International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 10, No. 2, July 2021, pp. 123~129
ISSN: 2089-4864, DOI: 10.11591/ijres.v10.i2.pp123-129

Comparing reliabilities of centralized and distributed switching architectures for reconfigurable 2D arrays

Behrooz Parhami
Department of Electrical and Computer Engineering, University of California, Santa Barbara

Article Info

Article history:

Received Jun 30, 2020
Revised Aug 15, 2020
Accepted Jan 23, 2021

Keywords:

Dependability
Modelability
Reconfiguration switching
Redundant processor array
Reliability bound
Reliability modeling
Spare row or column

ABSTRACT

Whether used as main processing engines or as special-purpose adjuncts, processor arrays are capable of boosting performance for a variety of computation-intensive applications. For large processor arrays, needed to achieve the required performance level in the age of big data, processor malfunctions, resulting in loss of computational capabilities, form a primary concern. There is no shortage of alternative reconfiguration architectures and associated algorithms for building robust processor arrays. However, a commensurately extensive body of knowledge about the reliability modeling aspects of such arrays is lacking. We study differences between 2D arrays with centralized and distributed switching, pointing out the advantages of the latter in terms of reliability, regularity, modularity, and VLSI realizability. Notions of reliability inversion (modeling uncertainties that might lead us to choose a less-reliable system over one with higher reliability) and modelability (system property that makes the derivation of tight reliability bounds possible, thus making reliability inversion much less likely) follow as important byproducts of our study.

Corresponding Author:

Behrooz Parhami
Department of Electrical and Computer Engineering,
University of California
Santa Barbara, CA 93106-9560, USA
Email: parhami@ece.ucsb.edu

1. INTRODUCTION

Reconfiguration can be used at system, architecture, and circuit levels to establish alternate or bypass signal paths in order to circumvent parts that have been rendered unusable by defects or malfunctions. We cite just a few examples: Defect tolerance for VLSI yield improvement [1], computation-mapping onto an incomplete or damaged FPGA using alternate components and interconnects [2], and building gracefully-degrading array processors [3]. A number of architecture-level methods for allowing a system to adapt to changing conditions can also fall in this category, although some may not be readily recognized as reconfiguration. Note, for example, that replacing a failed functional unit with a spare one upon failure detection, is the same as restructuring an $(s + 1)$-unit "parallel" system by switching-out the previously-active module and switching-in one of the s spare modules [4].

In the just-cited references, and in the context of this work, the goal of reconfiguration is returning a processor array containing a number of malfunctioning nodes to the original intact configuration, in order to run the existing algorithms with no modifications. Stated differently, we will not consider the use of a damaged array, providing a degraded level of inter-node connectivity, owing to bandwidth limitation, path dilation, or resource reduction [5], which would require computations to be remapped by increasing processor loads and/or link communication traffic. Also, we won't deal with certain kinds of reconfiguration

that provide extended computational power for the array (in the sense of complexity-theory) that significantly speed up a number of global computations [6]. Given that the use of processor arrays can lead to high performance for many applications, numerous researchers have studied alternative reconfiguration schemes for such arrays [7]. Proposed architectures use various types/numbers of switches and place them differently within the array. Control schemes for effecting reconfiguration also differ (centralized, distributed, and hybrid), as do the complexities, and thus the speeds, of algorithms that decide on how to reconfigure the system.

Key contributions of this paper are as follows. First, we show, through simple models, how distributing the switching mechanisms among the processors can lead to reliability improvement by making switch malfunctions no more serious than processor malfunctions, that is, removing the system's hard core. Second, we introduce parametrized reliability models for both centralized switching architectures and distributed schemes, in a way that the sensitivity of the derived reliabilities with respect to chosen parameter values can be studied. Third, we demonstrate that the ability to derive tight reliability bounds is at least as important as envisaging ingenious reliability enhancement schemes. In other words, if reliability rises, but the reliability model does not reflect that rise, then we really do not have a guaranteed reliability improvement. Fourth, we show that the choice among multiple candidate systems with respect to highest reliability isn't a straightforward task, given uncertainties in reliability values reflected in loose or tight bounds.

2. RECONFIGURABLE 2D ARRAYS

We will use a small 5 × 5 array of processors, with one spare row and one spare column, to illustrate the advantages of distributed switching over a centralized scheme. This arrangement leads to a 6 × 6 host array. From specific results for this small array, we establish general results by demonstrating that the advantage grows when we expand the array-size or increase the redundant resources, that is, use more spare rows/columns. More work remains to be done on quantifying the added advantages with size and redundancy, which remain qualitative in this paper.

An example array is depicted in Figure 1, where tracks of embedded reconfiguration switches are placed between processor rows/columns. Four-port, 3-state switches appear as small circles in Figure 1. Figure 2 depicts the details and usage of these switches within the processor array of Figure 1. The actual method of reconfiguration and its capacity to tolerate malfunctions will be discussed in Sections 3 and 4

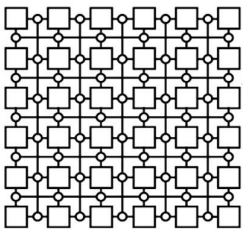


Figure 1. Two-dimensional 5 × 5 array, outfitted with embedded reconfiguration switches, a bottom-edge spare row and a right-edge spare column

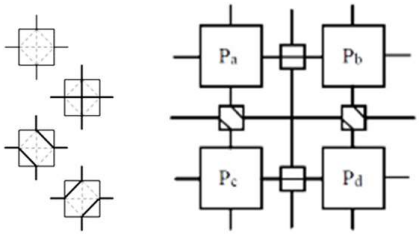


Figure 2. Three-state reconfiguration switches and their use in the array of Figure 1

Figure 1 represents only one way of how processor bypassing capability might be provided by reconfiguration switches for the purpose of diverting signals from their original destinations in the same row or column to alternate nodes, perhaps in other rows/columns. Upon the replacement of a node, either the new node taking over is initialized to the current state of the outgoing node to continue its work or else the entire computation is repeated from the beginning or from a previously-saved rollback point. Given that the complexity of initialization or restart is independent of the replacement and reconfiguration architecture, we will not discuss it further in this paper.

Some alternate processor-array reconfiguration schemes use more than one track of reconfiguration switches; others employ switches that are more-complex than the one depicted in Figure 2. Figure 3 depicts a particular reconfiguration scheme using two switch tracks, embedded between processor rows/columns. The use of more switches or ones with higher complexity presents a reliability tradeoff to the designer, which isn't easy to quantify in general. One reason is that potential benefits of greater flexibility in reconfiguration may be offset by the higher failure rates of more-numerous or more-complex switches.

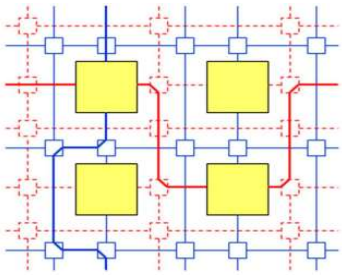


Figure 3. Two-dimensional array, outfitted with 3-state reconfiguration switches, lined up on two tracks in the gaps between processor rows/columns

In this paper, we limit our discussion to only one track of switches between array rows and columns. Furthermore, we will focus on identical switches, to simplify reliability models and VLSI realization. However, our switches can be arbitrarily complex, given that, in our reliability model, the failure rate of a switch is an independent parameter unrelated to those of other components. With one spare row and one spare column added to an $n \times n$ array, the redundancy factor for nodes is $(2n + 1)/n^2 = O(1/n)$. Increasing the number of spare rows and spare columns to k leads to the redundancy factor $(2kn + k^2)/n^2 = O(k/n)$. In either case, the redundancy factor is relatively low.

3. RECONFIGURATION WITH CENTRALIZED CONTROL

Consider embedding a 5 × 5 guest array within the 6 × 6 host array of Figure 1. In the beginning, the active nodes are within the 5 rows at the top and the 5 columns on the left. As nodes malfunction, the configuration undergoes changes. Figure 4 depicts an array after reconfiguration, with the assumption that, in addition to switches that re-route signals, the array is equipped with a separate mechanism for bypassing a node within the row/column where it is located.

Intuitively, re-routing of signals is accomplished by shifting rows downward (in the direction of the spare row) and shifting columns rightward (in the direction of the spare column), to compensate for unavailable row/column nodes. We offer no discussion of reconfiguration algorithms (see [8], for example). It suffices for our purposes in this paper to note that reconfiguration allows the guaranteed tolerance of up to two malfunctioning nodes. In other words, there exist certain patterns of three unavailable nodes that defeat this particular scheme's reconfiguration capability [4].

Two schemes can be envisaged for the operational control of the reconfiguration switches. In the first scheme, switches may be controlled by a centrally-located unit. Disadvantages of centralization include the control mechanism constituting a single point of failure, requirement for long wires to connect the control unit to all the switches, and excessive delay in uploading the configuration information upon a node replacement. In the second scheme, switches are paired with nearby nodes, which controls them. One drawback of distribution is that a malfunctioning node may not be able to properly control the switch(es) assigned to it, making other processing resources or certain array configurations inaccessible; this would reduce system reliability and increase the effort needed to model all pertinent scenarios.

Let's assess the reliability of the array in Figures 1 and 4, using a simple combinatorial model. We lump the switching mechanism into a hard core and consider the collection of nodes as a 34-out-of-36 system; recall that our scheme guarantees the tolerance of up to 2 bad nodes. Denoting the node failure rate by λ, the switch failure rate by σ, and noting that there are 60 embedded switches within the 6 × 6 array:

$$\text{Node reliability} = r = e^{-\lambda t} \quad (1)$$

$$\text{Switching reliability} = e^{-(60\sigma)t} \quad (2)$$

$$\text{Overall reliability} = e^{-(60\sigma)t}\, R_{34\text{out}-\text{of}-3}\ (r) \quad (3)$$

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

$$R_{34-out-of-36}(r) = r^{36} + 36r^{35}(1-r) + (36 \times \tfrac{35}{2})r^{34}(1-r)^2 = r^{34}[r^2 + 36r(1-r) + 630(1-r)^{2]} = r^{34}[595r^2 - 1224_r + 630] = r^{34}[1 + (1-r)(629 - 595r) \quad (4)$$

Substituting (4) into (3) and plotting the system reliability for different switch complexities, reflected in the ratio $\rho = \sigma/\lambda$, leads to Figure 5. For processors with typical complexities, ρ is likely to be near the lower end of the interval of considered values [0.001, 0.1], whereas for bare-bone PEs used in some arrays, ρ may approach, but likely not get very close to, the higher end.

Unreliability curves in Figure 5 have expected shapes. When λt is small, switching is the main source of unreliability, given the extremely high reliability of the processing part, as a 34-out-of-36 system. This observation explains the significant differences between reliabilities for the three cases with varying switch complexities. For large values of λt, multiple node malfunctions that exceed the system's tolerance limit dominate, making switching differences much less relevant.

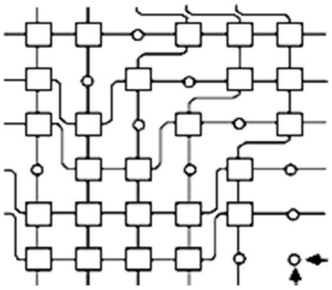


Figure 4. Salvaging a healthy 5 × 5 array from the 6 × 6 array of Figure 1, which is injured

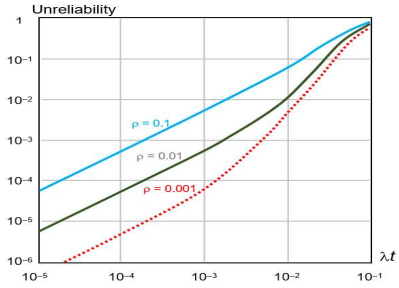


Figure 5. Reliability plots for a 2D reconfigurable array with centralized switching as a function of λt, with variations in the parameter $\rho = \sigma/\lambda$

4. RECONFIGURATION WITH DISTRIBUTED CONTROL

We now shift our focus to a switching scheme in which system configuration is dictated by good processors, through setting the states of their own internal switches. Malfunctioning processors become irrelevant via their outputs not being considered by good processors. The scheme works, as long as all good nodes know the identity of malfunctioning nodes. This isn't an added burden, because such information is needed even in the centralized scheme.

One can implement distributed switching in many different ways. For our comparisons to be fair, we choose a distributed scheme that offers the same reconfiguration capability as the centralized scheme of Figures 1 and 2. Let's consider modules having two built-in 3-input multiplexers (muxes) which allow them choose their east and north neighbors from among three candidate nodes. Consider, for example, that a processor's north neighbor can one of three nodes in the preceding row : The node directly above, plus those before and after it in the same row. This is the same as the capability provided by the scheme in Figures 1 and 2, assuming a single spare row and a single spare column.

In this scheme, each node, with its built-in switches, becomes a little bit more complex, which we model by increasing the node failure rate from λ to $\lambda + \alpha\sigma$ (σ is the failure rate of the original switches of Figure 2 and α is a new parameter reflecting the distribution overhead, that is, modeling the fact that switches become a bit more complex when we distribute them among the nodes). Reliability equations now become:

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

$$\text{Overall reliability} = R_{34-\text{out}-\text{of}-3}\ (r') \quad (6)$$

We can consider $\alpha = 2$ as a reasonably pessimistic value in our running numerical example: The centralized scheme of Figure 1 includes 60/36 ≅ 1.67 switches per node, each 3-state switch realizable with two 2-to-1 muxes. As depicted in Figure 6, two 3-input muxes are needed for each node in the distributed scheme. Figure 7 depicts the unreliabilities of our example array as $\rho = \sigma/\lambda$ assumes the values in {0.001, 0.01, 0.1}. Again, the results are as expected. Reconfiguration switches and processing nodes are merged in the distributed scheme, with successful reconfiguration requiring only that 34 of the 36 nodes be healthy. The proper functioning of the switching mechanism is no longer more critical than the health of processors. Put another way, we have eliminated the system's single point of failure.

One may worry about the impact of choosing $\alpha = 2$ on the validity of results. Table 1 shows the outcome of a sensitivity analysis on the value of α, by considering different values for the parameter that represent negligible overhead ($\alpha = 1$) and more significant overhead ($\alpha = 3$). These extreme values of α serve to confirm that reliability values show little change when adjusting the value of α.

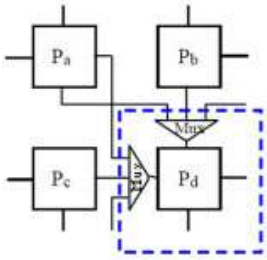


Figure 6. Use of modules with internal switching capabilities in the form of muxes

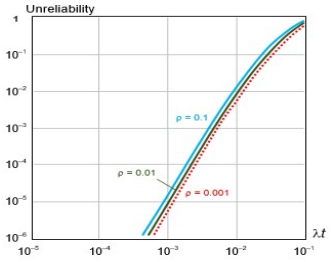


Figure 7. Reliability plots for a 2D reconfigurable array with distributed switching as a function of λt, with variations in the parameter $\rho = \sigma/\lambda$

Table 1. Demonstrating that reliability isn't sensitive to variations in α

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

5. COMPARATIVE RELIABILITY EVALUATION

Comparing Figures 5 and 7 reveal the advantages of distributed reconfiguration switching with respect to reliability. The advantages are greatest when the value of λt is small, that is, the expected range of operation for high-reliability systems. In order to highlight the reliability differences for a reasonable case of switch complexity ($\rho = 0.01$), we superimpose the unreliability curves of Figures 5 and 7 to create Figure 8. Please refer to Section 7 for an explanation of the dotted gray curve.

Figure 8 tells us that reliability differences disappear as λt approaches 1, making centralized and distributed switching indistinguishable with regard to reliability. Note, however, that the region near $\lambda t = 1$ is practically useless, given the extremely low reliability values. A natural question at this point is whether the results pertaining to a specific, fairly small array utilizing a specific switching scheme, carry more-general significance. The three claims, with informal justifications, that follow suggest that the results are indeed more general.

Claim 1: Increasing the size of the array, while keeping the same switching scheme and, thus, tolerance level, leads to a growth of the advantage of distributed switching over centralized switching.

Proof outline for Claim 1: Take the reliability-lower-bounds expressions for an $h \times h$ array, and consider the next-larger, $(h + 1) \times (h + 1)$, array. Prove that the reliability lower-bound deterioration for the larger array is less for distributed switching compared with centralized switching. Intuitively, the centralized switch complexity growth directly affects the reliability, while the increase in module complexity as a result of distribution is moderated by passing through the k-out-of-n filter.

Claim 2: Adding spare rows or spare columns causes the advantage of distributed switching over centralized switching to grow, all else being kept the same.

Proof outline for Claim 2: With 2 spare rows and 2 spare columns, the reliability function changes from $(n^2 + 2n - 1)$-out-of-$(n^2 + 2n + 1)$ to $(n^2 + 2n - 3)$-out-of-$(n^2 + 2n + 1)$, because of the tolerance level improving to 4 processor malfunctions. Here, too, the improvement in the k-out-on-n function value dominates the improvement in the centralized scheme, with its increased switch complexity.

Claim 3: Using more-complex switches to improve the malfunction tolerance level causes the advantage of distributed switching over centralized switching to grow.

Proof outline for Claim 3: The justification is quite similar to that of Claim 2. Other than the filtering effect of the k-out-of-n function, distributing the more-complex switching capabilities is beneficial.

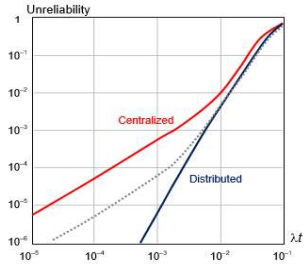


Figure 8. Comparative unreliability plots as a function of λt, assuming $\rho = 0.01$ and $\alpha = 2$

6. THE NOTIONS OF MODELABILITY AND RELIABILITY INVERSION

In reliability engineering, the exact reliability of a system is unknowable. If we could observe system failures by running hundreds of identical copies of a system for decades, we could derive the actual reliability with high confidence. We need a large number of copies and extended running times because failure events are extremely rare in practical, highly-reliable systems, so extensive data collection is needed for results to be statistically valid. A practical alternative is to use an analytic or simulation model, involving simple, pessimistic assumptions about subsystems and their interactions, to derive a reliability lower bound. The latter approach becomes mandatory for evaluating a system at design time, when a physical implementation isn't available. The need for experimentation still exists, even when using models. Parameters for such model are estimated, and models are corrected/tuned, with help from experimental observations.

The nature of modelability is similar to (design for) analyzability, or "design for analysis" [9], a notion along the same lines as design for manufacturing (manufacturability). For electronic circuits, design for packageability [10] is rather similar. Analyzability can be ensured by insisting that certain design constraints be honored so as to allow the use of simpler tools for analysis. The notions just listed constrain the design process, but, surprisingly, lead to cost reduction and faster development. Modelability makes it easier to derive tighter lower bounds for reliability. Figure 9 compares Systems A and B with respect to their (unknowable) reliabilities and the corresponding model-based lower bounds, revealing System B to be more modelable than System A. Even though modelability is a qualitative notion at this juncture, it can certainly be quantified with further work (which is in the planning stage), in the same way that testability, serviceability, and several other "ilities" were quantified, after starting out as qualitative notions.

To recap, obtaining guaranteed lower bounds on a system's survival probability requires that reliability analysis be based on worst-case assumptions. Reliability experts are aware of the desirability of tight lower-bounds, but sometimes system structure makes it difficult to derive tight bounds. Consider System B with (unknowable) actual reliability of 0.997 and a computed reliability lower bound of 0.995 for mission time t_0. Similarly, System A with (unknowable) actual reliability of 0.999 has a computed reliability lower bound of 0.993 for the same mission time. Figure 9 depicts the situation above. Unknowable actual reliabilities force us to make decisions based on lower bounds, leading to the pronouncement that B is more reliable than A at t_0. This condition is referred to as reliability inversion [11], in analogy to "priority inversion" in the scheduling of real-time tasks [12], which caused serious trouble during the 1997 Mars Pathfinder mission [13].

Paying greater attention to Figure 9, we see that each of the systems A and B is preferable to the other system for certain ranges of mission time. However, these nuances are hidden from us due to unknowable actual reliabilities. Computed reliability bounds proclaim System B as uniformly preferable to System A with regard to reliability. All of the results reported in this paper should be extensible to alternative reconfiguration architectures and algorithms proposed more recently [14] - [17].

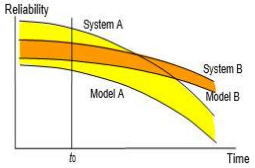


Figure 9. True, but unknowable, reliability vs. pessimistically-derived lower bound

7. CONCLUSION

An important conclusion of our work is that two properties tend to make processor arrays with distributed reconfiguration switching preferable to those with centralized switching: Inherently higher reliability and better modelability, the latter property allowing us to use simple combinatorial models to derive tight reliability lower bounds. The author has previously shown that, under certain unusual conditions, centralized switching can exhibit higher reliability (the dashed line in Figure 8), even though its pessimistically-derived reliability bound is lower. Future research must address several remaining problems, as pointed out in the preceding sections. First, Claims 1-3 of Section 5 show that certain changes in array configuration improve the system reliability. We shall aim to quantify these improvements, either by constructing expanded formal proofs or by deriving reliability formulas for different array configurations and redundancy factors. Second, we shall move in the direction of quantifying the modelability notion, thus making it usable for comparative assessment competing system designs or for producing design guidelines and methodologies. Third, we may look into extending our results to alternative reconfiguration architectures and algorithms proposed more recently, so as to produce results that enjoy broader applicability.

REFERENCES

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

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

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

[4] B. Parhami, "Behrooz Parhami's Textook on Dependable Computing," *Dependable Computing: A Multilevel Approach*, 2020. [Online] Available: https://www.ece.ucsb.edu/~parhami/text_dep_comp.htm.

[5] Yu-Chee Tseng, Ming-Hour Yang, Tong-Ying Juang, "Achieving fault-tolerant multicast in injured wormhole-routed tori and meshes based on Euler path construction," in *IEEE Transactions on Computers*, vol. 48, no. 11, pp. 1282-1296, Nov. 1999.

[6] Y. Ben-Asher, D. Peleg, R. Ramaswami, A. Schuster, "The power of reconfiguration," *Journal of Parallel and* Distributed Computing, Springer, vol. 13, no. 2, pp. 139-153, 1991.

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

[8] M. Fukushi and S. Horiguchi, "Reconfiguration algorithm for degradable processor arrays based on row and column rerouting," *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, DFT, 2004, pp. 496-504.

[9] R. Suri, M. Shimizu, "Design for analysis: A new strategy to improve the design process," *Research in Engineering Design*, vol. 1, pp. 105-120, 1989.

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

[11] B. Parhami, "Reliability Inversion: A Cautionary Tale," in *Computer*, vol. 53, no. 6, pp. 28-33, June 2020.

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

[13] G. Reeves, "What really happened on Mars," *The Risks Digest*, vol. 19, no. 54, pp. 1-7, 1998.

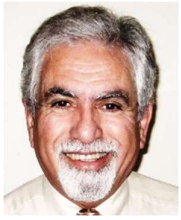
[14] G. Jiang, J. Wu, J. Sun, "Efficient reconfiguration algorithms for communication-aware three-dimensional processor arrays," *Parallel Computing*, vol. 39, no. 9, pp. 490-503, 2013.

[15] J. Qian, W. Cao, J. Hu, J. Zhang, Z. Xu, Z. Zhou, "Satisfiability-based method for reconfiguring power efficient VLSI array," *IEICE Electronics Express*, vol. 13, no. 23, pp. 20160930-20160930, 2016.

[16] J. Wu, N. Liu, S. Lam, G. Jiang, "Shortest Partial Path First Algorithm for Reconfigurable Processor Array with Faults," *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 1198-1203, 2016.

[17] W. Jigang, T. Srikanthan, G. Jiang, K. Wang, "Constructing Sub-Arrays with Short Interconnects from Degradable VLSI Arrays," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 929-938, April 2014.

BIOGRAPHY OF AUTHOR



Behrooz Parhami (PhD, University of California, Los Angeles, 1973) is Professor of Electrical and Computer Engineering, and former Associate Dean for Academic Personnel, College of Engineering, at University of California, Santa Barbara, where he teaches and does research in computer arithmetic, parallel processing, and dependable computing. A Life Fellow of IEEE, a Fellow of IET and British Computer Society, and recipient of several other awards (including a most-cited paper award from *Journal Parallel & Distributed Computing* and recognition as a Distinguished Visitor of IEEE Computer Society), he has written six textbooks and more than 300 peer-reviewed technical papers. Professionally, he serves on journal editorial boards and conference program committees and is also active in technical consulting.