Reliability and Modelability Advantages of Distributed Switching for Reconfigurable 2D Processor Arrays

Behrooz Parhami
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract— **Processor arrays have been used, either as the main computation engine or as special-purpose adjuncts, for a variety of applications requiring very high performance. As the size of such an array increases, the possibility of processor malfunctions, leading to loss of computational capabilities, can no longer be ignored. While many switching architectures and reconfiguration algorithms have been proposed for building processor arrays, modeling of their reliability has been inadequately addressed. In this paper, I study differences between 2D processor arrays with centralized and distributed switching, pointing to advantages of the latter in terms of reliability, regularity, modularity, and VLSI realizability. As important side results, I formulate the notions of reliability inversion (a less reliable system prevailing over a more reliable one due to modeling uncertainties) and modelability (the property of a system that makes it possible to derive tight reliability bounds, thus making reliability inversion less likely).**

Keywords—dependability, reconfigurable 2D arrays, reliability bound, reliability modeling, spare row/column, switch tracks

I. Introduction

Reconfiguration is a powerful method that can be used at multiple architectural levels to overcome the effects of defective, unavailable, or malfunctioning parts through the establishment of alternate or bypass signal-paths and computations. Examples include reconfiguration for defect tolerance to improve VLSI yield [1], using alternate components and interconnects during mapping of computations onto a partially-used or damaged FPGA [2], and allowing graceful degradation on parallel array architectures [3]. Many other methods used at the architectural level to help adapt a system to changing conditions are also instances of reconfiguration, although some may not be recognized as such. For example, when we replace an operational unit within a redundant system by a spare unit upon failure detection, we are really reconfiguring an $(s + 1)$-unit "parallel" subsystem to switch from the working module to one of the s spares [4].

In the references cited, and the examples we will draw upon, reconfiguration is performed to return a processor array with malfunctioning nodes to its original healthy configuration, so as to be able to execute the original algorithms unaltered. Specifically, we are not considering the use of an injured or partially-damaged array, which still provides inter-node connectivity, though at a degraded level due to reduced bandwidth and dilated paths [5], or a smaller healthy subarray, which would require computation remapping with a higher per-processor load and increased congestion. We are also not considering the kind of reconfiguration that extends the computational power (in a complexity-theory sense) of the array, allowing it to achieve significant speed-up in performing certain global computations [6].

Given the importance of processor arrays in building high-performance parallel systems for a number of applications, many alternative reconfiguration schemes have been proposed over the years [7]. These alternative architectures differ in the types, multiplicity, and placement of switches used, the control scheme for effecting array reconfiguration (centralized or distributed), and the relative complexity/speed of the reconfiguration algorithm, once the set of unavailable processors becomes known.

II. Reconfigurable Processor Arrays

I will illustrate the advantages of distributed switching over a centralized scheme on a small 5×5 example array, with one row and one column of redundant or spare processors, leading to a 6×6 host array. I then follow the specific result with arguments/proofs that the advantage grows with both array size and an increase in redundant resources (number of spare rows and columns), thus establishing a fairly general result. The extent of the reliability advantage growth remains unquantified in this paper. Working towards quantifying the advantage growth constitutes a fruitful area for further research.

An example 2D array, with embedded reconfiguration switches, placed along tracks between rows and columns, is depicted in Fig. 1. The small circles represent 4-port switches, whose 3 states and usage within the processor array of Fig. 1 are shown in Fig. 2. The actual reconfiguration method used for the array and its malfunction-tolerance capabilities will be discussed in Sections III and IV of the paper.

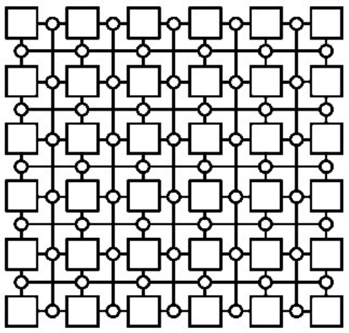

Fig. 1. Square 5 × 5 processor array with a spare row (bottom) and a spare column (right), along with embedded reconfiguration switches.

The scheme shown in Fig. 1 represents but one example of how switches can be embedded within the processor array to allow signals normally going from one processor to a row- or column-neighbor to be diverted to a different processor, thus offering processor bypassing capability. When such a replacement takes place, the node that takes over must either be initialized to the current state of the replaced node or else the entire computation must be restarted or rolled back with a different assignment of node identities. This isn't a trivial problem, but its complexities are independent of the actual replacement and reconfiguration scheme used. So, we will not discuss the problem further.

Some reconfiguration schemes for processor arrays entail the use of multiple tracks of switches or more complex switches than the one shown in Fig. 2. An example reconfiguration scheme with two switch tracks (red and blue) between rows and columns of processors is depicted in Fig. 3. Using a larger number of switches or more complex ones presents a reliability tradeoff, which is hard to quantify in general. The advantages of greater reconfigurability may be nullified by higher failure rates for more complex or more numerous switches.

For this study, we limit ourselves to a single track of switches between processor rows and columns. We will assume the use of identical switches, for ease of reliability modeling and VLSI realization. However, we place no restriction on switch complexity, given that the switch failure rate is taken to be an independent parameter in the reliability model we use.

For an $n \times n$ processor array with a spare row and a spare column, the redundancy factor is $(2n + 1)/n^2 = O(1/n)$. With k spare rows and k spare columns, the redundancy ratio becomes $(2kn + k^2)/n^2 = O(k/n)$, so we are dealing with relatively low redundancy factors.

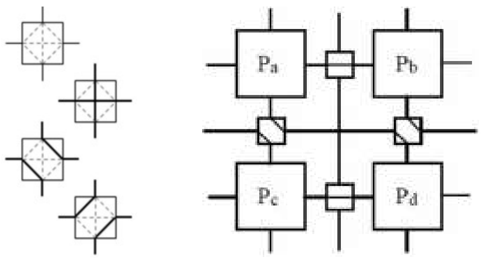

Fig. 2. Four-port, three-state switches and how they are used in the reconfigurable array of Fig. 1.

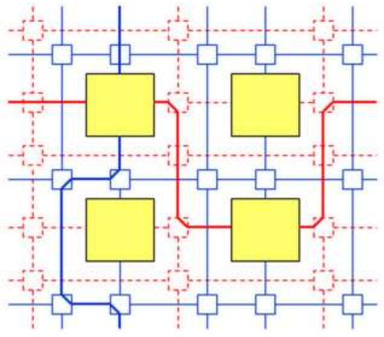

Fig. 3. A small portion of a 2D processor array with two tracks of embedded switches between rows and columns of processors.

III. Centralized Reconfiguration

Consider the 6 × 6 host array of Fig. 1 embedding a 5 × 5 guest array. Originally, the nodes in the topmost 5 rows and the leftmost 5 columns are active, with the configuration changing as nodes malfunction. A particular reconfigured array is depicted in Fig. 4, where it is assumed that in addition to embedded signal re-routing switches, there is a mechanism that allows the bypassing of a processor within its row or column.

Intuitively, signal re-routing is done by shifting rows downward (toward the spare row) in order to circumvent the effect of unavailable row processors. Similarly, columns are shifted rightward (toward the spare column) to avoid bad nodes within a column. We won't discuss the algorithms that effect reconfiguration (see, e.g., [8]), mentioning only that any double-node malfunction can be tolerated via reconfiguration, but there are worst-case patterns of 3 unusable nodes that exceed the scheme's reconfigurability [4].

As for controlling the reconfiguration switches, two schemes are possible. In the first category of methods, switches are under the control of a central unit. Such a scheme has the drawbacks of creating a single point of failure in the control mechanism and requiring long wires, as well as excessive delay for shifting-in the configuration information in the event of a processor malfunction. In the second category of methods, each switch is controlled by a nearby processing node. The drawback here is that a bad processor can make other nodes or certain configurations of the array inaccessible, reducing the overall reliability and increasing the modeling effort to take all scenarios into account.

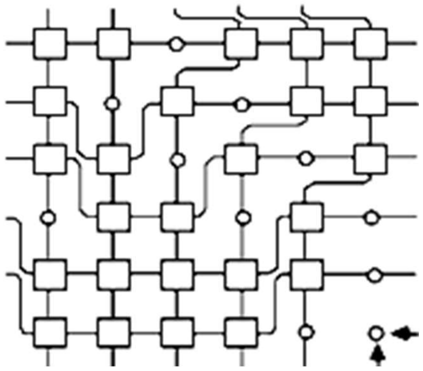

Fig. 4. Proccessor array of Fig. 1 configured to salvage a healthy 5 × 5 array from a 6 × 6 injured one.

A simple combinational model for assessing the reliability of the processor array of Figs. 1 and 4 is to lump the switching hardware into a hard core and to model the processing part as a 34-out-of-36 system, given that up to 2 processor malfunctions are guaranteed to be tolerable. Let the processor failure rate be λ and the switch failure rate σ. Then, given that there are 60 embedded switches within the 6×6 array:

$$\text{Module/Processor reliability} = r = e^{-\lambda t} \qquad (1)$$

$$\text{Overall switching reliability} = e^{-(60\sigma)t} \qquad (2)$$

$$\text{System reliability} = e^{-(60\sigma)t} R_{34\text{-out-of-}36}(r) \qquad (3)$$

where $R_{k\text{-out-of-}n}(r)$ is the k-out-of-n reliability for modules of uniform reliability r. Computationally:

$$\begin{aligned} R_{34\text{-out-of-}36}(r) &= r^{36} + 36r^{35}(1-r) + (36\times35/2)r^{34}(1-r)^2 \\ &= r^{34}[r^2 + 36r(1-r) + 630(1-r)^2] \\ &= r^{34}[595r^2 - 1224r + 630] \\ &= r^{34}[1 + (1-r)(629 - 595r)] \qquad (4) \end{aligned}$$

Substituting Eq. (4) into Eq. (3) results in the reliability plots of Fig. 5 for different $\rho = \sigma/\lambda$ ratios, ranging from 0.001 to 0.1, reflecting the relative switch complexity. For typical processor nodes, ρ will be in the lower range of the considered values, whereas for simple processing elements used in some arrays, the value of ρ will approach the higher end, though it is unlikely to get very close to 0.1.

The shapes of the unreliability curves in Fig. 5 are according to expectation. For small values of λt, the processing part of the system, as a 34-out-of-36 system, is highly reliable, so switches are the main contributors to unreliability. This is why we have significant differences between the three cases with different levels of switch complexity relative to a processor. As the value of λt increases, multiple processor malfunctions exceeding the system's tolerance capability become dominant, reducing the impact of switching differences.

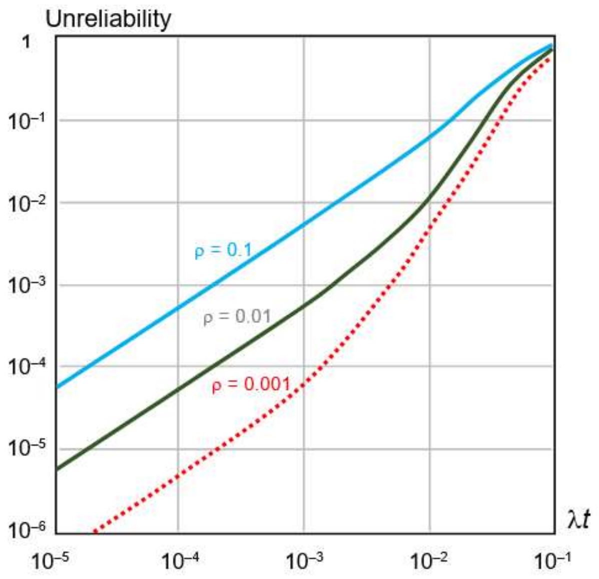

Fig. 5. Reliability of our example processor array with centralized reconfiguration as a function of λt, for different values of $\rho = \sigma/\lambda$.

IV. Distributed Reconfiguration

We now introduce a reconfiguration switching scheme for processor arrays in which good processors dictate the system configuration by controlling their own internal switches. Bad processors are rendered immaterial by their outputs not being selected/used by any good processor. The scheme works as long as good nodes possess information about malfunctioning nodes, but such information is needed for any reconfiguration scheme, whether centralized or distributed.

Distributed switching can be achieved in a variety of ways. For fairness of direct comparisons, we choose a scheme that offers the same reconfiguration capability as the centralized scheme depicted in Figs. 1 and 2. The switching mechanism in each module is composed of two 3-input multiplexers (muxes) that allow each processor to choose its east and north neighbors from among three possible modules. As an example, the north neighbor of a processor can be any one of the three processors in the immediately preceding row, the one directly above and the ones preceding and following it in the row. This is what the scheme in Figs. 1 and 2 allow, given the availability of only one spare row and one spare column.

Now, each node becomes a tad more complex, increasing its failure rate to $\lambda + \alpha\sigma$, where σ is the failure rate of the original track switches of Fig. 2 and α is the distribution overhead, representing the increase in the hardware complexity of the switching mechanisms as a result of the distribution process. We now have the following reliability equations:

$$\text{Module reliability} = r' = e^{-(\lambda+\alpha\sigma)t} \qquad (5)$$

$$\text{System reliability} = R_{34\text{-out-of-}36}(r') \qquad (6)$$

In our numerical example, we take $\alpha = 2$ as a reasonable pessimistic value, given the presence of $60/36 \cong 1.67$ switches per node in the centralized scheme (see Fig. 1), with each 2×2 switch built from two 2-to-1 muxes. The distributed scheme needs two 3-input muxes per node, as depicted in Fig. 6.

Unreliabilities of the resulting reconfigurable array for three different values of $\rho = \sigma/\lambda$ are plotted in Fig. 7. Again, the unreliability curves match our expectation. In a distributed reconfiguration scheme, switches are integrated into the processing nodes, thus as long as 34 of the 36 processor-switch modules are functional, we can successfully reconfigure the system. We do not care about the health of the switching mechanism any more than we care about processor health. In other words, the system has no single point of failure.

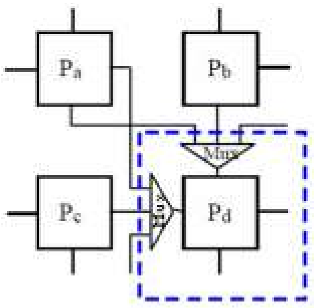

Fig. 6. Processors incorporated into modules with built-in switches.

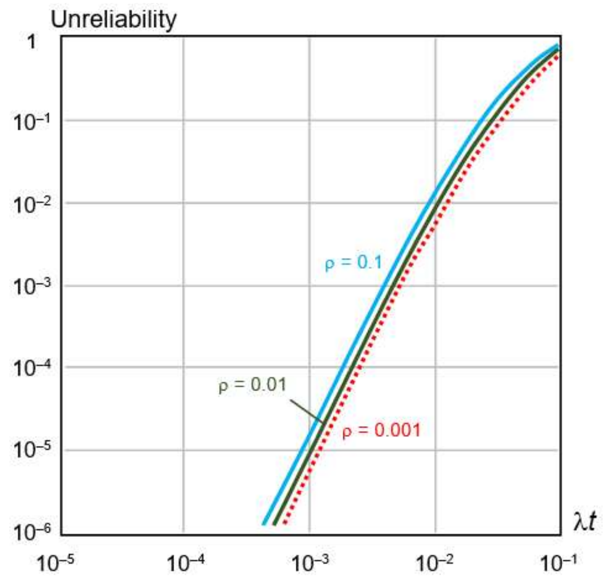

Fig. 7. System unreliability for our example processor array with distributed reconfiguration as a function of λt, for different values of $\rho = \sigma/\lambda$.

TABLE I. LOW SENSITIVITY OF RELIABILITY TO VARIATIONS IN α

λt $\rightarrow$	0.0001	0.0010	0.0100	0.1000
$R(\alpha = 1)$	1.000,000	0.999,993	0.994,343	0.314,752
$R(\alpha = 2)$	1.000,000	0.999,993	0.994,189	0.308,424
$R(\alpha = 3)$	1.000,000	0.993,992	0.994,031	0.302,192

Just to make sure that the results aren't overly sensitive to our decision to use $\alpha = 2$, we perform a sensitivity analysis by considering two other values for the parameter α, representing no distribution overhead at all ($\alpha = 1$) and much greater overhead ($\alpha = 3$). These values are extreme, but they serve to confirm, as seen in Table I, that reliability values do not change much by a small adjustment in the value of α.

V. RELIABILITY COMPARISONS

The reliability advantages of distributed reconfiguration switching are apparent from comparing Figs. 5 and 7. The greatest advantages are seen for smaller values of λt, which is consistent with the normal operating region for highly reliable systems. Figure 8 shows the two unreliability curves corresponding to $\rho = 0.01$ from Figs. 5 and 7 superimposed to better visualize the reliability differences for a reasonable instance of switch complexity. The dotted gray curve will be explained in Section VIII.

We see from inspecting Fig. 8 that as λt approaches 1, differences in reliabilities diminish, rendering the centralized and distributed reconfiguration schemes indistinguishable and also practically useless. Thus, in the region of λt values where systems tend to operate, distributed reconfiguration switching offers significant advantages.

The reader may wonder whether the results above, obtained for a specific array size and with a particular switching scheme, have more general significance. The following informally-justified claims suggest that they do.

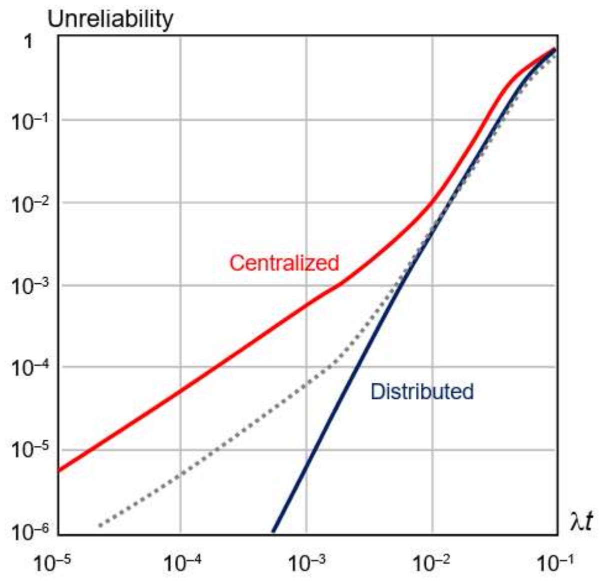

Fig. 8. Comparing system unreliability for a reconfigurable array of PEs as a function of λt, under the assumptions of $\rho = 0.01$ and $\alpha = 2$.

Claim 1: The reliability advantage of distributed reconfiguration over a centralized scheme grows with an increase in array size, while keeping the switching scheme and, thus, tolerance level the same.

Proof outline: Write out the expression for the reliability lower bounds of an $h \times h$ array and then increase the array size to $(h + 1) \times (h + 1)$. Show that the reliability bound decreases less as a result of the array enlargement for the distributed scheme compared with the centralized one. The intuition behind this result is that the centralized switch complexity growth has a direct effect on reliability, whereas the increase in module complexity due to the distribution process passes through the filter of, and is moderated by, the k-out-of-n function.

Claim 2: The reliability advantage of distributed reconfiguration over a centralized scheme grows with an increase in the number of spare rows and/or spare columns, all else being kept the same.

Proof outline: Two spare rows and two spare columns convert the $(n^2 + 2n - 1)$-out-of-$(n^2 + 2n + 1)$ reliability function to the $(n^2 + 2n - 3)$-out-of-$(n^2 + 2n + 1)$ function, representing the tolerance of 4 processor malfunctions. Again, the improvement in the k-out-on-n function value dominates the corresponding improvement in the centralized scheme, which now has somewhat greater switch complexity.

Claim 3: The reliability advantage of distributed array reconfiguration over a centralized scheme grows with an increase in the switching complexity (more switch tracks or greater switch hardware sophistication) that allows more malfunctioning processors to be tolerated.

Proof outline: The proof strategy for this claim is similar to that used for Claim 2. Other than the k-out-of-n function "softening" the impact of switch imperfections, the distribution of the more complex switching capabilities among multiple modules leads to greater reliability improvement.

VI. Modelability as a System Attribute

The exact reliability of a system is often unknowable. If we had 100s of identical copies of a system and could run them for decades, observing system failures, we could ascertain the actual reliability with high confidence. The large number of copies and long running times are needed because, at typically high system reliabilities that are of practical interest, failures are extremely rare, so to obtain statistically valid results, extensive data collection is required.

An alternative to the impractical experimental validation process outlined in the preceding paragraph is to make simple, pessimistic assumptions about subsystems and their interactions, in an analytic or simulation model, to derive a lower bound on reliability. The latter is the only option when evaluation is being done at the design stage, when no physical system exists. Of course, models don't completely eliminate the need for experimentation, as model parameters may be derived, and models themselves fine-tuned, based on the results of experimental observations.

We call a system "modelable" when it lends itself to the derivation of better (tighter) reliability lower bounds. Modelability is of the same nature as (design for) analyzability, also known as "design for analysis" [9], itself predated by notions such as design for manufacturing (manufacturability). Analyzability requires honoring certain design constraints that would allow the use of simpler tools for analysis. In the domain of electronic circuits, design for packageability [10] is quite similar. Both notions constrain the design process, but, somewhat counterintuitively, the end result is often economy and shorter time-to-market.

VII. Reliability Inversion

A modelable system lends itself to the derivation of tighter reliability bounds. Considering Fig. 9, which shows the (unknowable) reliabilities of Systems A and B and lower bounds for them derived through reliability models, system B seems to enjoy greater modelability, because its reliability lower-bounds are closer to actual values. At this point, modelability is a qualitative notion, but there is no reason whatsoever why we cannot quantify it. Other "ilities," such as testability and serviceability, started out as qualitative notions, but were later quantified with much success. Pursuing the quantification of modelability constitutes a fruitful area for further research.

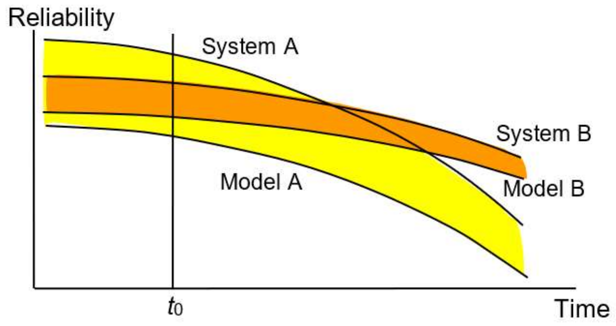

Fig. 9. True reliability vs. modeled lower bound.

In short, reliability analysis is often based on worst-case assumptions, so as to produce guaranteed lower-bounds on system survival probability. Reliability engineers try to derive tight lower-bounds, but sometimes system structure is unfriendly to the derivation of tight bounds.

For a mission time t_0 (see Fig. 9), we may obtain the lower bound 0.995 for the reliability of System B, whose actual reliability is 0.997, and the lower bound 0.993 for System A, with an actual reliability of 0.999. Because actual reliabilities are unknowable, as previously noted, we have to compare systems based on lower bounds, leading to the declaration of System B as more reliable than System A at t_0. I have referred to this condition as reliability inversion [11], in analogy to the similarly disruptive phenomenon of "priority inversion" in real-time task scheduling [12] that wreaked havoc during the Mars Pathfinder mission of the late 1990s [13].

In reality, Fig. 9 shows that each of the systems A and B is more reliable for some range of mission times. But given the model-based reliability bounds, System B appears to be uniformly more reliable than System A.

VIII. Conclusion

In this paper, I have shown that reconfigurable processor arrays with distributed switching tend to be more reliable than those with centralized switching due to the combination of two factors: Inherently higher reliability and better modelability, the latter allowing the derivation of tighter lower bounds for system reliability using relatively simple combinatorial models. In a companion paper [11], I have shown that centralized switching can exhibit better reliability under certain unusual conditions (the gray dotted curve in Fig. 8).

As pointed out in the previous sections of the paper, several problems remain to be addressed in future research. First, the claims presented in Section V show improved reliability with certain changes in array configuration. These improvements should be quantified, either by generalizing the proofs or by direct derivation of reliability formulas for various array sizes and configurations. Second, an attempt should be made to quantify the notion of modelability, so that it can be applied to evaluating competing system designs in a systematic way or to produce design guidelines and methodologies. Third, extension of our results to arbitrary system sizes and form factors, alternative hardware switching architectures, and a variety of reconfiguration processes and algorithms [14] [15] [16] [17] can lead to more-widely applicable results, as well as new application and research challenges.

References

[1] M. Wang, M. Cutler, and S. Y. H. Su, "Reconfiguration of VLSI/WSI mesh array processors with two-level redundancy," *IEEE Trans. Computers*, vol. 38, no. 4, April 1989, pp. 547–554.

[2] S. Hauck and A. DeHon (eds.), *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, Elsevier, 2008.

[3] M. Sami and R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays," *Proceedings of the IEEE*, vol. 74, no. 5, May 1986, pp. 712–722.

[4] B. Parhami, *Dependable Computing: A Multilevel Approach*, 2020. Draft of graduate-level textbook available via the author's Web site at UCSB, under the "Textbooks" tab.

[5] Y. C. Tseng, M. H. Yang, and T. Y. Juang, "Achieving fault-tolerant multicast in injured wormhole-routed tori and meshes based on Euler path construction," *IEEE Trans. Computers*, vol. 48, no. 11, November 1999, pp. 1282–1296.

[6] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The power of reconfiguration," *Proc. Int'l Colloquium Automata, Languages, and Programming*, Springer, 1991, pp. 139–150.

[7] M. Chean and J. A. B. Fortes, J. A. B., "A taxonomy of reconfiguration techniques for fault-tolerant processor arrays," *IEEE Computer*, vol. 23, no. 1, January 1990, pp. 55–69.

[8] M. Fukushi and S. Horiguchi, "Reconfiguration algorithm for degradable processor arrays based on row and column rerouting," *Proc. 19th IEEE Int'l Conf. Defect and Fault Tolerance in VLSI Systems*, October 2004, pp. 496–504.

[9] R. Suri and M. Shimizu, "Design for analysis: A new strategy to improve the design process," *Research in Engineering Design*, vol. 1, no. 2, June 1989, pp. 105–120.

[10] P. H. Dehkordi and D. W. Bouldin, D. W., "Design for packageability—Early consideration of packaging from a VLSI designer's viewpoint," *IEEE Computer*, vol. 26, no. 4, April 1993, pp. 76–81.

[11] B. Parhami, "Reliability inversion: A cautionary tale," *IEEE Computer*, vol. 53, no. 6, June 2020, pp. 28–33.

[12] D. Locke, L. Sha, R. Rajikumar, J. Lehoczky, and G. Burns, "Priority inversion and its control: An experimental investigation," *ACM SIGADA Ada Letters*, vol. 8, no. 7, June 1988, pp. 39–42.

[13] G. Reeves, "What really happened on Mars," *The Risks Digest*, vol. 19, no. 54, 1998 (item 6 within the issue).

[14] G. Jiang, J. Wu, and J. Sun, "Efficient reconfiguration algorithms for communication-aware three-dimensional processor arrays," *Parallel Computing*, vol. 39, no. 9, September 2013, pp. 490–503.

[15] J. Qian, W. Cao, J. Hu, J. Zhang, Z. Xu, and Z. Zhou, "Satisfiability-based method for reconfiguring power efficient VLSI array," *IEICE Electronics Express*, vol. 13, no. 23, August 2016.

[16] J. Wu, N. Liu, S.-K. Lam, and G. Jiang, "Shortest partial path first algorithm for reconfigurable processor array with faults," *Proc. IEEE Trustcom/BigDataSE/ISPA*, August 2016, pp. 1198–1203.

[17] J. Wu, T. Srikanthan, G. Jiang, and K. Wang, "Constructing sub-arrays with short interconnects from degradable VLSI arrays," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 4, April 2013, pp. 929–938.