Thread-level Parallelism in Fault Simulation of Deep Neural Networks on Multi-Processor Systems

Masoomeh Karami[1], Mohammad-Hashem Haghbayan[1], Masoumeh Ebrahimi[1,2], Antonio Miele[3], Juha Plosila[1]
[1]Department of Future Technologies, University of Turku, Turku, Finland
[2]Department of Electronics and Embedded Systems, Royal Institute of Technology (KTH), Kista, Sweden
[3]Dip. Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy
Email: {mkaram, mohhag, juplos}@utu.fi, mebr@kth.se, antonio.miele@polimi.it

***Abstract*—High-performance fault simulation is one of the essential and preliminary tasks in the process of online and offline testing of machine learning (ML) hardware. Deep neural networks (DNN), as one of the essential parts of ML programs, are widely used in many critical and non-critical applications in Systems-on-Chip and ASIC designs. Through fault simulation for DNNs, by increasing the number of neurons, the fault simulation time increases exponentially. However, the software architecture of neural networks and the lack of dependency between neurons in each inference layer provide significant opportunity for parallelism of the fault simulation time in a multi-processor platform. In this paper, a multi-thread technique for hierarchical fault simulation of neural network is proposed, targeting both permanent and transient faults. During the process of fault simulation the neurons for each inference layer will be distributed among the executing threads. Since in the process of hierarchical fault simulation, the faulty neuron demands proportionally enormous computation comparing to behavioural model of non-faulty neurons, the faulty neuron will be assigned to one thread while the rest of the neurons will be divided among the remaining threads. Experimental results confirm the time efficiency of the proposed fault simulation technique on multi-processor architectures.**

***Index Terms*—Multi-threading Fault Simulation, Neural Network, Reliability**

I. Introduction

Neural networks (NNs) has shown potential in solving various problems such as classification tasks in different domains. To achieve high accuracy in complex applications, NNs are getting deeper, so-called Deep Neural Networks (DNNs), which extend their applications to critical domains such as surgery robots and rescue drones [1]–[3]. Considering safety-critical applications, the neural network computations should satisfy three main requirements as *high accuracy*, *short execution time* and *reliable operation* [4]. Higher accuracy is usually obtained by designing new DNN architectures. To achieve short response time, recently large amount of effort has been devoted to accelerate DNNs in software and hardware, including specialized hardware and ASIC design [3]–[5]. To ensure the reliability in the hardware level, the whole network has to be simulated and the impact of hardware faults should be analyzed. For this purpose, first all possible faults, either at design or run time, should be identified and then modeled properly. Then faults have to be injected into the simulator to analyze fault propagation, observe their impact on different elements of the neural network, and most importantly the impact on the final accuracy. This process of evaluating the behaviour of the hardware with regard to the injected faults is called *fault simulation*. Considering all possible locations that a fault may occur in a DNN, simulation of these faults using traditional methods is practically infeasible, and thus a specifically-designed fault simulation procedure is demanded.

978-1-6654-5938-9/22/$31.00 ©2022 IEEE

There has been a massive amount of research trying to improve the fault simulation time on single-core [6], [7] and multi-core systems [8], [9]. Beside the ideas to improve the simulation of circuit, e.g., event-driven and multi-thread simulation techniques, there has been several ideas that exploit the independent effect of different faults on the circuit to increase the parallelization of fault simulation, e.g., parallel fault simulation [8], [9] and concurrent fault simulation [10]. In the fault simulation process, besides the general behaviour of the fault's effects on the circuit, the structure of the circuit under test (CUT) also helps significantly to improve the fault simulation time. Unfortunately, most of the proposed ideas are focused on a general-purpose circuit architectures, improving the average fault simulation time, where the structure of the circuit is not usually considered. We argue that for the fault simulation of DNNs, their special repetitive and hierarchical structure provides a great opportunity to significantly reduce the fault simulation time.

In this paper, we propose a multi-threading hierarchical fault simulation which is specifically designed for DNNs. In proposed approach, all operations in the DNN is divided into two categories as faulty and non-faulty. Faulty operations are simulated in the hardware level while non-faulty operations are simulated using higher abstraction models (e.g., C++ level). Based on being faulty or non-faulty module the neurons are mapped to the suitable thread for processing.

More specifically, during the fault simulation, only the places on the CUT where the fault is being injected must be simulated based on the lowest abstraction levelwhile other places can be simulated based on the higher possible modeling abstraction level. To enhance the performance of the simulation, it is important to map neurons efficiently to make

balance the overhead load of gate-level implementation. The hierarchical structure consists of different levels of abstraction where the top level is the behavioural DNN. The lowest level considered in the hierarchy is the gate-level that is the level for fault injection. The proposed multi-threading hierarchical fault simulation could both satisfy fast simulation time and hardware-level fault simulation.

To further decrease the fault simulation time, the proposed approach is combined with traditional techniques such as event-driven and parallel fault simulation. The main contributions of this paper are as follows:

- Implementing a multi-threading for hierarchical fault simulation of deep neural network architectures.
- Adapting the hierarchical cross-level fault simulation for DNNs, for multi-threading while utilizing their special structure.
- Implementing the proposed event-driven multi-thread simulation on a multi-processor platform.

II. Background and Related Work

A. Fault Simulation

Several studies have tried to decrease the fault simulation time by utilizing different techniques such as *parallelization* [11]–[14], *event-driven simulation* [15], [16], and the use of *mixed abstraction levels* of system presentation [17], [18].

In the parallelization technique, mutually exclusive parts of the circuit are simulated in parallel [11], [12]. Another form of the parallelization is to simulate the CUT for different sets of faults [14] and different test patterns [13]. The former is called data-parallel fault simulation while the later is known as pattern-parallel fault simulation.

A *event-driven* method, only the propagation of occurred *events*, e.g., injecting a fault or a change in the wire, are tracked and processed by the simulator [15], [16]. Using the event-driven fault simulation, the simulator avoids re-processing those parts of the CUT that there is no change into its inputs. The event-driven approach is more efficient than time-driven method by 1) being faster, 2) using less memory, and 3) being more flexible [15], [16].

In the fault simulation process, there exists a tight relationship between the fault model and the modeling abstraction level. The high-level fault models can be simulated via higher modeling abstract of the CUT. This results in low simulation time. However, high-level fault models usually result in a low coverage of real failures. The stuck-at-fault model in the gate-level modeling abstraction level, is considered as one of the most suitable fault models w.r.t. covering the CUT's real failures and simplicity of the model. We assume the stuck-at-fault model throughout the paper.

The CUT's modeling abstraction level cannot be higher than the modeling level envisioned for the fault model. For example, if the fault model is the stuck-at-fault model, then the CUT should be modeled in gate-level to able to inject faults. Therefore, it is not possible to get benefit from the fast simulation time, offered by the high-level modeling of the CUT. Mixed-level fault simulation techniques solve this issue by using the higher abstraction level of CUT for non-faulty partitions (e.g., the behavioral level [17]) and the low-level model (i.e., compatible with the fault model) only for the area of CUT that the fault(s) is intended to be injected [17]. In fact, mixed-level fault simulation is a technique to combine the benefits of: 1) low simulation time provided by modelling in high abstraction levels, and 2) high coverage offered by low-level fault models [18].

The main drawback of the existing mixed-level fault simulation methods is that the effectiveness of such techniques is highly dependant on the structure of the CUT and the possibility of modelling the CUT in a hierarchical way. Therefore, before applying fault simulation, the structure of the CUT in different levels of abstraction should be investigated carefully. Another issue is the integration of the mixed-level fault simulation with other fault simulation techniques such as parallel and/or event-driven fault simulation. For example, in the case of simulating several faults in parallel while performing mixed-level fault simulation, it is more efficient to simulate the nearby faults together. This is due to the fact that nearby faults on the CUT most probably belong to the same module, requiring only one low-level instance of that module to inject faults.

B. DNN Fault Simulation

Various studies implement the fault simulation of DNNs [4], [19] by injecting faults and analyzing the results. Xun et. al. [5] assesses the vulnerability of DNNs in hardware by injecting timing errors into the network during inference. However, most of these works cannot be generalized as they focus on specific fault models or DNN model. There are also some frameworks which provide the possibility of injecting and analyzing faults. Among these frameworks is TensorFI which can be used for DNNs supported by TensorFlow [20]. In TensorFI, computations are presented by a data-flow graph where faults can be injected into them. Although TensorFI provides the fault injection and fault analysis features at a high speed, it does not support fault models in lower abstraction levels such as stuck-at fault models in the gate level. This implies that the fault coverage of TensorFI is low and limited to the faults that occur in higher abstraction levels. Li et al. [21] implements a fault injector simulator by using the tiny-CNN framework. The simulator maps each line of the code in the framework to the corresponding hardware component for injecting faults. This work considers transient faults with the bit-flip fault model.

III. Multi-threading Hierarchical Fault Simulation of DNN

Figure 1 shows a sample of DNN and a neuron structure. The neuron structure follows a repetitive architecture, which includes multiply and accumulate (MAC) units that can be computed in parallel. In our proposed fault simulation method, we first define different abstraction levels from the DNN top-level down to the gate level as shown in Figure 1. This figure

illustrates a feed-forward network where the network level is composed of several computational layers that are cascaded *horizontally* from the DNN inputs towards the DNN outputs. Each layer contains several neurons that are connected, via *edges*, to the neurons of the preceding and subsequent layers. A neuron comprises several multiplication units and accumulation. The lowest abstraction level, considered in this paper, is the gate level where stuck-at fault models can be implemented and evaluated. More details of the abstraction level of DNN can be found in [22]. These four abstraction levels are the ones that are used in our proposed fault simulation. However, with a marginal change, the algorithm can be adapted for a design with other granularities and different number of abstraction levels.

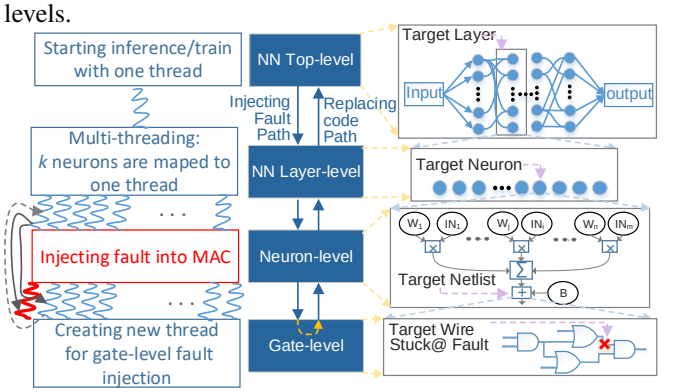

Figure 1. Simulation abstraction levels [22] and new added multithreading approach

Hierarchical fault model: We use a top-down approach in the fault-injection process. At the highest level (NN top-level), the impact of the fault can be observed from the output of the whole DNN network. In the level of NN layer, the output of the layer is the indicator on whether a fault has occurred or not. The fault model in the neuron level concerns the output of the neuron to see whether it is deviated from the expected value or not. The fault model in the lowest level is stuck-at fault which is injected based on the fault pattern. By this hierarchical approach, the location of a fault is positioned from the highest to the lowest level while the the fault propagation can be observed and evaluated from the lowest level to the highest level. This allows us to analyze whether stuck-at faults are masked at higher levels and also at which abstraction level.

Event-driven fault simulation: In the hierarchical fault simulation of DNNs, an event is defined as a change in the input(s) of a gate, a neuron, a layer, or the whole DNN. Thereby, in each hierarchical level, if an event is observed in the input, the related module and all its subsequent modules will be simulated.

Stuck-at faults embedded in gate-level: As was mentioned earlier, our finest granularity level to inject faults is the gate level with the fault model defined as stuck-at faults. To enable the injection of stuck-at fault models, the main operators in a DNN model, including multiplication and addition, will be equipped with the stuck-at fault injection capabilities. There are two implementations of an operator, one is the behavioral description in C++ and the other one is the fault-injectable gate-level description in C++. In case no event is triggered, the behavioral description of an operator will be used, otherwise, the operator is replaced with its equivalent fault-injectable operator.

Multi-thread fault simulation: As Figure 1 shows, in the first abstraction level, the simulation execution starts with one thread. Then, simulation is executed layer by layer, since each layer needs the output of previous layer. In each layer, threads process their mapped neurons. Consider in each layer we have N neurons and T threads. So, k neurons are assigned to T threads, where each thread executes k $(=\frac{T}{N})$ neurons. If a neuron is the target neuron (faulty neuron), a new thread is added to take care of the execution of the injected gate-level faults. Figure 2 shows how neurons' computation in each layer is split into different threads. The thread where a fault is injected is called *faulty thread*. Behavioral threads refer to the threads that execute non-faulty neurons which are implemented at the behavioral abstraction level.

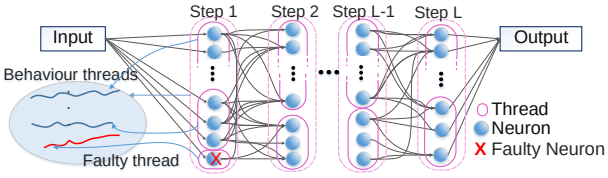

Figure 2. Multi-threading in the layer abstraction level

Algorithm 1 illustrates multi-threading fault-injection method. The inputs are: *DNN* that is the specification of the DNN module based on the abstraction level; *FN* which defines the fault number in that DNN; and T that is the number of threads. For each layer of the DNN, if the fault is targeted a neuron, the algorithm dedicates a single thread to simulate that neuron, Line (2-5). For the remaining neurons that are not faulty, i.e., that will be simulated in behavioural level of abstraction, the algorithm splits them equally among the remaining $T-1$ threads, Line 5-6. This process step by step proceeds until simulating all the DNN's layers.

Algorithm 1 The multi-threading fault-injection algorithm

Inputs: FN, DNN, T;
Body:
1: **for** All layers L of DNN **do**
2: $N \leftarrow$ Neurons(L);
3: **for** The faulty neuron $n \in L$ **do**
4: Assign neuron to faulty thread;
5: **for** The set of non-faulty neurons $N = L - n$ **do**
6: Assign $\frac{|N|}{(T-1)}$ neurons to $T-1$ behavioural thread(s);

IV. Experimental Setup and Result

To evaluate the proposed technique, we adopt the well-known LeNet-5 neural network model, coded in C++ programming language, simulated on the tiny-DNN simulation environment, and based on Ubuntu OS. The CPU specifications for processing the simulation is: the total cores is 20, the total threads is 40. The architecture of the neural network consists of six layers of neurons, including three convolution layers, two average pooling layers, and a fully connected layer. The output layer of the neural network composed of ten neurons, representing the associated label for the given input. In this neural network model, the total number of MAC modules is

341k modules. The fault simulation input is selected from the MNIST benchmark [23].

We select 32-bit variables (i.e., 10-bit fraction, 21-bit integer, and 1-bit sign) as precision levels. For gate-level implementation of the neural network, fixed-point number representation is used while for modeling the high-level specification of the neural network, floating-point variable in C++ is used. The reason is that compared to floating point, fixed-point variable consumes smaller hardware area while the loss of precision is negligible, specially when the bit-width of the fixed point is high.

Figure 3 shows the obtained fault simulation time for the *LeNET*. The simulation time for double thread compared to one thread 56% decreases. It means while using two threads, the simulation time goes below the half. The reason is that using two threads not only utilize two processing elements for simulation, but also it provides a opportunity for utilizing the memory hierarchy of the system in more efficient way. While using single thread, the high miss rate in first-level private caches of the core, cause noticeable latency in the simulation process. This is revealed when using two threads. Another fact also that supports this result is the weak dependency of data in each layer of the neural network that makes less time consuming memory access from different location of memory hierarchy.

By increasing the number of threads from two to three and four, the simulation time 24% and 28% becomes better respectively. As it can be seen, the simulation time improvement saturates and becomes less while the number of threads are increasing which is due to overhead of multi-threading. In some cases, this overhead slightly worsens the simulation time with respect the obtained results of fewer number of threads. The best obtained simulation time is for 10 threads that reduces the time by 81%.

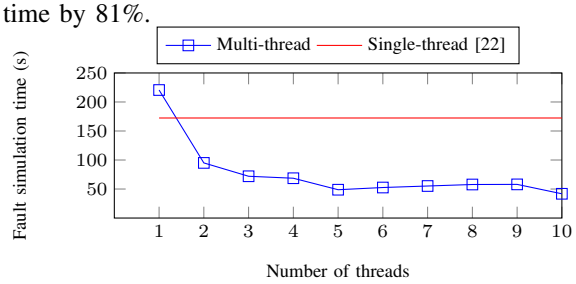

Figure 3. Multi-thread hierarchical fault simulation time for LeNET versus the number of executing threads.

V. Conclusion

In this paper, a multi-thread hierarchical fault simulation for deep neural networks (DNNs) is presented. The faulty module in this model that is modeled in netlist is assigned to one thread to be executed while the other computing parts are divided between the remaining threads. To increase the performance and avoid unnecessary simulations, different event models in gate-level and neuron-level are defined. The events in gate-level are defined as any changes in the gates' inputs, i.e., bit-flip, while the events in the neuron level are the changes in the neuron's inputs. Concurrency in different levels are accelerated via pipelining. Experimental results shows that the proposed technique decreases the fault simulation time in comparison with traditional single-thread fault simulation techniques.

Acknowledgment

This work has been financially supported by the Academy of Finland funded projects 335512 - ADAFI (Adaptive-Fidelity Digital Twins for Robust and Intelligent Control Systems) and 330493 - AURORA (Autonomous Performance Management in Digital Manufacturing), and by Nokia Jorma Ollila Grant.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," in *Nature*, 2015, p. 436–444.

[2] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," in *IEEE*, 2017, pp. 2295–2329.

[3] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," in *IEEE Access*, 2019, pp. 7823–7859.

[4] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," in *Journal of Systems Architecture*, 2020.

[5] X. Jiao, M. Luo, J. Lin, and R. K. Gupta, "An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations," in *ICCAD*, 2017, pp. 945–950.

[6] N. Bombieri, F. Fummi, and V. Guarnieri, "Accelerating rtl fault simulation through rtl-to-tlm abstraction," in *ETS*, 2011, pp. 117–122.

[7] Z. Navabi, *Digital System Test and Testable Design: Using HDL Models and Architectures*. Springer Publisher, 2010.

[8] S. Hadjitheophanous, S. N. Neophytou, and M. K. Michael, "Scalable parallel fault simulation for shared-memory multiprocessor systems," in *VTS*, 2016, pp. 1–6.

[9] M. Gorev, R. Ubar, and S. Devadze, "Fault simulation with parallel exact critical path tracing in multiple core environment," in *DATE*, 2015, pp. 1180–1185.

[10] D. G. Saab, "Parallel-concurrent fault simulation," in *IEEE Transactions on VLSI Systems*, 1993, pp. 356–364.

[11] V. N. et al., "Fault Simulation on Massively Parallel SIMD Machines: Algorithms, Implementations and Results," in *J. Electron Test*, 1992.

[12] S.-E. Tai and D. Bhattacharya, "Pipelined fault simulation on parallel machines using the circuit flow graph," in *ICCD*, 1993, pp. 564–567.

[13] K. Gulati and S. Khatri, "Towards acceleration of fault simulation using graphics processing units," in *DAC*, 2008, pp. 822–827.

[14] R. Mueller-Thuns *et al.*, "VLSI logic and Fault Simulation on General-purpose Parallel Computers," in *IEEE Trans. on CAD of Integrated Circuits and Systems*, 1993.

[15] E. Gascard and Z. Simeu-Abazi, "Quantitative analysis of dynamic fault trees by means of monte carlo simulations: Event-driven simulation approach," in *Reliability Engineering and System Safety*, 2018, pp. 487–504.

[16] J. A. Garrido, R. R. Carrillo, N. R. Luque, and E. Ros, "Event and time driven hybrid simulation of spiking neural networks," in *Advances in Computational Intelligence*, 2011, pp. 554–561.

[17] S. Mirkhani, M. Lavasani, and Z. Navabi, "Hierarchical fault simulation using behavioral and gate level hardware models," in *ATS*, 2002, pp. 374–379.

[18] M. Karami, A. Abdi, and H. R. Zarandi, "A cross-layer aging-aware task scheduling approach for multiprocessor embedded systems," in *Microelectronics Reliability*, 2018, pp. 190–197.

[19] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," in *IEEE Access*, 2017, pp. 17 322–17 341.

[20] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," in *arXiv*, 2020.

[21] 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 accelerators and applications," in *High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[22] M. Karami, M.-H. Haghbayan, M. Ebrahimi, A. Miele, H. Tenhunen, and J. Plosila, "Hierarchical fault simulation of deep neural networks on multi-core systems," in *ETS*, 2021, pp. 1–2.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *IEEE*, 1998, pp. 2278–2324.