efficiently. Therefore, a fault-tolerant design capable of preserving its original topology in the presence of faults may be preferable for many architectures, especially for those making use of pipelining and parallelism heavily, such as the CCC. This type of designs is referred to as *strongly fault-tolerant* designs. A strongly fault-tolerant design not only keeps an identical service level but also retains the same system topology even after faults arise. Redundant PE's and links are incorporated in the design to achieve strong fault-tolerance. They are employed to replace faulty ones upon reconfiguration so as to maintain the original form and service level of the system.

This paper concerns a strongly fault-tolerant CCC design, called the XCCC. The XCCC exhibits significantly enhanced reliability and its reconfiguration procedure is simple, involving only a few PE's. Each cycle can tolerate one PE failure, independent of PE faults in other cycles. In addition, the XCCC has the unique feature that under the fault-free situation, it enjoys faster communication, which would translate to improved performance, in comparison with its CCC counterpart. The XCCC layout is pursued and its area overhead is found to be moderate under situations where the PE is much wider than the link/switch. This design approach is thus applicable to the CCC in which a PE is relatively complicated, taking far more area than a switch.

CCC Topology: The CCC interconnects identical processors each with three ports [1]. Each connecting link between two PE's can be used for bidirectional data transmission. A CCC consisting of 2^k cycles with each cycle having h PE's is denoted by CCC(h, k), where h is no less than k. For simplicity, the total number of PE's in CCC(h, k), $N = h \times 2^k$, is assumed to be a power of 2. Each PE has an address expressed as a pair of integers, (c, p), where c and p denote, respectively, the address of the cycle containing the PE and the position of the PE within the cycle, as illustrated in Fig. 1. Cycles are numbered from 0 to $2^k - 1$ starting with the leftmost one, whereas PE's in a cycle are numbered from 0 to $h - 1$ starting with the lowest PE.

Suppose that the three ports of each PE are called F, B, and L (mnemonic for *F*orward, *B*ackward, *L*ateral), respectively. The CCC interconnection is specified as follows [1]: F of PE(c, p) is connected to B of PE$(c, (p+1) \bmod h)$; B of PE(c, p) is connected to F of PE$(c, (p-1) \bmod h)$; L of PE(c, p) is connected to L of PE$(c + \alpha 2^p, p)$, where $\alpha = 1 - 2 \times$ (the pth bit of c). All PE's inside a cycle are circularly connected by the $F - B$ links. The lower k PE's in each cycle are interconnected following the hypercube connection pattern to other PE's in different cycles through the L links, i.e., the ith $(0 \le i < k)$ PE is connected to the corresponding PE in another cycle which is 2^i away from the current cycle (and this L link forms the ith dimensional connection). The upper $(h - k)$

Manuscript received May 16, 1991; revised May 27, 1992.
This work was supported by the NSF under Grant MIP-8807761.
The author is with the Center of Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504.
IEEE Log Number 92002837.

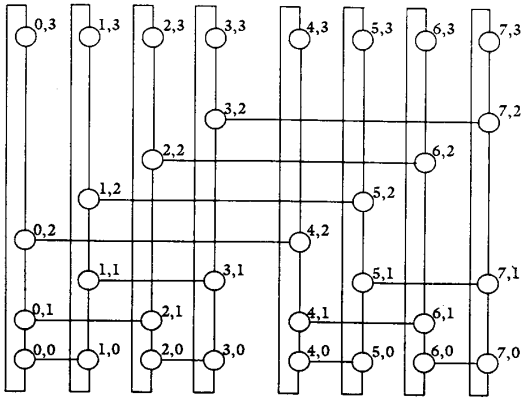

Fig. 1. A layout for CCC (4, 3).

PE's do not utilize their L links. A standard layout for CCC (4, 3) is depicted in Fig. 1.

II. XCCC Design

A. Architecture Description

The CCC structure [1] makes use of the pipelining concept to achieve fast algorithmic executions. A stream of data participates in different stages of pipelining. If one PE fails at a certain cycle, the pipeline can no longer sustain and the system tends to get considerable adverse impacts. When PE(1,1) in Fig. 1 fails, for example, the set of data in cycle 1 loses the ability of direct access to cycle 3 through the dimension 1 connection, as does the set of data in cycle 3 lose direct access to cycle 1. In order to overcome this problem, a pair of extra connections for each PE are provided, as shown in Fig. 2. When PE(1, 1) fails in this structure, PE(1, 2) becomes the vertical successor to PE(1, 0) and the dimension 1 connection now exists between PE(1, 2) and PE(3, 1), as depicted in Fig. 3. It is due to the existence of "cross connections" among cycles that this structure is called the XCCC.

Failures at the F and B links can be tolerated by utilizing added "cross connections" plus control switches; namely, a control switch is placed at the intersection of each pair of "cross connections" [as depicted in Fig. 4(a)] so that a substitute link for any F or B link is formed when the switch is set to the "$\supset$" state [see Fig. 4(b)]. A pair of "cross connections" together with their control switch are referred to as one *switch connection pair* (SCP). After the link between PE(4, 1) and PE(4, 2) fails in Fig. 2, for example, an alternative connection between them is established as shown in Fig. 3. Similarly, a failed L link can be tolerated by replacing it with an alternative connection formed by the SCP existing between the two cycles connected by the failed L link, with the control switch set to the "V" state [see Fig. 4(b)]. When the L link between cycles 2 and 3 fails in Fig. 2, for example, an alternative direct connection is created, as depicted in Fig. 3.

A failed PE is assumed to be bypassed, and data can pass successfully between its B and F links. This can be accomplished by adding four switches to a PE to make data get around the PE, if those switches are set appropriately, as demonstrated in Fig. 4(c). Normally, switches 1, 2, and 3 are on, while switch 4 stays off. If the PE is to be isolated, then switches 1, 2, and 3 are set off, whereas switch 4 is set on to make a direct connection between its B and F links. The four switches cannot fail at the same time as their associated PE. With this provision, cycle 1 ignores the failed PE(1, 1) in Fig. 2,

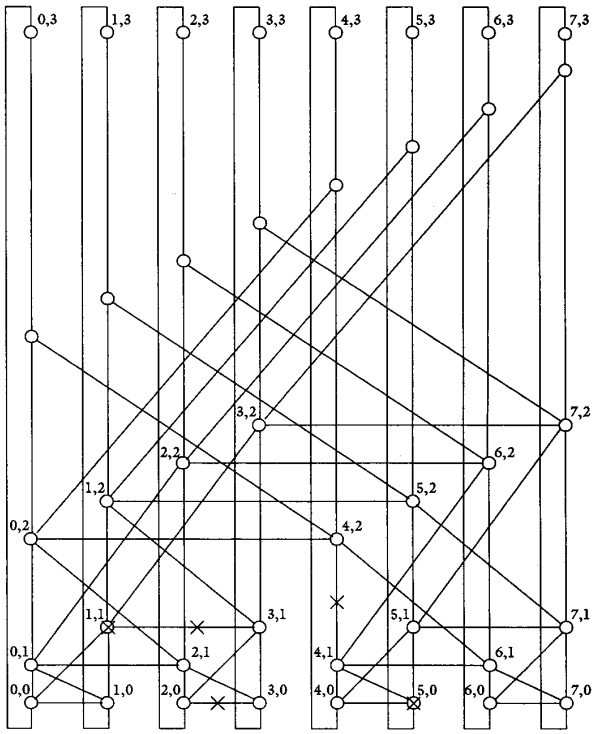

Fig. 2. The structure of XCCC(5, 3) formed by adding two extra connections to each PE and adding an extra row of PE's to CCC(4, 3).

and the cycle is now made of PE(1, 0), followed by PE(1, 2) and its upward successors.

An additional PE is necessary for each cycle to assure that the same sized cycle is available even when there is a fault in the cycle. The extra PE is placed at the top of dimensionally connected PE's in a cycle, namely, immediately above PE($c, k-1$) in every cycle c. The added PE in a cycle acts as a standby spare and is not used under normal fault-free circumstances; it will be enabled only when one PE fails in that cycle.

Every PE in the XCCC has 5 ports, with 3 of them for making the CCC interconnection style and the remaining two for augmenting the structure to achieve fault tolerance as well as performance improvement. The XCCC has $(h+1)$ PE's per cycle and is denoted by XCCC($h+1, k$). Formally, XCCC($h+1, k$) can be defined as a collection of $(h+1)2^k$ PE's, which form 2^k cycles each with $(h+1)$ PE's. In addition to the F, B, and L ports, each PE has the U and D (mnemonic for Up and Down) ports. The interconnection of F, B, and L among PE's is the same as that given in Section I, and the interconnection of U and D is characterized as follows: U of PE(c, p) is connected to D of PE($c+\alpha 2^p, p+1$), for $0 \le p < k$; D of PE(c, p) is connected to U of PE($c+\alpha 2^{p-1}, p-1$), for $0 < p \le k$, where α equals $1-2 \times$ (the pth bit of c) defined before, and PE(c, k) refers to the spare in cycle c when $k = h$. XCCC($h+1, k$) involves 2^k spare PE's and $k \times 2^{k-1}$ added SCP's. The complete configuration of XCCC(5, 3) is illustrated in Fig. 2. Note that the XCCC so drawn is easy to understand and convenient for subsequent explanation. A more compact layout of the XCCC will be presented later.

B. Tolerance of Failures

The capability of tolerating PE failures is addressed first, followed by a discussion of the link fault-tolerance ability. When a spare (PE)

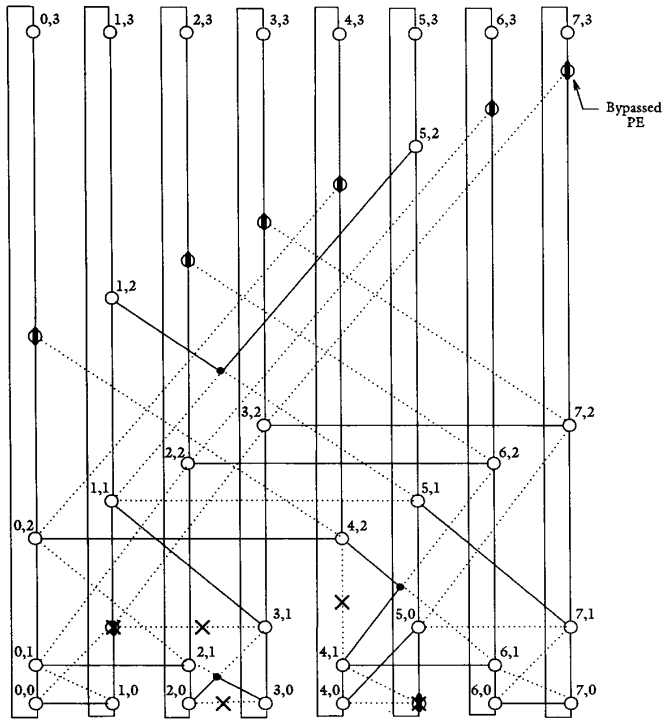

Fig. 3. A reconfigured XCCC(5, 3) in the presence of multiple faults as marked. (Solid lines indicate active links after reconfiguration, and a bypassed PE has a bold line across it. The numbers show the PE logic addresses, rather than physical ones.)

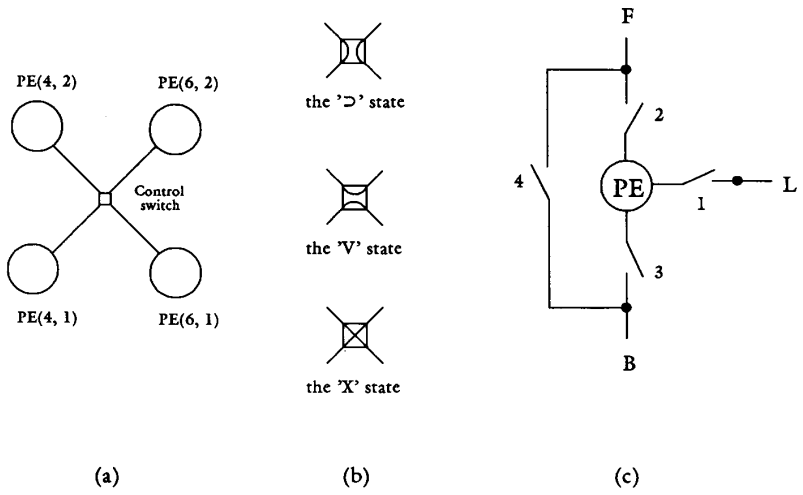

Fig. 4. (a) A cross connection pair. (b) The possible settings of a control switch. (c) A PE with smile switches to facilitate reconfiguration.

fails, it is bypassed and no reconfiguration takes place since the spare in each cycle normally is inactive. If a fault occurs at an active PE, it is bypassed and its role is replaced by its physical successor (in the same cycle), which in turn is replaced by its successor, and so on, until the space is brought in. When a PE replaces the role of its predecessor, it also activates its D to create a direct connection to the PE which is dimensionally connected with the failed PE, with the switch on the link set to an "X" state [see Fig. 4(b)]. After PE(1, 1) failed in Fig. 2, for example, PE(1, 2) replaces the role of the failed PE and also activates its D so as to create a direct connection to PE(3, 1).

Each cycle can tolerate one and only one PE failure, independent of the PE failures in other cycles. As an instance, the failure of both PE(1, 1) and PE(5, 0) in Fig. 2 is tolerable, despite the fact that cycles 1 and 5 are dimensionally connected through PE(1, 2) and PE(5, 2). Due to these two failed PE's, the spares in cycles 1 and 5

serve, respectively, as the roles of PE(1, 2) and PE(5, 2), with a direct connection between them formed by setting the SCP switch to the "V" state, as shown in Fig. 3.

The XCCC can tolerate multiple failed links. A failed link is tolerable, provided that the needed SCP is available to support an alternative connection. More precisely, if the SCP between cycle c and cycle $c' = c + \alpha 2^p$ ($\alpha = 1 - 2 \times$ (the pth bit of c)) is not used yet, then, the failure of one of the following three links can be tolerated: 1) link between PE(c, p) and PE($c, p+1$) in cycle c, 2) link between PE(c', p) and PE($c', p+1$), and 3) L link between PE(c, p) and PE(c', p), as such a link failure makes use of that SCP to provide an alternative connection during reconfiguration.

The necessary and sufficient condition for a failed L link, say the link connecting PE(c, p) and PE(c', p), to be tolerable is that none of the two F's of PE(c, p) and PE(c', p) fails. This is because when the condition holds, the SCP between the two cycles is free for creating an alternative connection through any one of the four PE pairs: 1) PE(c, p) and PE(c', p), 2) PE($c, p+1$) and PE(c', p), 3) PE(c, p) and PE($c', p+1$) or 3) PE($c, p+1$) and PE($c', p+1$), depending upon whether there is any PE fault in the two cycles at dimension p or below. In Fig. 2, for example, the failure of L between PE(2, 0) and PE(3, 0) is tolerated by using an alternative connection between the two PE's; whereas the failed L between PE(1, 1) and PE(3, 1) is replaced by a connection between PE(1, 2) and PE(3, 1) (a reconfigured structure is illustrated in Fig. 3). On the other hand, the failure of F link of PE(c, p) [i.e., B link of PE($c, p+1$)] is tolerable if and only if 1) L of PE(c, p) is fault-free, and 2) no fault arises at PE(c, d) or PE(c', d), for all $0 \le d \le p$. It is clear that the XCCC can tolerate multiple failed F/B links in a cycle.

The XCCC not only achieves strong fault-tolerance with respect to any single PE or link fault but also can withstand multiple faults in many cases. Interestingly, the maximum possible number of PE faults that XCCC($h+1, k$) can tolerate is equal to the total number of spare PE's, i.e., 2^k. This sort of maximum tolerable fault patterns is not unique but is abundant. If neither links nor SCP's are faulty, for example, any situation in which every cycle has one and only one failed PE, regardless of its location, constitutes a maximum tolerable fault pattern.

C. Reconfiguration Procedure

We consider a procedure that guarantees proper reconfiguration of the XCCC in the presence of any set of multiple faults which are tolerable, and reports an error if a successful reconfiguration is not possible. Once a fault arises and is detected, the reconfiguration procedure is evoked. The procedure requires the participation of only a few PE's and can be performed in a distributed manner. Let us consider the PE failure first. Suppose that PE(c, p) fails in XCCC($h+1, k$). Each PE is assumed to have a local Boolean variable sw_on, which is initialized to "false" and is accessible to its immediate successor in the same cycle. (The variable sw_on is used for recording whether or not the switch on its U link has been activated during reconfiguration.) The faulty PE is bypassed and every PE at dimension $d, p < d < k$, in cycle c carries out the subsequent procedure one after another, starting with PE($c, p+1$). A discussion including examples and explanation of the procedure follows after the procedure is formally given below.

1) If the PE finds its L active, deactivate it; else, deactivate its U and write "true" to its local variable sw_on;
2) If the PE has an activated D, reconfiguration fails; else, activate its D and set the switch on the link to the "X" (or "V") state when the variable sw_on in its immediate predecessor is "false" (or "true"); and
3) Inform the PE on the other end of the activated D link, say PE$'$, so that it would activate its U and deactivate its L. If PE$'$ finds its U activated already, reconfiguration fails.

Finally, the spare PE in cycle c performs Steps 2) and 3) above.

If a PE has an inactive L in Step 1), it must use its U link to connect with the cycle to which the inactive L used to connect, indicating that the switch on its U link had been enabled previously and the variable sw_on has to be so recorded. As an example, consider that PE(1, 1) failed earlier than PE(5, 0) in Fig. 2. PE(5, 2) activated its U and disabled its L after the failure of PE(1, 1). During reconfiguration in response to the second failure, PE(5, 2) would find its L inactive an sets its variable sw_on to "true". After that, the spare PE in the cycle, noticing sw_on of its predecessor equal to "true", sets the control switch on its D link to the "V" state (according to Step 2) above), giving rise to a desired setting as shown in Fig. 3. In Step 2), a reconfiguration failure happens when two (or more) PE's become faulty in any single cycle. Reconfiguration may also fail if the PE at the other end of a newly activated D link had already enabled its U (for creating a connection alternative to a failed link), as stated in Step 3). In Fig. 3, for example, a subsequent failure at PE(6, 0) or PE(6, 1) makes reconfiguration unsuccessful, because during reconfiguration in response to such a failure, PE(6, 2) finds that PE$'$, namely PE(4, 1), has an activated U.

It is clear that the number of PE's involved in reconfiguration decreases when a fault arises at a higher dimension. The reconfiguration procedure is rather simple, and those involved PE's can easily be identified. Each cycle involved in reconfiguration may carry out the procedure concurrently and independently.

If a failed F or B link arises, say in cycle c between PE(c, p) and PE($c, p+1$), only two PE's [i.e., PE(c, p) and PE($c, p+1$)] are involved in reconfiguration. Specifically, PE(c, p) activates its U and PE($c, p+1$) activates its D, with the switch on them set to the "$\supset$" state, effectively giving rise to a healthy connection between the two PE's. Reconfiguration fails if the U link of PE(c, p) or the D link of PE($c, p+1$) was activated previously. A failed F/B link always needs the participation of two PE's only, independent of the faulty location. Likewise, when a fault occurs at L link connecting PE(c_1, p) and PE(c_2, p), the two connected PE's are responsible for reconfiguration by activating their respective U's, with the associated switch set to the "V" state. Reconfiguration is unsuccessful if any one of the two U links had been enabled before. Again, exactly two PE's participate in the procedure, no matter where the failed L is.

III. Reliability Analysis

A fault pattern is tolerable in our design if and only if it does not destroy the full rigid CCC structure. Many multiple link failures can be tolerated, as was described earlier, and in practice the link tends to be far more reliable than a PE. For simplicity, our reliability analysis does not consider the failure of regular links individually, and the failure rate of regular links is included in the PE failure rate. The failures of the added U/D links and their control switches are considered separately from the PE failure.

Suppose that every PE becomes faulty randomly and independently, with a constant failure rate λ, as does a spare PE. The reliability of XCCC($k+1, k$) is derived in the following. Every cycle can tolerate up to one PE failure, no matter whether there is a PE failure in other cycle or not. A cycle is reliable if 1) all active PE's are fault-free (whether the spare is faulty or not), or 2) the spare is fault-free and one active PE is faulty, with all needed SCP's for reconfiguration fault-free. These two events are independent. The mean number of SCP's involved in reconfiguration per cycle is

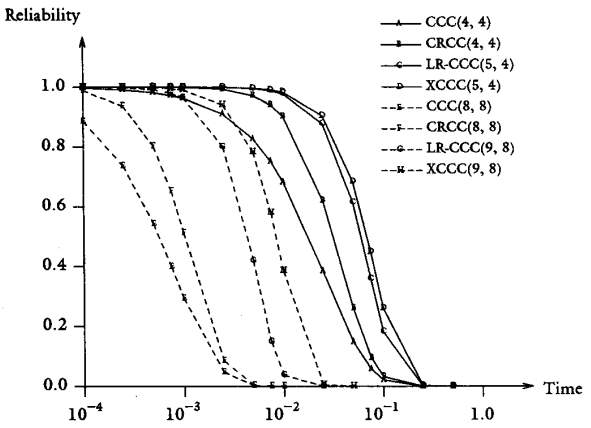

Fig. 5. Reliability comparison among CCC(k, k), CRCC(k, k), LR-CCC-($k+1, k$), and XCCC($k+1, k$).

$(k+1)/2$. As a result, the cycle reliability, R_{cyc}, is expressed by

$$R_{cyc}(t) = \left(e^{-\lambda t}\right)^k + e^{-\lambda t}\left[k\left(e^{-\lambda t}\right)^{k-1}\left(1-e^{-\lambda t}\right)\left(e^{-\lambda_s t}\right)^{\frac{k+1}{2}}\right] \quad (1)$$

where λ_s is the failure rate of an SCP. It should be noted that while every cycle in the XCCC can tolerate a PE failure independently, the SCP's employed in reconfiguration for two cycles are not necessarily irrelevant. In Fig. 2, for example, both PE failures need to use the SCP between cycles 1 and 5 during reconfiguration. The actual reliability of the two cycles can readily be shown to be greater than the product of the two cycle reliabilities, with each calculated by (1), since the latter value is underestimated due to multiplying the reliability of the same SCP twice. The reliability of XCCC($k+1, k$) is thus bounded from below by the product of all 2^k cycle reliabilities, yielding

$$R_{\mathrm{XCCC}}(t) > (R_{cyc}(t))^{2^k}. \quad (2)$$

The exact expression for $R_{\mathrm{XCCC}}(t)$ is too involved to derive, because it depends not only on the number of faults but also on their locations, and the lower bound provided by (2) will be employed instead. The lower bounds for XCCC(5, 4) and XCCC(9, 8) according to (2) are depicted in Fig. 5, where the PE failure rate λ is assumed to be 1.0 per unit time (e.g., 10^6 h), and the failure rate of an SCP, λ_s, is assumed 0.1 per unit time. (The value λ_s so chosen is somewhat pessimistic.) The exact reliabilities of the CCC, the CRCC [2], and the LR-CCC [6] are also shown in Fig. 5, where all the added links in the CRCC and the LR-CCC are assumed to be totally fault-free (this assumption favors the CRCC and the LR-CCC designs) and the failure rates of PE's in different designs are assumed to be proportional to the PE port numbers so as to provide meaningful reliability comparison. Specifically, the failure rates of PE's in the CRCC and the CCC are, respectively, 0.8 and 0.6 per unit time (rather than 1.0 as in the XCCC, because a PE in the CRCC has four ports and a PE in the CCC has three ports), whereas the PE failure rate in LR-CCC($k+1, k$) is $(k+2)/5$ per unit time (because its PE has $k+2$ ports). It can be seen that the XCCC indeed improves reliability significantly and compares favorably with its CRCC and LR-CCC counterparts.

A useful measure for comparing reliabilities of two different systems is the *mission time improvement factor (MTIF)* [5]. This measure indicates the improvement in the mission time of one system over the other, when a certain desired minimum mission reliability is concerned. Let $T_i(R)$ be the time for system i to decrease its reliability from 1 (at time $t = 0$) to R, the specified value. $T_i(R)$ is a useful absolute measure of system reliability because it provides information as to the suitability of system i for mission reliability no less than R. The MTIF of two systems, a and b, under the desired minimum mission reliability R is simply the ratio of $T_a(R)$ to $T_b(R)$.

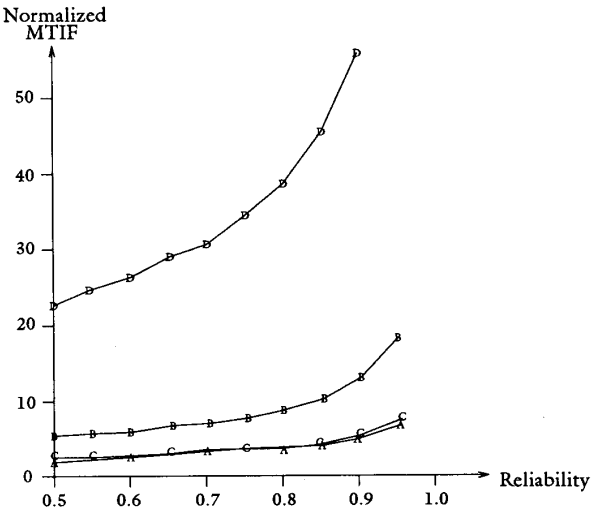

Fig. 6. Normalized mission time improvement factor versus reliability. (A: CRCC(4, 4) & CCC(4, 4); B: XCCC(5, 4) & CCC(4, 4); C: CRCC(8, 8) & CCC(8, 8) D: XCCC(9, 8) & CCC(8, 8).)

A fault-tolerant CCC is more reliable than its regular CCC counterpart but requires more PE's due to spares. Let ξ be the ratio of the number of PE's in a fault-tolerant CCC to the number of PE's in a compatible CCC. It appears meaningful and interesting to calculate the ratio of MTIF to ξ, called the *normalized MTIF,* because this measure takes into consideration the spare amount of a fault-tolerant design. When two fault-tolerant CCC designs are compared, the one with a larger normalized MTIF is more effective and is desirable.

We plot in Fig. 6 the normalized MTIF of the XCCC and the CCC as a function of required mission reliability R for the same two system sizes as in Fig. 5, namely, $k = 4$ and $k = 8$. It is clear that the XCCC, even with its added spares taken into account, demonstrates a considerable improvement in the mission time, for any given mission reliability. The normalized MTIF grows gradually as the needed mission reliability increases, because when the reliability requirement of a system is higher, the extent of benefits from fault tolerance swells. The normalized MTIF's of the CRCC and the CCC for $k = 4$ and 8 are also provided in the figure, while the normalized MTIF's of the LR-CCC and the CCC are omitted since the LR-CCC involves the same number of spares as the XCCC but possesses a lower reliability value. The XCCC is more effective than the CRCC, and the normalized MTIF gaps become larger as the system size grows.

IV. Faster Communication in the XCCC

Under the fault-free situation, all the cross connection pairs can be enabled, with switches on them set to the "X" state. The XCCC so configured has performance advantages due to faster communication, as will be explored subsequently. Once a fault arises and is detected, however, all the cross connection pairs must be disabled so that the reconfiguration procedure given in Section II can be employed to obtain a CCC and the performance advantages disappear. This section considers only the fault-free XCCC.

The global broadcast operation is one of the most frequently used operations in paralled machines. Broadcasting a message from one PE to all the other PE's on the XCCC$(h+1,k)$ can be done faster by making use of the "cross connections." In the XCCC, a broadcast message is sent over U links (rather than L links as in the CCC) to other cycles. A PE in one step can forward a copy of broadcast messages over up to 3 links, namely, the U, B, and F links. Every broadcast message carries variables *weight* and *count*. Let the broadcast algorithm for XCCC$(h+1,k)$ be Algorithm A, then, Algorithm A carried out at PE(c,ρ), the current visited PE, is formally given as shown at the bottom of the this page.

Because of statement I1 in Algorithm A, it can readily be verified that the farthest cycle in XCCC$(h+1,k)$ is reached by the sequence of U link traversals: $U_0, U_1, \cdots, U_{k-1}$, where U_i is the U link of a node located at the ith position of a cycle. This sequence of traversals involves only k steps, as opposed to $2k-1$ steps in regular CCC(h,k) where the farthest cycle is reached instead by the sequence of L link traversals: $L_0, L_1, \cdots, L_{k-1}$. It should be noted that statement I2 in Algorithm A is done after the variable *count* is checked (to ensure that no PE would receive duplicate copies due to the existence of statement I1). The process of broadcasting a message originating from PE$(0,1)$ in XCCC$(5,3)$ is depicted in Fig. 7. As mentioned above, the farthest cycle, cycle 7, is reached by the sequence: U_0, U_1, and U_2 [which form the path from PE$(0,0)$ through PE$(1,1)$ and PE$(3,2)$ to PE$(7,3)$]. This broadcast process takes 6 steps, in contrast to 8 steps required in regular CCC$(4,3)$.

Following a simple argument, we arrive at that broadcasting a message in XCCC$(h+1,k)$ takes anywhere from $(k+\lceil (h-1)/2\rceil)$ steps to $(k+2\lceil (h-1)/2\rceil)$ steps. Compared with CCC(h,k), where a broadcast message takes anywhere from $(2k-1+\lceil (h-1)/2\rceil)$ steps to $(2k-1+2\lceil (h-1)/2\rceil)$ steps, XCCC$(h+1,k)$ supports a faster broadcast process by reducing $(k-1)$ steps, which yield a roughly 30% to 40% speedup as $h=k$, for $h \geq 4$. When $h > k$ (like the example shown in Fig. 7), the speedup ratio tends to decrease slightly because the number of broadcast steps then increases. This broadcast speedup in the XCCC would lead to a performance improvement.

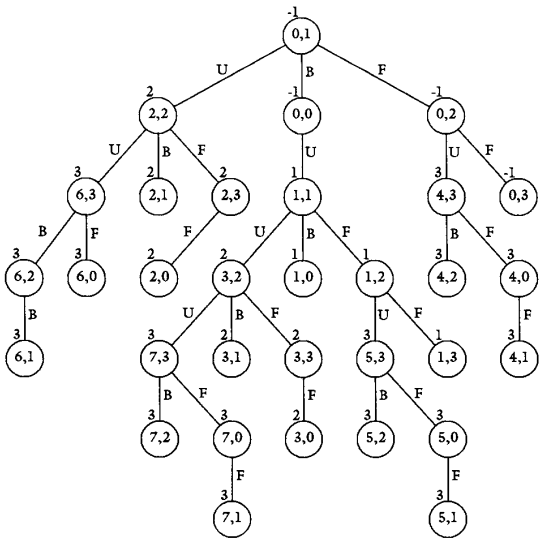

Fig. 7. Broadcasting a message originating from PE$(0,1)$ in XCCC$(5,3)$. (The number next to a circle indicates *weight*, and the letter next to an edge indicates the link traversed.)

Along the same line, we expect to get a faster PE-to-PE communication in the XCCC because the mean number of links traversed by a message is less. A deadlock-free routing algorithm for the CCC based on the virtual channel concept has been addressed by Dally and Seitz [4]. When messages are uniformly distributed and are routed in a deadlock-free manner, we can in fact show that the mean number of links traversed by a message in XCCC$(h+1,k)$ is reduced, on an average, by $k/2$, in comparison with that in CCC(h,k). This is because every two traversals over L then F in the CCC is replaced

---

Algorithm A: (originate or forward a message in the XCCC)
if (this is the originating PE)
  { set *weight* to –1
    send the message over the F link with count $= \lceil (h-1)/2 \rceil$
    send the message over the B link with count $= \lfloor (h-1)/2 \rfloor$
    send the message over the U link
  }
else
  if (the message comes from the D link)
  { set *weight* ρ
    send the message over the F link with count $= \lceil (h-1)/2 \rceil$
    send the message over the B link with count $= \lfloor (h-1)/2 \rfloor$
    if (the U link exists)
      }send the message over the U link} /* I1 */
  }
  else /* the message comes from either the F or B link */
  { set *count* to *count* –1
   if (*count* > 0)
    {**if** ($\rho >$ weight) /* I2 */
      {send the message over the U link}.
    if (the message comes from the F link)
      {send the message over the B link}
    else
      {send the message over the F link}.
    }
  }

by a single link traversal in the XCCC, namely, the U link traversal. This reduced amount in PE-to-PE routing is equivalent to at least 25% traversal reduction, for $h = k$, and may translate to improved performance.

V. XCCC Layout

The layout of CCC(h, k) given in Fig. 1 takes area $O(N^2/\log^2 N)$, which has been shown optimal with respect to the *area* $\times$ *(time)*^{2} complexity measure [1] following the VSLI grid model, where N is equal to $h \times 2^k$. In the model, a PE and a link are assumed to have the same width (and area). Under practical situations, we expect the links to be much thinner than the PE's because the PE involves far more components, as was pointed out by Krishnan and Hayes [7] and adopted in [3]. As a result, for any given p, placing all PE(c, p), $0 \leq c < 2^k$, on the same row can reduce layout area by a constant factor. Also, folding the highest dimension of the original CCC layout shown in Fig. 1 would further reduce layout area (again, by a constant factor), as was noted by Shen and Koren [3]. Let g be the ratio of PE width to link width, then, the area needed for such a folded CCC(h, k) layout equals

$$A_{CCC} = (g+1) \times 2^{k-1} \times \left(2h \times g + 2^k - 4\right). \tag{3}$$

The term $(g + 1) \times 2^{k-1}$ in (3) is the width of the CCC layout, which consists of one half of the total cycles with each cycle taking g units for PE's plus 1 unit for a link connecting two end PE's; while $(2h \times g + 2^k - 4)$ indicates the height of the layout, where $2^k - 4$ is the total horizontal tracks needed for lateral links.

Every PE in the XCCC has 5 ports (links), with two of them for making "cross connections." Because of the two additional ports, a PE is assumed to have width $\eta g, \eta > 1$, times of link/switch width for fair area comparison. The switch connection pairs (SCP's) in the XCCC occupy extra layout area, as do the added spare PE's. We can lay out the XCCC in a way that the highest dimension is folded and all PE's in the same dimension are placed along the same row, as shown in Fig. 8. This way of layout reduces the area not only for the lateral links in the highest dimension, but also for SCP's added at the highest level. Consequently, the area overhead of the XCCC is kept moderate.

Let us consider the SCP's between dimensions l and $l + 1$ of XCCC$(h + 1, k)$ layout, $(0 \leq l < k - 1$. There are totally 2^{k-2} SCP's in each half. As $g \geq 3$, we are able to arrange SCP's so that no extra vertical track is necessary, with 2×2^l horizontal tracks used. This is because the three vertical tracks then occupy no more than a PE width and can run between two vertically adjacent PE's (without taking extra space). This way of arranging SCP's is pursuant to a VLSI model such that links run only vertically or horizontally, and a PE takes ηg times as much space as a link, called the "modified" VLSI grid model. Under the "modified" VLSI grid model, $2^{k+1} - 4$ total horizontal tracks are necessary for SCP's in the XCCC$(h+1, k)$ layout. As far as those 2^{k-1} SCP's at the highest level are concerned, three vertical tracks are sufficient for every pair of such connections. Fig. 8 depicts the resulting layout of XCCC(5, 3) in accordance with the "modified" VLSI grid model. The area of a XCCC$(h + 1, k)$ layout is given by

$$A_{XCCC} = (\eta g + 4) \times 2^{k-1} \times \left(2(h+1) \times \eta g + 2^k + 2^{k+1} - 8\right) \tag{4}$$

where $(\eta g + 4) \times 2^{k-1}$ accounts for the layout width, and $(2(h + 1) \times \eta g + 2^k + 2^{k+1} - 8)$ is the layout height. It can be easily verified that, for a large k and any constant g and η, A_{XCCC} becomes

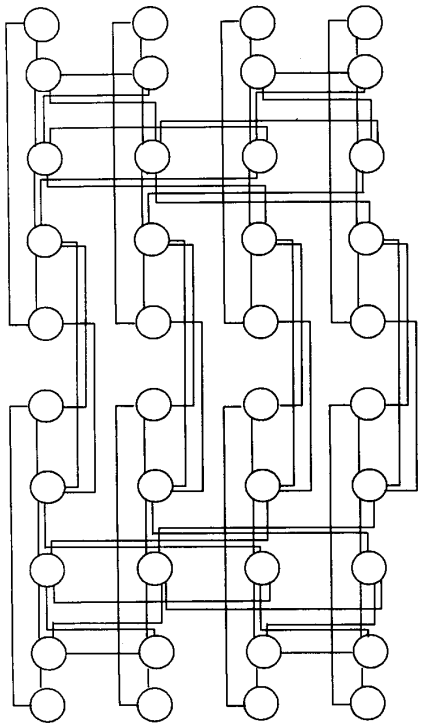

Fig. 8. The XCCC(5, 3) folded layout following the "modified" VLSI grid model.

$O(N^2/\log^2 N)$, $N = h \times 2^k$. This indicates an interesting result that A_{XCCC} is identical in the big-O sense to the area of a CCC with N PE's.

For a situation where the ratio g is much larger than 1, say $g \geq 10$, it is meaningful to compare the actual area amounts needed for the XCCC layout and for the CCC layout, since these amounts provide better insight into the real area overhead. (Note that the area taken by a PE can be two orders of magnitude higher than the area of a switch; a 17-bit switch plus interfaces involves some 1266 transistors in the design given in [8].) From (3) and (4), we calculate the ratio of XCCC layout area to CCC layout area for different sets of h and k under various g's, given that η is 1.1. The results are sketched in Fig. 9. Note that the results are quite conservative because a PE in the XCCC is assumed to take 21% (i.e., $\eta^2 = 1.21$) more area than a PE in the CCC. It can be observed from these curves that for $h = k = 4$, the XCCC needs insignificant extra area as compared with its CCC counterpart. When the width ratio is 20, example, XCCC(5, 4), which involves 80 PE's, takes 87% more area than the CCC with 64 PE's. The area overhead becomes larger for $h = k = 8$, and the width ratio must exceed 30 in order to maintain 100% or less extra area. For any given size, the area overhead decreases when g grows, as expected, because the total area for the additional tracks then shrinks. For a fixed g, the area overhead extends as the system size increases due to more area taken by extra tracks. This design approach is attractive when the PE is relatively complex and the width ratio is reasonably large.

VI. Concluding Remarks

We have presented a very reliable cube-connected cycles (CCC) architecture, dubbed the XCCC. The XCCC is realized by adding a row of spare PE's to the CCC and making "cross connections" among

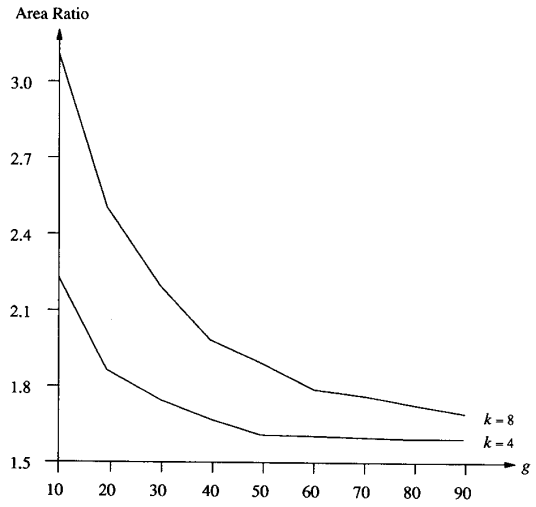

Fig. 9. The ratio of XCCC($k+1, k$) layout area to CCC(k, k) layout area versus g for $k = 4$ and $k = 8$.

cycles, with a control switch on each pair of "cross connections." It can tolerate many link failures and up to one PE failure per cycle. Reconfiguration in response to any fault is simple and can be performed in a distributed manner. The XCCC is more reliable than earlier fault-tolerant designs and takes a moderate extra area, in comparison with the CCC layout, if PE width is far larger than link/switch width. Due to these "cross connections," the XCCC achieves faster broadcasting and PE-to-PE routing, when there is no fault.

The XCCC structure can be viewed as a superimposition of two different architectures, namely, the CCC and the butterfly structure. It is feasible to have two sets of data share the PE's, with each set being independently executed, provided that the required communication patterns can be supported simultaneously. The recursive doubling class of parallel algorithms, like MIN, MAX, SUM, etc. [1], are such typical examples that two data pipelines can independently be executed on the XCCC. This way of execution would get a speedup over serially executing one set of data after another. To make this possible, it requires that every PE in a time unit can receive two data, one from its B link and the other from its D link. Two streams of data enter the XCCC through the bottom PE's at the same time. One set of data is routed along the F/B links to higher dimensions, whereas the other set is along the U/D links, with L links used by both sets. The XCCC supports two sets of computations of the recursive doubling class concurrently, arriving at a significant performance improvement.

It is possible that a failed PE is replaced by a PE at the immediate lower (rather than higher) dimension, if the spare PE's are added to the bottom (rather than the top) of the CCC and the pattern of "cross connections" changes accordingly. Also, the restriction that the XCCC tolerates only one PE failure per cycle can be relaxed if we augment the structure with more redundant PE's and "cross connections." For example, if every cycle is equipped with two extra PE's, and "cross connections" are provided from each PE to its two immediate adjacent dimensions, then, two PE failures per cycle can be tolerated. This design naturally results in a higher layout area overhead.

References

[1] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, vol. 24, pp. 300–309, May 1981.

[2] P. Banerjee, "The cubical ring connected cycles: A fault-tolerant parallel computation network," *IEEE Trans. Comput.*, vol. C-37, pp. 632–636, May 1988.

[3] J.-J. Shen and I. Koren, "Yield enhancement designs for WSI cube connected cycles," in *Proc. Int. Conf. Wafer Scale Intergration*, Jan. 1989, pp. 289–298.

[4] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, pp. 547–553, May 1987.

[5] S. Makam, A. Avizienis, and G. Grusas, "UCLA ARIES 82 User's guide," Tech. Rep. CSD-820830, U.C.L.A., Aug. 1982.

[6] S.-Y. Kuo and W. K. Fuchs, "Reconfigurable cube-connected cycles architectures," *J. Parallel Distributed Comput.*, vol. 9, pp. 1–10, May 1990.

[7] M. S. Krishnan and J. P. Hayes, "A normalized-area measure for VLSI layouts," *IEEE Trans. Comput.-Aided Design*, vol. 7, pp. 411–419, Mar. 1988.

[8] M. Blatt, "Effects of switch failure on soft-configurable WSI yield," in *Proc. 1990 Int. Conf. Wafer Scale Integration*, Jan. 1990, pp. 152–159.

Gossiping in a Distributed Network

A. Bagchi, S. L. Hakimi, and E. F. Schmeichel

***Abstract*—Consider a network in which each unit initially knows only its own identity and the identity of its immediate neighbors. Suppose each unit has a message intended for all other units. We give a distributed algorithm to accomplish this in point-to-point networks which is optimal in the number of transmissions it requires. We also show that this algorithm accomplishes this efficiently for broadcast (radio) networks, although the problem of finding a solution with the least number of transmissions, in broadcast networks, is shown to be NP-hard.**

***Index Terms*—Distributed algorithms, information dissemination, optimal number of transmissions, point-to-point networks, radio networks.**

I. Introduction

Our terminology and notation will be standard. A good reference for any undefined terms is [2].

Consider a computer network of n units represented by a connected, symmetric directed graph $G(V, E)$ with vertex set V and edge set E, where by symmetric directed graph we mean that if $(u, v) \in E$, then $(v, u) \in E$. In particular, the elements of V represent the units (i.e., processors) in the network, and each directed edge (u, v) represents a direct one-way communication link through

Manuscript received May 15, 1989; revised April 15, 1990 and March 10, 1992. This work was supported by the National Science Foundation under Grant NCR-8716876.

A. Bagchi is with Bell Communications Research, Red Bank, NJ 07701. The work of this author was done while he was with the University of California, Davis.

S. L. Hakimi is with the Department of Electrical Engineering and Computer Science, University of California, Davis, CA 95616.

E. F. Schmeichel is with the Department of Mathematics and Computer Science, San Jose State University, San Jose, CA 95192. The work of this author is supported in part by the National Science Foundation under Grant DMS-8904520.

IEEE Log Number 9202841.