Fault Resilient Physical Neural Networks on a Single Chip

Weidong Shi, Yuanfeng Wen, Ziyi Liu, Xi Zhao, Dainis Boumber, Ricardo Vilalta, Lei Xu
University of Houston Houston, TX 77004, U.S.A
{larryshi,wyf, ziyili}@cs.uh.edu, xzhao21@central.uh.edu, dboumber@uh.edu,
vilalta@cs.uh.edu, lxu13@uh.edu

ABSTRACT

Device scaling engineering is facing major challenges in producing reliable transistors for future electronic technologies. With shrinking device sizes, the total circuit sensitivity to both permanent and transient faults has increased significantly. Research for fault tolerant processors has primarily focused on the conventional processor architectures. Neural network computing has been employed to solve a wide range of problems. This paper presents a design and implementation of a physical neural network that is resilient to permanent hardware faults. To achieve scalability, it uses tiled neuron clusters where neuron outputs are efficiently forwarded to the target neurons using source based spanning tree routing. To achieve fault resilience in the face of increasing number of permanent hardware failures, the design proactively preserves neural network computing performance by selectively replicating performance critical neurons. Furthermore, the paper presents a spanning tree recovery solution that mitigates disruption to distribution of neuron outputs caused by failed neuron clusters. The proposed neuron cluster design is implemented in Verilog. We studied the fault resilience performance of the described design using a RBM neural network trained for classifying handwritten digit images. Results demonstrate that our approach can achieve improved fault resilience performance by replicating only 5% most important neurons.

1. INTRODUCTION

Device physics, fabrication, and process scaling engineering are facing major challenges in producing reliable transistors for future electronic technologies. With shrinking device sizes [21], the total circuit sensitivity to both permanent faults and transient faults is expected to increase (e.g., [16]). In the future, devices will likely fail due to permanent faults in silicon (e.g., [5, 6, 10]). These permanent faults include manufacturing faults that escape testing (e.g., [35]), faults that appear during the chip′s lifetime [49], or age-related device wear-out [40]. Multiple transient faults may also occur within a circuit′s operating time window [18]. These include single particle strikes that lead to double bit-flips or multiple particle strikes that are close enough temporally. Traditional fault tolerant techniques that work well under single transient fault may not scale when subjected to a multiple-fault environment. Research for fault tolerant computing devices has primarily focused on the conventional processor architectures. (e.g., [11, 30, 31, 37, 15, 51]). The studies include: hardware fault detection (e.g., [11, 30, 31, 37]), and fault recovery (e.g., [15, 51]).

Neural network computing has been employed to solve a wide range of applications such as, speech recognition (e.g., [32]), image processing and pattern recognition (e.g., [14]), compression and dimension reduction (e.g., [19]), hand tracking [27], autonomous robots (e.g., [45]), real-time embedded controller (e.g., [8]), cognition and behavior modeling (e.g., [46]).

Physical neural networks (e.g., [23, 39]) implement neural computing and the associated learning algorithms in hardware; it capitalizes on the inherent parallelism of neural computing. Among physical neural networks, neuromorphic networks refer to the hardware design or implementation that closely emulates the biological neural mechanisms (e.g., [22, 42, 2, 34]). For example, the recent DARPA SyNAPSE program is aimed to create physical neuromorphic machine technology that is scalable to biological levels, and that will pave the road for creating future intelligent machines.

A variety of technologies have been explored for implementing physical neural networks such as FPGA (e.g., [44, 41]), specialized analog and digital circuits (e.g., [3, 36, 4, 1, 7]), asynchronous circuits (e.g., [26, 38, 20]), systolic arrays (e.g., [9]), and nano-electronics (e.g., [52]).

Compared with the conventional processor architectures, physical neural networks offer the advantage of being fault tolerant. According to a previous analysis [43], acceptable fault tolerant performance in the conventional many-core processor architecture requires 100% faults covered. This leads to increasing design costs and hardware complexity. In contrast, when properly designed, *physical neural networks can continue to operate in the presence of both permanent hardware faults and transient faults.* In this paper, we propose new physical neural network approaches that are highly fault resilient. The main contributions of the paper are as follows:

- We describe a physical neural network architecture based on inter-connected tiles of neuron clusters integrated on a single-chip.
- We demonstrate the ability of efficient and *graceful performance degradation* (defined as the capability of gradually reducing performance with the increase of hardware faults).
- Our solution supports *performance recovery* from hardware damages to physical neural networks under controllable budgets of hardware resources.
- We propose an efficient approach to route neural computing outputs for a mesh of neuron clusters based on spanning trees and source based routing, and design a

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ESWEEK '14, October 12 - 17 2014, New Delhi, India
Copyright 2014 ACM 978-1-4503-3050-3/14/10$15.00.
http://dx.doi.org/10.1145/2656106.2656126.

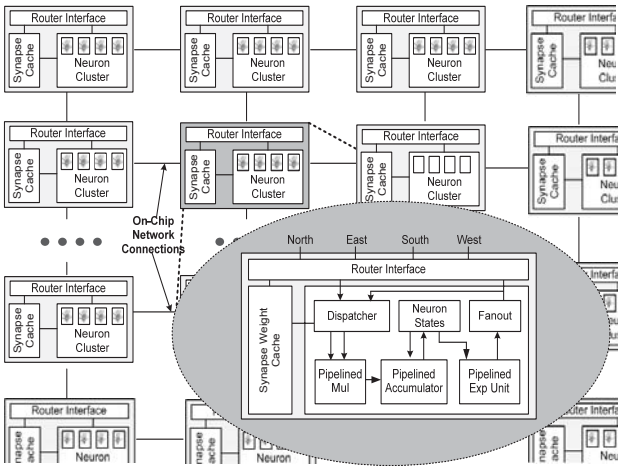


Figure 1: *Physical neuron clusters on a single chip: a mesh of physical neuron clusters with on-chip interconnects. Each cluster contains a routing interface for routing neural outputs sent from the neighbors. The functional units of each neuron cluster are pipelined. In addition, each neuron cluster contains a weight cache. Routing is spanning tree based using source neuron cluster identity. Each cluster contains build-in support to reconstruct new spanning trees when permanent faults occur to routing neuron clusters.*

solution to recover spanning tree routes in the face of permanent routing path faults.

2. DESIGN OF SCALABLE AND FAULT RESILIENT PHYSICAL NEURAL NETWORKS

2.1 Scalable Physical Neuron Networks

For a typical multi-layer neural network, during the "forward pass", an input pattern (e.g., image) is presented to the neurons of an input layer. For successive layers, the input to each neuron is the sum of the scalar products of the inputs with their respective weights: $x_j = b_j + \sum_{i=1}^{n} v_i * w_{ij}$, where w_{ij} is the weight connecting neuron j to neuron i, and v_i is the output of neuron i. b_j is a bias value. The output of a neuron j is calculated according to the equation $o_j = \frac{1}{1+exp(-x_j)}$.

The output is then sent to the neurons of the next layer. In restricted Boltzmann machine (RBM) [19, 28], in which stochastic, binary image pixels are connected to stochastic feature detectors using symmetrically weighted connections. The feature detectors correspond to hidden units.

To achieve scalability, we implement physical neural networks as tiles of neuron clusters integrated together on-chip (neuron cluster tiles on chip). Figure 1 shows a high level diagram of the proposed architecture. Each chip comprises a grid of inter-connected neuron clusters. Each neuron cluster connects to the four neighbors with bi-directional links. The four connections are named respectively, north, west, south, and east. Neuron clusters on the edges have three neighbors and the neuron clusters at the corners have only two neighbors. Each neuron cluster contains, a routing interface (connecting to the four neighboring clusters), a synapse weight cache (a directly mapped cache for storing connection weights), three pipelined functional units, a set of registers for storing neuron states, a scheduler that dispatches, for each neuron, the input summation to a pipelined exponential function after all the weighted inputs are summed together, and fanout logics for forwarding neuron outputs to the

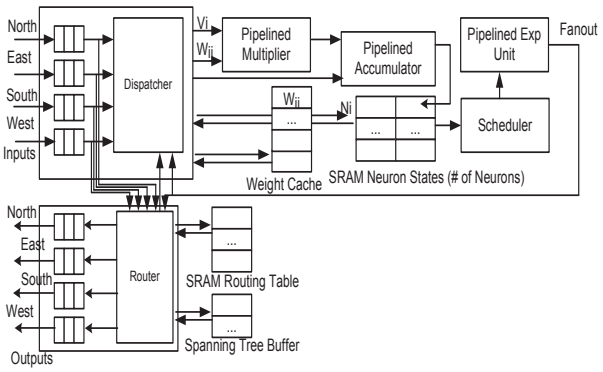


Figure 2: *Block diagram of a physical neuron cluster. Each cluster contains pipelined neural computing components, a weight cache, a routing unit (with routing table) and scheduling logics.*

target neurons who need them as inputs. For each neuron, the three functional units implement the neural computing equations that compute the neuron output based on the inputs. For each neuron, its output is routed by the routing interface to all the target neurons that receive the output as input. For both performance and efficiency reasons, the functional units are pipelined. They are, a pipelined multiplier, a pipelined accumulator, and a pipelined exponential evaluator that performs the last neural computing equation for each neuron based on the summed inputs.

Multiple neurons are allocated to each neuron cluster. Each neuron cluster is self-contained in the sense that for the n allocated neurons, it stores the synapse weights for all the incoming connections to the n neurons (in the synapse weight cache) and states of the n neurons (in the neuron state registers). In addition, the routing interface contains a routing table that stores all the necessary information for forwarding neuron outputs sent from a neighbor neuron cluster. Since all the needed weight values are stored locally inside each cluster, computing the output of each neuron based on the inputs does not incur any external memory traffic. All the computations involve only local data accesses to the synapse cache and the neuron state register bank. This significantly improves performance and efficiency over many prior efforts of designing physical neural networks where connection weights or neuron outputs are stored in external DRAM. The gap between ASIC processor performance and delay of DRAM memory systems create the memory wall problem. Our tiled neuron cluster design does not suffer from this problem because all the connection weights are cached on-chip and the neuron outputs are distributed to the target neurons using on-chip networks. Since all the weights are cached internally, there is an upper bound on the number of input connections that a neuron can have. Size of the synapse cache increases with the upper bound value. Under the same transistor count budget, increasing the upper bound will reduce the total number of neuron clusters that one can have on a single chip. This upper bound limit applies to the incoming connections only. There is no limit on the number of target neurons that a neuron can connect. Such a design favors neural networks with sparse connections. It is worth pointing out that biological neural networks have sparse connections. For example, the human brain has close to 10 billion neurons, but the number of synapse connections is about five thousand per neuron. Alternatively, synapse values can be stored in 3D stacked memory chips (e.g., [50, 33]) integrated with physical neural network circuit.

Figure 3 shows a high level implementation of the weight cache and how a neuron cluster retrieves the weights upon

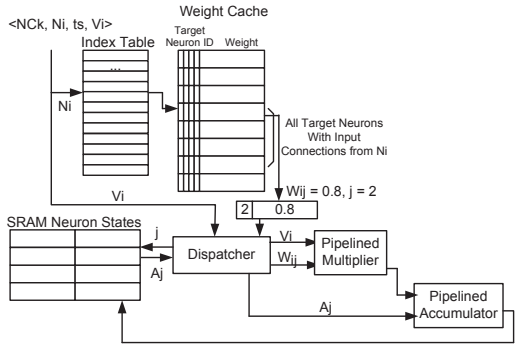


Figure 3: *Weight cache and weight lookup.*

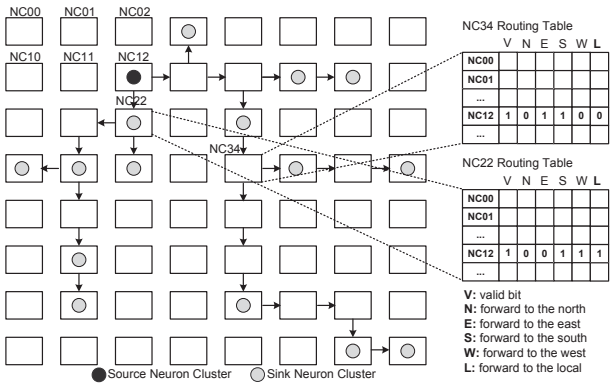


Figure 4: *Routing of neuron outputs using spanning tree. The source cluster is NC_{12}. Its outputs are routed to the leaf clusters (sink clusters) using a spanning tree.*

receipt of output V_i from neuron N_i. First, the routing table will determine if V_i should be forwarded to the local neurons. Then an index table will be used to find the weight cache base address for all the target neurons that receive input from N_i. After that, the weight values retrieved from the cache, in combination with V_i, and the previous accumulation results, all together are sent to the pipelined multiplier and accumulator.

Our design of tile based neuron clusters is scalable. There is no external data traffic when neuron outputs are computed. The neuron cluster itself contains a routing interface for forwarding neuron outputs to all the destination clusters. The neuron cluster can be replicated to form a 2D grid of N by N clusters. If each cluster can handle n neurons and there are N by N clusters on-chip, there will be N x N x n neurons on each chip. Two connecting neurons can be allocated to the same neuron cluster. However, the more likely scenario is that they are allocated to different neuron clusters. To support efficient routing of neuron outputs, each neuron cluster contains a routing interface and a routing table. Each cluster has bi-directional connections to all the immediate neighbors. Incoming data from the four directions (N, W, S, E), are temporarily stored in four input queues. If the received input is needed by one of the neurons, the value will be forwarded to a dispatcher and then sent to the functional units with the corresponding connection weight retrieved from the synapse cache. At the same time, the routing interface will determine where the input should be forwarded to by looking up a routing table. Using a single bit for each immediate neighbor, the routing table indicates whether and where an incoming data should be forwarded. The routing decision is based on the data source. Ideally, there can be one routing entry for each neuron. However, such design requires too many hardware resources. A more resource efficient solution is to have one routing entry per source neuron cluster. In this case, if a chip contains 10x10 neuron clusters, the routing table requires only total 600 bits (10x10 cluster x 6 routing bits).

In neural network computing, the output of a neuron is often needed as inputs by multiple neurons. A naive solution is to flood a neuron output to all the neuron clusters. Flooding neuron output will certainly slow down the performance. It will waste routing resources and create unnecessary energy consumption. An optimal solution is to send the neuron outputs to only these neuron clusters that need them. This can be achieved using spanning trees or multicast (e.g., [47, 53]). Figure 4 shows how the output from a neuron in cluster N_{12} can be routed to the destination clusters using a spanning tree. The number of routing messages can be measured by the number of spanning tree edges. The grid organization of neuron clusters can tolerate permanent cluster faults because for a data source, it provides multiple alternative paths to reach a destination neuron cluster, and therefore prevents a single point of failure in neuron output transmissions.

For spanning tree based routing using data source as index, once the spanning trees are configured and stored in the routing table of each cluster, permanent fault of a cluster can prevent all the downstream neuron clusters from receiving the inputs. To address this issue, we designed a spanning tree recovery approach for reconstructing new spanning trees when permanently failed neuron clusters are detected. The details are presented in section 2.4.

2.2 Tolerating Hardware Faults

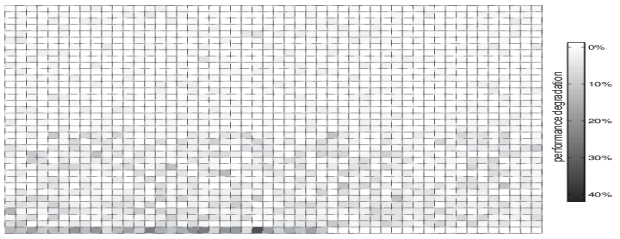
To investigate the capacity of the physical neural networks to tolerate hardware faults, we conducted detailed studies of how hardware faults affect pattern recognition performance of physical neural networks. A fault resilient example using the restricted Bolzmann machine (RBM) is studied. The RBM was tested on some of the recent neural computing hardware (e.g., [38]) designed using asynchronous circuits. Our example is a multi-layer image recognition RBMs following the description of [17]. The training set comprises 60,000 images of handwritten digits and the test set comprises 10,000 separate images [29]. The last layer of neurons comprises 10 perceptron nodes that classify the hidden layer features into the ten digits. During evaluation, transient faults were induced to the physical neurons or transmissions of the fanout signals. Recognition performance measured in terms of percentage of correctly classified digital images was recoded. As shown in Figure 5(a), performance gradually degrades when the number of physical neuron transient faults increases. For the purpose of testing its limit, we subjected the neural network model to a range of failure rates including extreme high failure rates unlikely to occur in the real world. For example, the system can retain 84% accuracy even under a per neuron transient failure rate of 5% (on average five erroneous outputs every 100 rounds of computing for each physical neuron). For tolerating transmission faults, the system has no performance degradation when the rate of transient transmission faults is less than 3%. Studies of permanent neuron faults indicate that, *not every neuron contributes equally to the overall performance.* Permanent faults to certain neurons are more critical than faults in other neurons. As illustrated in Figure 5(b), different amount of performance degradation is observed when permanent failure is introduced to each of the feature neurons.

2.3 Neuron Allocation and Neuron Cluster Map

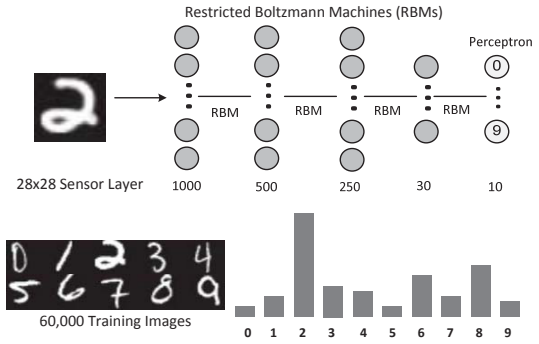
Given a neural network model, there exist multiple ways to allocate the neurons to a set of neuron clusters. As shown

in Figure 6, a simple way is to allocate neurons to a set of neuron clusters sequentially. However, sometimes, many critical neurons could end up in the same neuron cluster, which will likely amplify the negative consequence when the cluster is permanently damaged. To minimize the risk, one can randomly allocate neurons to the clusters, see Figure 6c. Or a third approach is to evenly distribute neurons with similar importance to all the neuron clusters. This can be achieved by first sorting all the neurons based on their importance. Then, the sorted neurons are allocated to the clusters sequentially.

(a) Performance of tolerating transient faults.



(b) Performance of tolerating permanent failures of physical neurons.



(c) Multi-Layer RBMs for recognizing images of handwritten digits.

Figure 5: *Illustration of fault tolerance for physical neural networks using restrictive Bolzmann machine as an example: The study uses a multi-layer RBM trained for handwritten digit recognition (60,000 training images and 10,000 test images). Transient faults were induced to the physical neurons and transmissions of the fanout signals. Recognition performance was measured. In (b), performance effects of permanent faults to the physical neurons were tested. Each box represents a neuron. The gray level corresponds to the impact on the overall image classification accuracy when a neuron fails. Darker means worse performance.*

After completing the allocation of neurons to a set of neuron clusters, the next step is to map neuron clusters to the physical neuron tiles, see Figure 7. Similar to neuron allocation, a simple approach is to map each neuron cluster to a physical neuron tile according to a pre-determined pattern such as row by row with sequential order inside each row, or

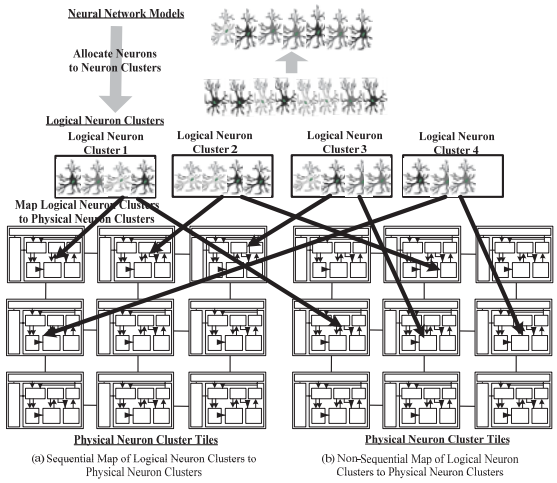


Figure 7: *Different approaches of mapping logical neuron clusters to physical neuron clusters: (a) map logical neuron clusters to physical neuron tiles according to a predetermined pattern; (b) randomly map logical neuron clusters to physical neuron tiles.*

column by column. Different mapping schemes could affect the distribution of routing workloads among the physical neuron tiles because each neuron tile also acts as a router for exchanging neuron outputs. Certain physical neuron tiles may become routing hotspots under certain neuron cluster mapping schemes. To avoid routing hotspots, one can randomly map logical neuron clusters to the physical neuron tiles. For certain neural networks, random mapping can more evenly distribute the routing demands to the physical tiles. Different mapping schemes may also affect the average length of routing spanning trees for a given neural network.

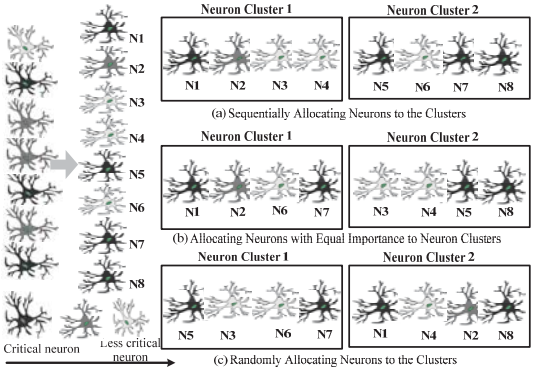


Figure 6: *Different schemes of allocating neurons to neuron clusters: (a) sequential allocation of neurons to neuron clusters; (b) priority balanced allocation of neurons to neuron clusters, each cluster with roughly equal number of critical neurons; (c) random allocation of neurons to neuron clusters. Neurons with darker color are more important than neurons with a lighter color.*

2.4 Fault Resilient Neuron Replication

One of our objectives is to develop low overhead solutions to preserve neural network performance when facing permanent hardware faults. One naive approach is to replicate all the neurons and allocate them to different physical neuron clusters. When one neuron cluster fails, the neuron replicas in the still operational clusters will continue to provide the

needed neural computing outputs. Although being fault tolerant, such a solution is less preferred because it incurs too much hardware overhead in both area and energy consumption. Leveraging the observation that failures of individual neurons have drastically different impacts on the overall neural network performance, we designed a more efficient solution that selectively replicates only these most critical neurons that, when missing, would lead to significant performance degradation. The importance of neurons can be evaluated quantitatively by measuring changes of the overall neural network performance. *To preserve performance* in the face of hardware faults, our approach only selectively replicates a small percentage of neurons that are performance critical. The exact percentage can be determined based on the amount of available hardware resources. Using such a solution, neural network performance is efficiently preserved to the maximum with the smallest resource overhead possible.

For detecting permanent faults in a neuron cluster, one simple approach is to ensure that, for each neuron cluster, at least one neuron is replicated. Outputs from two identical neurons can be compared. Any mismatch of the neural outputs can be detected and used to identify the failing neuron cluster. Here we use an example to demonstrate how a failing cluster can be detected. Assume that there are three neuron clusters, NC_i, NC_j, and NC_k. Cluster NC_i contains neuron N_x and N_y; cluster NC_j contains neuron N_x' (replication of N_x), and cluster NC_k contains neuron N_y' (replication of N_y). If a permanent fault happens to cluster NC_j (e.g., failed functional units), outputs of N_x and N_x' will mismatch. If outputs of N_y' and N_y match, then the system can infer that cluster NC_j is the source of failure. When a faulty neuron cluster is identified, the information can be reported to all the neuron clusters by sending a broadcast message. To tolerate permanent faults of neuron clusters, another alternative is to replicate the critical neurons twice. This way, each critical neuron is allocated to three clusters. By comparing the three outputs, a destination neuron cluster can choose the output shared by at least two clusters.

For further optimization, we propose an additional technique called, *neuron gating*, to support low power and efficient neural computing without significantly sacrificing performance. The idea is to selectively clock gate the neurons that are less performance critical. By only enabling these critical physical neurons, one can achieve an optimal balance between performance preservation, speed, and energy consumption.

2.5 Recovery of Fanout Routing Paths

Routing of neuron outputs is affected by permanent faults to neuron clusters. To restore routing paths for distributing neuron outputs, we designed a runtime approach that re-computes spanning trees after failed neuron clusters are detected. According to our approach, when a permanent fault of a neuron cluster is detected, its neighbor will send a report message upstream. The message will be forwarded along the spanning tree path and reach the source neuron cluster. Upon receipt of the report message, the source neuron cluster will initiate the process to compute a new spanning tree. The spanning tree re-compute algorithms described in pseudo codes are shown in list Algorithm I. A source neuron cluster NC_i will broadcast a probe message <probe, NC_i, ts> where ts is a time stamp. A neuron cluster increments the time stamp ts for each round of spanning tree computation.

To support spanning tree computation, each neuron cluster contains a spanning tree buffer with a small number of entries to keep track of the probe message source and responses from the neighbors. For fast access, the spanning tree buffer is implemented using content addressable memory (CAM) with source neuron cluster id as the index. In our design, the default number of entries is four. Figure 8 shows structure of the spanning tree buffer. When a neuron cluster NC_j receives a probe message initiated by cluster NC_i, it will find an available entry in the spanning tree buffer and set the valid bit. If neurons of NC_j receive outputs from cluster NC_i, NC_j will send a sink message back to the neighbor who forwards the probe message. The sink message is used to indicate that either NC_j or its downstream clusters contain neurons who have input connections from the neurons of cluster NC_i. Cluster NC_j records the neighbor that sends/forwards the probe message in the spanning tree field, probe source. Then NC_j forwards the probe message to all its operational neighbors (excluding the original neighbor who forwards the probe message and permanently failed neighbors), and records the number of probe messages sent.

Algorithm 1 Pseudocodes for recomputing spanning trees in response to permanent faults in routing: part I

```
1: Upon receipt of a probe message <probe, NC_i, ts> issued
   by NC_i from sender NC_k in direction D by NC_j;
2: Invalidate NC_i's routing paths in RoutingTable[NC_i]
3: Using NC_i as index and find if there is a matching entry p in the
   SpanningTreeCAM;
4: if p == NULL then
5:    p = Find an available spanning tree entry;
6:    if p == NULL then
7:       Find one occupied entry NC_m where (row(NC_m)
         > row(NC_i)) or (row(NC_m)==row(NC_i) and
         col(NC_m)>col(NC_i));
8:       if p!= NULL then
9:          Send a cancel message <cancel, NC_m> to direction X
            where p.probe_source[x]==1;
10:      end if
11:   end if
12:   if p!= NULL then
13:      p.content = NC_i; p.ts = ts; p.valid = true;
14:      Send <probe, NC_i, ts> to all the operational neighbors
         and record p.number_of_probes_sent;
15:      p.probe_source[D] = 1;
16:      if NC_j is a sink then
17:         p.sink = true;
18:         Send a sink message <sink, NC_i > to NC_k;
19:         p.responded = true;
20:      end if
21:   else
22:      Send a cancel message <Cancel, NC_i > to NC_k;
23:   end if
24: else
25:   Send a drop message <Drop, NC_i > to the sender NC_k;
26: end if

27: Upon receipt of a drop message <drop, NC_i > by NC_j
    from direction D;
28: Using NC_i as index and find if there is a matching entry p in the
    SpanningTreeCAM;
29: if p == NULL or p.valid == false then
30:    return;
31: end if
32: Set p.drop_neighbors[D] = 1;
33: if Sum(p.drop_neighbors) == p.number_of_probes_sent then
34:    if p.sink == false then
35:       Send a drop message <drop NC_i > along direction X where
          p.probe_source[X] == 1;
36:    end if
37:    p.valid = false;
38: end if
```

Upon receipt of a probe message, neighbors of cluster NC_j will execute the same algorithms. If a duplicate probe message is received by a neighbor NC_k, NC_k will send a drop message back to its forwarder. A neuron cluster keeps track of the number of drop message responses in the spanning tree field, named drop neighbors. If all the neighbors of NC_j send back drop messages as responses to the probe messages and NC_j itself does not contain neurons who need the outputs, NC_j will send a drop message back to its upstream neighbor who forwards the probe message to it. If a neuron cluster

NC_j receives one or multiple sink messages as responses to the forwarded probe messages and NC_j itself has not sent a sink message to its upstream probe message forwarder, NC_j will send a sink message back to the upstream neighbor who forwards/sends the original probe message.

Figure 8 shows status of the spanning tree buffers with an example. In the example, cluster NC_{12} has permanent fault. Neuron cluster NC_{11} initiates the spanning tree reconstruction process. Cluster NC_{12} and NC_{23} have input connections from the neurons contained by cluster NC_{11}. The old spanning tree for reaching NC_{23} with NC_{12} as a routing cluster is broken. Figure 8 also illustrates the messages exchanged for computing a new spanning tree. Meanwhile, it highlights the resulting spanning tree (in purple color). The computed spanning tree is recorded in each neuron cluster's routing table.

Algorithm 2 Pseudocodes for recomputing spanning trees in response to permanent faults in routing: part II

1: **Upon receipt of a sink message <sink, NC_i > by NC_j from direction D;**
2: Using NC_i as index and find if there is a matching entry p in the SpanningTreeCAM;
3: **if** p == NULL or p.valid == false **then**
4: return;
5: **end if**
6: **if** p.responded == false and NC_j!=NC_i **then**
7: Send a sink message along direction X where p.probe_source[X] == 1;
8: p.responded = true;
9: **end if**
10: Set RoutingTable[NC_i][D] = 1;

11: **Upon receipt of a cancel message <cancel, NC_i >**
12: Using NC_i as index and find if there is a matching entry p in the SpanningTreeCAM;
13: **if** p == NULL or p.valid == false **then**
14: return;
15: **end if**
16: **if** NC_j!=NC_i **then**
17: p.valid = false;
18: Send a cancel message along direction X where p.probe_source[X] == 1;
19: **else**
20: Recompute of a spanning tree failed. Wait for time period t and resend probe messages <probe, NC_i, ts+1> to the operational neighbors;
21: **end if**

Since the spanning tree buffer of each neuron cluster contains only a small number of entries, it is possible that a cluster may receive more probe messages than it can process. When this happens, to avoid potential deadlocks in sending probe messages, we use a tie breaking solution that assigns probe precedence to the neuron clusters. For instance, neuron clusters that have smaller row or column numbers have higher precedence than the neuron clusters that have larger row and/or column numbers. This means that when a cluster NC_j receives a probe message initiated by NC_i and its spanning tree buffer is fully occupied, it will compare NC_i with these already stored in the spanning tree buffer. If NC_i has less priority than all of them, NC_j will send a cancel message <Cancel, NC_i, ts> back. The cancel message will be forwarded upstream all the way to cluster NC_i. Upon receipt of the cancel message, cluster NC_i will abort the current round of spanning tree computation, wait for a while, and restart a new round of spanning tree calculation. If NC_i has higher priority than one of the source clusters that occupy NC_j's spanning tree buffer, NC_j will evict one of the source clusters that have less priority, and allocate the available spanning tree entry to NC_i. Accordingly, a cancel message will be sent to the neighbor that forwards the probe message of the evicted source cluster. To improve efficiency, a neuron cluster can wait before it initiates a round of spanning tree calculation. For example, a low priority cluster can wait until the clusters with higher priorities complete first and then starts its spanning tree computation.

2.6 Fault Resilience of the Recovery Mechanism Itself

Our fault model assumes that transient or permanent failures may occur at each physical neuron tiles. When a physical neuron tile fails, it may stop functioning or emit erroneous neural computing outcomes. Firstly, our fault resilient physical neural network design can tolerate neuron tile failures by replicating critical neurons and allocating them to different neuron tiles. Secondly, our fault detection and recovery mechanisms are completely distributed. Both fault detection and spanning tree recovery are integrated with the physical neuron tiles. Failed physical neuron tiles cannot prevent operational physical neuron tiles from performing their functions.

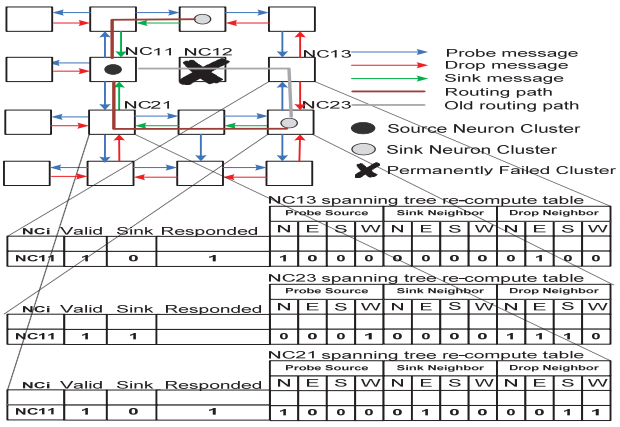


Figure 8: *Illustration of spanning tree recompute. When a routing neuron cluster has permanent fault, an affected source neuron cluster whose outputs are routed by the faulty neuron cluster can initiate a process to reconstruct a new spanning tree that can bypass the faulty neuron cluster. In the example, cluster NC_{12} has a permanent fault. Cluster NC_{11} is the source cluster. The figure shows some of the messages exchanged among the neuron clusters and the progress of spanning tree reconstruction.*

3. IMPLEMENTATION

As discussed in the previous sections, for each physical neuron cluster, the main logic and memory components include, pipelined functional units, which further include a single precision floating point accumulator, a multiplexer and an exponential evaluator; a synapse weight cache, which stores connection weights; a set of registers for storing neuron states; a SRAM routing table; input/output queues for the routing module; a fully associative spanning tree buffer for runtime spanning tree re-computation; various logic components for routing, scheduling, neuron output fanout, and spanning tree message handling. For rapid prototyping, we leveraged open source IP blocks whenever it is possible, and implemented the neuron cluster design in Verilog. We evaluate the power and area performance of our design by integrating these components and synthesizing the design. Verilog implementation of the designed physical neural networks is synthesized using the Synopsys Design Compiler with FreePDK at 45nm. It provides parameters for estimating overhead and tuning an architecture simulator that models the designed physical neural networks.

The floating pointer accumulator is derived from an open-source single precision, IEEE-754 compliant, signed adder.

We implemented a 6-stage pipelined accumulator. The design is fully synthesizable, occupying around 400 configurable logic blocks when tested using FPGA. The single precision floating pointer multiplier uses a 4-stage pipelined design. Compared to the single precision accumulator and multiplexer, the exponential unit is more complex (e.g., [12]).

The largest components per cluster are basically caches and SRAMs. Their implementation is straightforward. For the synapse weight cache, there are a maximum of 256 weights for each neuron. The weight value itself is single precision floating point based. The neuron states are stored in 16 registers, which are comparatively small. The routing table, which stores the forwarding information, one bit per direction (south/north/east/west/itself) per cluster. If the chip has 16x16 neuron clusters, the routing table has 256 entries as direct mapped SRAM buffer indexed by the neuron cluster id. The spanning tree buffer has four entries and is fully associative. Each entry contains 8-bit index and 32-bit data. There are eight queues (four inputs and four outputs), each with four packet buffers. The rest of the components are logics for routing, scheduling, neuron output fanout, and spanning tree recovery. Between two neighboring clusters, the bidirectional bandwidth is 200Mbits/ps.

Based on the synthesis results using Synopsys design compiler, a fully synthesizable implementation of our design at 45nm, occupies 84.29mm^2, and dissipates 15.5W of peak power. The largest area and power overhead come from the various SRAM based components. As a reference, Intel Core i5 Lynnfield fabricated in 45nm has die size of 296 mm^2.

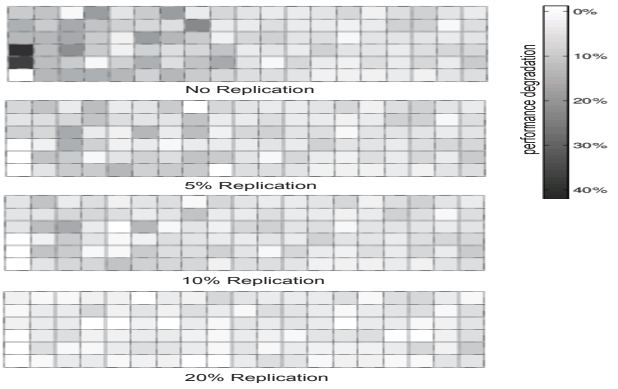


Figure 9: *Fault resilience by selectively replicating the critical neurons. The figures show decrease of image classification accuracy under different physical neuron replication settings: no replication (0%), replicating the most critical 5% neurons, replicating the most critical 10% neurons, replicating the most critical 20% neurons. Importance of a neuron is measured based on its impact on the overall classification accuracy using the 10,000 test images. In the figure, each box corresponds to a neuron cluster. The color of the box shows amount of performance decrease in image classification when the corresponding neuron cluster has permanent fault. Darker color means worse performance. As indicated by the results, replicating small percentage of the most critical neurons can significantly reduce the negative influence of neuron cluster with permanent fault on the classification performance.*

4. EVALUATION

To study the fault resilience performance of our design, we used the same RBM neural network and handwritten digit recognition task described in Section 2.2. The training image set comprises 60,000 images of handwritten digits and the test image set comprises 10,000 separate images [29]. The multi-layer RBM configuration and training procedure are based on the description of [17]. After training, the weights are pruned to satisfy the bound on the number of input connections per neuron by removing non significant weights (weights close to zero). Weight pruning is a common practice in neural computing for reducing computation workload (e.g., [48]). An architecture simulator is used to simulate the grid of neuron clusters. The neurons are mapped to the clusters. After training, the weights and connection settings are stored in the cluster. Routing paths for neuron outputs are pre-computed.

We then evaluate the influence of different options of allocating logical neuron clusters to the neuron clusters on the neural network performance. When assigning logical neuron clusters to physical neuron clusters, there are two approaches, sequential mapping (assigning logical neuron clusters to the physical clusters in linear order) and random mapping (randomly assigning logical neuron clusters to the physical clusters). Furthermore, the number of neurons allocated to each cluster can vary. For example, one can assign 8 neurons, 16 neurons, 24 neurons, or 32 neurons per cluster. As demonstrated in Figure 10, the setting of 16 neurons per cluster attains the best performance. Reducing the number of neurons per cluster will increase concurrency level with a price of decreased neuron cluster utilization. As the number of neurons allocated to each cluster decreases, the number of required clusters increases and routing becomes more complex. As a result, performance becomes communication dominated instead of computing dominated. On the other side of the spectrum, when more neurons are allocated to a cluster (e.g., 32 neurons per cluster), the cluster utilization increases, but the overall speed decreases due to reduced level of concurrency. For our pipelined neuron cluster, 16 neurons per cluster delivers the best speed performance.

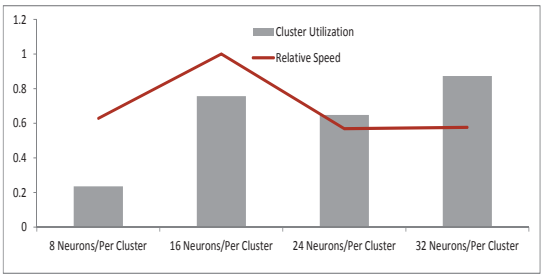


Figure 10: *Relative speed of physical neural network versus number of neurons per cluster. Speed is normalized against a default setting of sixteen neurons per cluster. Performance of physical neural network is communication bound when the number of neurons per cluster is small (low cluster utilization). Allocating large number of neurons per cluster increases cluster utilization with reduced concurrency level. For our pipelined neuron cluster, sixteen neurons per cluster achieves the best balance between speed and resource utilization.*

To evaluate the effects of selective neuron replication, we experimented with 5%, 10%, and 20% replication settings where 5%, 10%, and 20% neurons were replicated according to ranked importance. We rank the importance level of a neuron as follows. After a neural network is trained, during test phase, we induce fault to each neuron and measure the change in neural network performance (e.g., image classification accuracy in our tested RBM network). The procedure is repeated for all the hidden or feature detection neurons. Figure 9 shows the performance impact of failing clusters to the overall handwritten digit recognition performance (measured as the amount of performance decrease in correctly recognizing the test digit images). Each box rep-

resents one cluster. The gray color level is proportional to the performance change. Darker color means higher level of negative impact on the overall recognition performance. As shown by the results, without neuron replication, permanent faults to the clusters that contain critical neurons can cause significant decrease in performance. Brighter color means less change in the overall performance. With only 5% of the most important neurons replicated, the neural network performance can be effectively preserved.

We designed tools that, given a neural network model, allocate neurons to the neuron clusters and compute the routing tables. For each neuron cluster, our developed tool computes a spanning tree that will route its outputs to all the neuron cluster recipients (sink clusters). Each neuron cluster can be involved in routing outputs sent from multiple source neuron clusters. For each routing neuron cluster, one can count the number of source neuron clusters, it will forward their outputs to the downstream clusters. Similarly, one can also count the number of downstream clusters that will receive data from a source neuron cluster through a routing cluster. Figure 11 illustrates the upstream and downstream statistics of all the routing neuron clusters for the studied image classification neural network. When a neural cluster fails permanently, all the affected downstream clusters will stop receiving inputs from the affected source clusters.

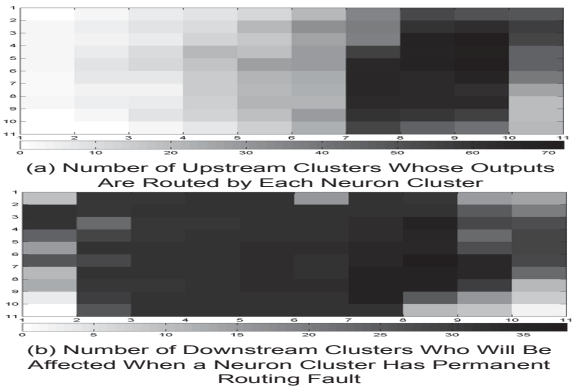


Figure 11: *Impact of permanent routing faults caused by failed neuron clusters. For each routing neuron cluster, the number of upstream clusters counts the number of unique upstream clusters whose outputs it will forward to a downstream cluster. The number of downstream impact counts the number of downstream clusters who receive inputs from a upstream cluster through a routing neuron cluster (each source - sink cluster pair is counted once). The gray level is proportional to magnitude of the number. The data was collected from sequential mapping of neuron clusters.*

The neuron cluster mapping approach shown in Figure 11 has one drawback that certain neuron clusters have higher routing workload than the others, as reflected in the uneven distribution of upstream routing statistics. An improved solution is to randomly map logical neuron clusters to the physical neuron clusters. The results are shown in Figure 12. When randomized, the standard deviation of upstream counts for all the routing clusters reduces from 22.87 to 16.4. The means that routing workload is more evenly distributed across the neuron clusters when logical neuron clusters are randomly mapped.

Another advantage is that with randomization, both the total and average spanning tree lengths for all the neuron clusters are also reduced. When randomly mapped, the average spanning tree height for all the spanning trees decreases from 8.95 to 7.1.

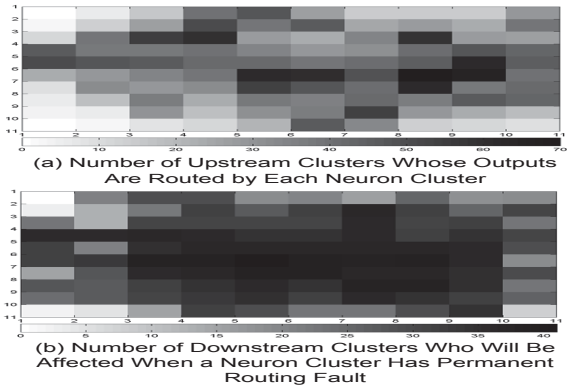


Figure 12: *Impact of permanent routing faults caused by failed neuron clusters. For each routing neuron cluster, the number of upstream clusters counts the number of unique upstream clusters whose outputs it will forward to a downstream cluster. The number of downstream impact counts the number of downstream clusters who receive inputs from a upstream cluster through a routing neuron cluster (each source - sink cluster pair is counted once). The gray level is proportional to magnitude of the number. The data was collected from random mapping of neuron clusters.*

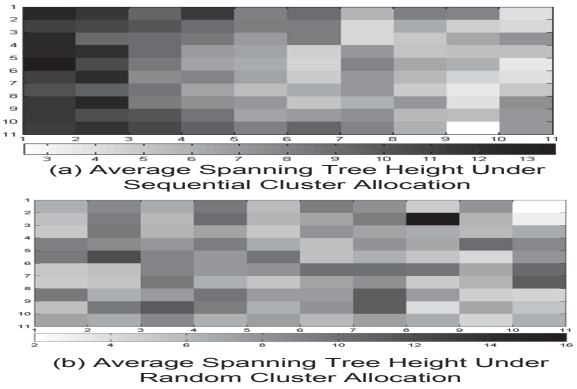


Figure 13: *Average spanning tree heights for all the source neuron clusters. Results on the left side show average spanning tree heights under sequential mapping of neuron clusters. Results on the right side show average spanning tree heights under random mapping of neuron clusters. For a source neuron cluster, average spanning tree height is calculated as, dividing the total routing path length for all the leaf clusters who receive input from the source neuron cluster by the number of leaf clusters. The gray level is proportional to the spanning tree length. Random cluster mapping reduces the total routing path for all the spanning trees by 25.3%.*

5. RELATED WORK

There have been continuous efforts in realizing physical neural networks (e.g., [23]). Physical neural networks implement neural computing and the associated learning algorithms in hardware. Recently, physical neuromorphic networks that closely emulate the biological neural mechanisms (e.g., [22, 42]) have also attracted significant research attention. For example, neuromorphic networks implemented using FPGA [44] or general purpose processors [24] have been described in the literature. In the past, a variety of hardware technologies have been explored for implementing physical neural networks including FPGA (e.g., [44]), specialized analog and digital circuits (e.g., [3, 36, 4, 1, 7]), asynchronous circuits (e.g., [26, 38]), systolic arrays (e.g., [9]), and nano-electronics (e.g., [52]). For efficient on-chip

communication of neuron outputs, multi-cast based routing (e.g., [47, 53]) has been proposed to reduce the traffic required to route neuron outputs to the destinations. Another set of related work can be found in the area of fault tolerant network-on-chip (e.g., [25, 13]).

Our paper distinguishes from the prior related work mainly in these aspects. Firstly, one of the main focuses of our work is to develop new physical neural network approaches that can attain fault resilience and pro-actively preserve the performance of neural computing in the face of permanent hardware faults and failures (an imminent problem faced by device engineers and ASIC designers). We propose new techniques such as selective replication of critical neurons to tolerate permanent hardware faults in physical neural networks. Since our solution only replicates the most critical neurons, it incurs minimal resource overhead. Secondly, we introduce a routing recovery approach that recomputes spanning trees for distributing neuron outputs after permanent faults occur to on-chip routing nodes. The existing multi-cast based approaches for neuromorphic computing doesn't address routing path reliability and recovery under permanent hardware faults. Furthermore, our neural output spanning tree routing is based on neuron cluster source addresses (source routing) instead of destination addresses. Although research in fault tolerant network-on-chip addresses routing paths recovery when fault occurs, these techniques primarily deal with on-chip networks exhibiting different behaviors and characteristics from a multi-layer physical neural network. Furthermore, those approaches don't address nor provide solutions for physical neural network fault resilience in general. Our system provides a comprehensive and holistic scheme for attaining physical neural network fault resilience with a collection of fault tolerant and recovery techniques (e.g., selective neuron replication, importance based neuron allocation) specifically considering the nature of neural computing. In addition, different from the prior work, our design mitigates the performance bottleneck of memory wall by eliminating the external data access traffic. Neuron outputs are routed to all the target neurons using efficient source based spanning tree routing and mesh based on-chip networks. Synapse weights are cached on-chip. Consequently, the design improves both reliability and performance.

6. CONCLUSION

This paper introduces a scalable and fault resilient physical neural network design. To achieve scalability, our design uses tiled neuron clusters where neuron outputs are efficiently forwarded to the target neurons using source based spanning tree routing. The design solves the memory wall problem by eliminating the off-chip data traffic with per cluster synapse caches. For fault resilience when facing permanent hardware failures, the design pro-actively preserves neural network computing performance by selectively replicating performance critical neurons. Furthermore, the paper presents a spanning tree recovery solution that mitigates disruption to distribution of neuron outputs caused by failed neuron clusters. The proposed neuron cluster design is implemented in Verilog. We study the fault resilience performance of the described design using a RBM neural network trained for classifying handwritten digit images. Results demonstrate that our approach can achieve improved fault resilience performance with minimal cost on hardware resources by selectively replicating only 5% most important neurons.

References

[1] A. Achyuthan and M. Elmasry. Mixed analog/digital hardware synthesis of artificial neural networks. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(9):1073 –1087, sep 1994.

[2] D. Balduzzi and G. Tononi. What can neurons do for their brain? communicate selectivity with bursts. *Theory in Biosciences*, page 17 pages, 2012.

[3] A. Bermak and D. Martinez. Digital VLSI implementation of a multi-precision neural network classifier. In *6th International Conference on Neural Information Processing*, volume 2, pages 560–565, 1999.

[4] A. Bermak and D. Martinez. A compact 3D VLSI classifier using bagging threshold network ensembles. *Neural Networks, IEEE Transactions on*, 14(5):1097 – 1109, sept. 2003.

[5] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, Nov. 2005.

[6] M. A. Breuer, S. K. Gupta, and T. M. Mak. Defect and error tolerance in the presence of massive numbers of defects. *IEEE Des. Test*, 21(3):216–227, May 2004.

[7] B. E. Brown, X. Yu, and S. L. Garverick. Mixed-mode analog VLSI continuous-time recurrent neural network. In M. H. Rashid, editor, *Circuits, Signals, and Systems*, pages 398–403. IASTED/ACTA Press.

[8] M. Chiaberge and L. M. Reyneri. Cintia: A neuro-fuzzy real-time controller for low-power embedded systems. *IEEE Micro*, 15(3):40–47, June 1995.

[9] J.-H. Chung, H. Yoon, and S. R. Maeng. A systolic array exploiting the inherent parallelisms of artificial neural networks. *Microprocess. Microprogram.*, 33(3):145–159, May 1992.

[10] Computing Community Consortium. CCC visioning study on cross-layer reliability. `http://www.relxlayer.org/`, 2012.

[11] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco. Software-based online detection of hardware defects mechanisms, architectural support, and evaluation. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 97–108, 2007.

[12] J. Detrey and F. de Dinechin. A parameterized floating-point exponential function for fpgas. In *Field-Programmable Technology, Proceedings. IEEE International Conference on*, pages 27 –34, 2005.

[13] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw. A highly resilient routing algorithm for fault-tolerant nocs. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 21–26, 2009.

[14] B. Girau and C. Torres-Huitzil. Massively distributed digital implementation of an integrate-and-fire legion network for visual scene segmentation. *Neurocomput.*, 70(7-9):1186–1197, Mar. 2007.

[15] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 141–151, 2008.

[16] J. Han, E. Taylor, J. Gao, and J. Fortes. Faults, error bounds and reliability of nanoelectronic circuits. In *Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 16th IEEE International Conference on*, pages 247 – 253, july 2005.

[17] R. Hecht-Nielsen. Replicator neural networks for universal optimal source coding. *Science*, 269(5232):1860–1863, 1995.

[18] C. J. Hescott, D. C. Ness, and D. J. Lilja. Scaling analytical models for soft error rate estimation under a multiple-fault environment. In *Proceedings of the 10th Euromicro Conference on Digital System Design Archi-*

tectures, Methods and Tools, DSD '07, pages 641–648, 2007.

[19] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. 313(5786):504–507, 2006.

[20] N. Imam, F. Akopyan, J. Arthur, P. Merolla, R. Manohar, and D. Modha. A digital neurosynaptic core using event-driven qdi circuits. In *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on*, pages 25 –32, may 2012.

[21] H. Iwai. Roadmap for 22nm and beyond (invited paper). *Microelectron. Eng.*, 86(7-9):1520–1528, July 2009.

[22] E. Izhikevich. Simple model of spiking neurons. *Neural Networks, IEEE Transactions on*, 14(6):1569 – 1572, nov. 2003.

[23] A. Jahnke, T. Schönauer, U. Roth, K. Mohraz, and H. Klar. Simulation of spiking neural networks on different hardware platforms. In *Proceedings of the 7th International Conference on Artificial Neural Networks*, ICANN '97, pages 1187–1192, 1997.

[24] X. Jin, M. Lujan, L. A. Plana, S. Davies, S. Temple, and S. B. Furber. Modeling spiking neural networks on spinnaker. *Computing in Science and Engineering*, 12(5):91–97, 2010.

[25] Y. B. Kim and Y.-B. Kim. Fault tolerant source routing for network-on-chip. In *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, pages 12–20, 2007.

[26] T. J. Koickal, L. C. Gouveia, and A. Hamilton. A programmable spike-timing based circuit block for reconfigurable neuromorphic computing. *Neurocomput.*, 72(16-18):3609–3616, Oct. 2009.

[27] M. Krips, T. Lammert, and A. Kummert. Fpga implementation of a neural network for a real-time hand tracking system. In *Proceedings of the The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)*, DELTA '02, pages 313–, 2002.

[28] H. Larochelle and Y. Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 536–543, 2008.

[29] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 2009.

[30] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, ASPLOS XIII, pages 265–276, 2008.

[31] Y. Li, S. Makar, and S. Mitra. Casp: concurrent autonomous chip self-test using stored test patterns. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 885–890, 2008.

[32] R. P. Lippmann. Review of neural networks for speech recognition. *Neural Comput.*, 1(1):1–38, Mar. 1989.

[33] G. Loh. 3d-stacked memory architectures for multi-core processors. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 453–464, 2008.

[34] C. K. Machens. Building the human brain. 338(6111):1156–1157, 2012.

[35] E. J. McCluskey, A. Al-Yamani, J. C.-M. Li, C.-W. Tseng, E. Volkerink, F.-F. Ferhani, E. Li, and S. Mitra. Elf-murphy data on defects and test sets. *VLSI Test Symposium, IEEE*, 2004.

[36] C. Mead. *Analog VLSI and neural systems*. VLSI systems series.

[37] A. Meixner, M. E. Bauer, and D. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 210–222, 2007.

[38] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1 –4, sept. 2011.

[39] J. Misra and I. Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomput.*, 74(1-3):239–255, Dec. 2010.

[40] S. Nassif, K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, E. Nowak, D. Pearson, and N. Rohrer. High performance cmos variability in the 65nm regime and beyond. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 569 –571, dec. 2007.

[41] N. Nedjah and L. de Macedo Mourelle. Reconfigurable hardware for neural networks: binary versus stochastic. *Neural Comput. Appl.*, 16(3):249–255, May 2007.

[42] A. Nere, U. Olcese, D. Balduzzi, and G. Tononi. A neuromorphic architecture for object recognition and motion anticipation using burst-stdp. *PLoS ONE*, 7(5):17 pages, 5 2012.

[43] S. Nomura, M. D. Sinclair, C.-H. Ho, V. Govindaraju, M. de Kruijf, and K. Sankaralingam. Sampling + dmr: practical and low-overhead permanent fault detection. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, 2011.

[44] A. R. Omondi and J. C. Rajapakse. *FPGA Implementations of Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[45] P. Rocke, B. McGinley, F. Morgan, and J. Maher. Reconfigurable hardware evolution platform for a spiking neural network robotics controller. In *Proceedings of the 3rd international conference on Reconfigurable computing: architectures, tools and applications*, ARC'07, pages 373–378, 2007.

[46] D. Rumelhart, J. McClelland, and S. D. P. R. G. University of California. *Foundations: Vol. 1 ; Foundations*. Computational Models of Cognition and Perception.

[47] F. Samman, T. Hollstein, and M. Glesner. Multicast parallel pipeline router architecture for network-on-chip. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 1396–1401, 2008.

[48] J. Sietsma and R. Dow. Neural net pruning-why and how. In *Neural Networks, IEEE International Conference on*, pages 325 –333 vol.1, 1988.

[49] A. W. Strong, E. Y. W. abd R.-P. Vollertsen, J. Sune, G. L. Rosa, T. D. Sullivan, and S. E. R. III. *Reliability Wearout Mechanisms in Advanced CMOS Technologies*. Wiley-IEEE Press, 2009.

[50] C. Tan, R. Gutmann, and L. Reif. *Wafer Level 3-D ICs Process Technology*.

[51] X. Tang and S. Wang. A low hardware overhead self-diagnosis technique using reed-solomon codes for self-repairing chips. *IEEE Trans. Comput.*, 59(10):1309–1319, Oct. 2010.

[52] O. Türel, J. H. Lee, X. Ma, and K. K. Likharev. Architectures for nanoelectronic implementation of artificial neural networks: new results. *Neurocomput.*, 64:271–283, Mar. 2005.

[53] J. Wu and S. Furber. A multicast routing scheme for a universal spiking neural network architecture. 53(3):280–288, 2010.