A Dataflow-Aware Fault Resilience Analysis Framework for Deep Neural Network Accelerators

I. Introduction

While the continuous scaling of CMOS technology nodes has led to improved performance, it has also caused hardware to be more susceptible to *soft errors* [3], [6]. Soft errors, which are temporary errors in hardware caused by radiation or temperature effects [19], have shown to cause failures in DNNs used in safety critical applications like autonomous vehicles (AVs) [8]. With more DNNs running on specialized hardware, it's been found that the resilience of DNN computation is dependent on the design of the accelerator on which inference is being performed [7], [11]. In particular, the way in which data is moved and reused in an accelerator impacts how vulnerable certain computations may be to soft errors.

Given this dependency, understanding how the interaction between the accelerator design and model topology affects the resilience of a DNN system is critical to deploy them safely. Quantification is especially important during the early design phase of such accelerators due to standards like the ISO 26262 [2], which mandates maximum failure rates for systems-on-chip used in AVs. Previous work has shown that transient errors in logic flip-flops alone can exceed the failure rate budget of an accelerator in such systems [7].

Currently there is no generalizable framework that allows for the exploration or quantification of the resilience of an accelerator's *dataflow* design. In this work we take an initial step to develop such a tool for dataflow-aware resilience analysis. We first introduce the idea of "dataflow error sites" (§III-A) as a way to model DNN hardware errors in software. To extract such sites, we use an accelerator's loop nest representation for its dataflow, which has not been done in previous resiliency work. Using these novel methods, we have taken initial steps to create a generalizeable software pipeline for resilience analysis given high-level design descriptions of an accelerator's dataflow (§III-B). Finally, we describe preliminary results demonstrating the framework's capabilities and discuss future directions (§III-C and §III-D).

II. Related Work & Background

Related Work: Previous research has developed methods for performing hardware-agnostic software injection into DNNs [12], [13], [17]; however, such methods ignore the hardware dependency of error propagation. FIdelity [7] aims to address this by developing an accelerator-aware framework for modeling logic transient errors; however, they lack support for analyzing the memory hierarchy design, where most data is stored and reused. In addition, while FIdelity provides psuedo-code describing their "reuse analysis", this forces users to implement the proposed framework themselves, limiting its productivity. Other studies have performed single case studies of memory errors in specific accelerator architectures

```
#---- DRAM ----#                              Key:
for m1=[0:M1]:                                o=[M,Q,P]       # Outputs
 pfor m0=[0:M0]: # parallel                   i=[C,Q+S,P+R]   # Inputs
  for c1=[0:C1]:                              w=[M,S,R]       # Weights
   for q1=[0:Q1]:                             M=M1*M0; C=C1*C0; etc.
    for p1=[0:P1]:
#----- weight/input buffer -----#
      pfor c0=[0:C0]: # parallel
       for r1=[0:R]:
        for s1=[0:S]:
         for q0=[0:Q0]:
          for p0=[0:P0]:
#----- PE weight/input regs -----#
           r0 = s0 = 0 # no tiling in s/r
           k = m1*M0+m0; c = c1*C0+c0; r = r1*R0+r0
           s = s1*S0+s0; p = p1*P0+p0; q = q1*Q0+q0
           o[m,q,p] += i[c,q+s,p+r] * w[m,s,r]
```

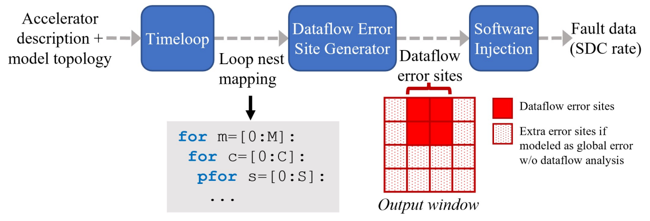
Fig. 1. NVDLA [1], [10] loop nest dataflow example. Various sections are commented, marking points below which all accessed elements are contained in that memory type. For example, DRAM is shown as being at the top-level loop, meaning that all the accessed elements for outputs, inputs, and weights will be contained within DRAM. For the weight/input buffer (or CBUF), it contains all weights for a consecutive $M0$ weight kernels, as well as an input tile across consecutive $C0$ input channels.

[4], [11], but they do not supply researchers with a generalizable framework for analyzing memory error resilience without access to low-level design descriptions or RTL.

DNN Accelerator Dataflow: DNN accelerators, in general, each implement a dataflow, which defines how it schedules a layer's computations [9], [15]. More specifically, the dataflow defines how computations are staged spatially across an array of parallel processing units, composed of MAC units and registers. It also specifies how computations are scheduled temporally, meaning the order in which the MACs are performed and how data is moved within the memory hierarchy (e.g., from off-chip DRAM to registers).

While it's well understood that dataflow has a large effect on an accelerator's performance due to different data reuse and MAC utilization [15], it's still unclear how the dataflow impacts resilience. Recent flexible dataflow architectures even allow for different mappings for the same workload [5], [18], motivating the need to understand how mapping can affect resilience.

For this work, we focus on CNNs as they are frequently used in safety critical settings like object detection in AVs. Due to the "sliding window" property of convolutions, inputs and weights are used for multiple output neuron computations. Thus, a single bit-flip can result in large errors at the output, as that one erroneous value will be reused across multiple output neurons. In order to categorize and describe an accelerator's dataflow, we utilize the fact that the computation of a convolution can be described by a 7-D loop nest (as illustrated in Fig. 1). The HW mapping of a workload (i.e., a CNN layer) for a dataflow can be described from a loop nest that reorders, tiles, and parallelizes the 7-D loop nest. The resulting altered loop nest models how different pieces of data (i.e., inputs, weights, and outputs) are reused temporally and spatially.


Accelerator description + model topology
Timeloop
Loop nest mapping
Dataflow Error Site Generator
Dataflow error sites
Software Injection
Fault data (SDC rate)
Dataflow error sites
Extra error sites if modeled as global error w/o dataflow analysis
Output window

```
for m=[0:M]:
 for c=[0:C]:
  pfor s=[0:S]:
   ...
```

Fig. 2. A simplified block diagram showing key components of the framework. Given some architecture description and DNN model architecture, Timeloop [15] produces a loop nest mapping. This mapping is then used by the dataflow error site generator to produce the dataflow error subset of the output window (shown in red). This can then be used to isolate errors within the software injection frontend to estimate SDC rates.

III. Proposed Solution & Preliminary Results

We propose a generalizable software framework for analyzing DNN accelerator resilience given dataflow descriptions. We first discuss our novel insight into modeling dataflow error propogation, building off of the "reuse factor" proposed by previous work [7] and tailoring it to memory errors in an accelerator dataflow. We then describe the implementation and preliminary results.

A. *Modeling Dataflow Error Propagation*

Because DNN accelerators have well-defined dataflows, it's possible to deterministically model how errors propagate through a memory hierarchy given a loop nest description. If an error occurs prior to any computation at the highest level of the memory hierarchy (e.g. DRAM), we call such an error a *global error*. A global error will affect the entire *output window* of that element, meaning all the output neurons for which that value is used for a partial sum. Pure software injection frameworks without any dataflow analysis can model such global errors by injecting into software-visible state.

However, if an error occurs within the memory hierarchy (e.g. in a buffer or register) during computation, only a subset of the output window will use that faulty value. For example, if an error occurs in a memory element after the value has already been used, the output neurons that used the "clean" value prior to the error will not be erroneous.

The dataflow and mapping of an architecture, given an error occurring at some time and location in memory, are what affect this erroneous subset of output neurons, and we call this subset *dataflow error sites*. This means that a memory error in hardware can be modeled in software by performing a global bit-flip injection and then only selecting the corresponding erroneous values that make up the dataflow error sites at the output. The only difference between two dataflows when modeling errors in software is the size and shape of this subset.

B. *Framework Description*

To allow for dataflow-aware error propagation modeling, we propose a two-stage analysis method as depicted in Fig. 2. The first step involves a "Dataflow Error Site Generator" backend. This backend uses an architecture's loop nest mapping for a given workload, supplied by Timeloop [15], which can perform this mapping for any dataflow, allowing for wide generalizability. This loop nest description allows us to simulate error propagation, tracking how a value within a memory level (DRAM, buffer, etc.) is used temporally and spatially within the output window.

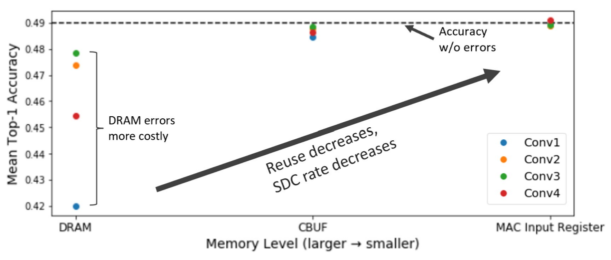


Fig. 3. Top-1 Accuracy in Alexnet-ImageNet as a result of modeled memory errors in input activations using our framework. The targeted accelerator is NVDLA [1] using the loop nest shown in Fig. 1. We find that as reuse decreases down the memory hierarchy (from DRAM to CBUF to regs), the SDC rate also decreases drastically. Moreover, the layer in which the injection occurs has a major impact.

The second step involves a "Software Injection Frontend" which takes the dataflow error sites generated from the backend to isolate those sites in software, providing accurate hardware error modeling in the software visible output activations. This frontend uses PyTorch [16] to propagate a layer's errors to the final model output. The final output can then be used to collect failure data such as silent data corruptions (SDCs), which are errors at the final output, such as image misclassification. This output data are also useful for validation purposes (§III-D).

C. *Preliminary Results*

Fig. 3 shows preliminary results using an early-stage implementation of the framework. We use the loop nest for NVDLA [1] as shown in Fig. 1 to model hardware errors in the input activations for different memory levels. We used Alexnet as the model topology, performing injections in convolutional layers (ignoring layers with non-unit stride due to limitations in the current framework). We used a random set of 100 images from ImageNet's validation set and randomly sampled input locations to inject an error. Values are represented in FP32, and errors are modeled as a flip of the fifth bit, as bit location randomization has not yet been implemented. Future work will do a thorough analysis on different representations and bit locations.

The short case study enabled by the framework already reveals that reuse has a large effect on resilience, with errors occurring in DRAM, the highest memory level, decreasing the accuracy by as much as 6× as compared to errors in buffer or register elements. This gives accelerator designers an idea of locations in the memory hierarchy that are particularly vulnerable. We plan to do further case studies comparing the resilience of various accelerator dataflows to characterize how the dataflow impacts resilience and to explore tradeoffs between energy, accuracy, and resiliency.

D. *Validation*

To be true to hardware, the dataflow error sites that the backend produces need to be validated. We've already used the reported error models for NVDLA [1] from FIdelity [7] and compared them with the dataflow error sites produced by our framework, and found that they exactly match, which is promising considering FIdelity validated their work with RTL injections. Future validation will compare results to fault injections performed on either a cycle-accurate accelerator simulator, like STONNE [14], or RTL for simple dataflow accelerator designs.

References

[1] "NVIDIA Deep Learning Accelerator." [Online]. Available: http://nvdla.org/

[2] "ISO 26262," Sep. 2021, page Version ID: 1046300685.

[3] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, May 2005, conference Name: IEEE Design Test of Computers.

[4] N. Chandramoorthy, K. Swaminathan, M. Cochet, A. Paidimarri, S. Eldridge, R. V. Joshi, M. M. Ziegler, A. Buyuktosunoglu, and P. Bose, "Resilient Low Voltage Accelerators for High Energy Efficiency," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2019, pp. 147–158, iSSN: 2378-203X.

[5] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, Jun. 2019, conference Name: IEEE Journal on Emerging and Selected Topics in Circuits and Systems.

[6] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *2011 International Reliability Physics Symposium*, Apr. 2011, pp. 5B.4.1–5B.4.7, iSSN: 1938-1891.

[7] Y. He, P. Balaprakash, and Y. Li, "FIdelity: Efficient Resilience Analysis Framework for Deep Learning Accelerators," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2020, pp. 270–281.

[8] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "AVFI: 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018," *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*, pp. 55–56, Jul. 2018, publisher: Institute of Electrical and Electronics Engineers Inc.

[9] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 754–768. [Online]. Available: https://doi.org/10.1145/3352460.3358252

[10] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous Dataflow Accelerators for Multi-DNN Workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 71–83, iSSN: 2378-203X.

[11] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3126908.3126964

[12] G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A Configurable Fault Injector for TensorFlow Applications," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Oct. 2018, pp. 313–320.

[13] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: A Runtime Perturbation Tool for DNNs," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Jun. 2020, pp. 25–31, iSSN: 2325-6664.

[14] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STONNE: Enabling Cycle-Level Microarchitectural Simulation for DNN Inference Accelerators," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 122–125, Jul. 2021, conference Name: IEEE Computer Architecture Letters.

[15] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2019, pp. 304–315.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[17] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, Jun. 2018, pp. 1–6.

[18] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 14–27. [Online]. Available: https://doi.org/10.1145/3352460.3358302

[19] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *The International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 129–173, 2014. [Online]. Available: https://doi.org/10.1177/1094342014522573