Understanding and Hardening the Resiliency of Machine Learning Applications in CPU and GPU

YAFAN HUANG & MD HASANUR RAHMAN, The University of Iowa, USA

With the ever-shrinking size of transistors and large-scaled computing systems, soft errors are more likely to appear, which leads to silent data corruptions (SDC) hence reducing system reliability. Due to the massive thread-level parallelisms in GPU, machine learning program in GPU has become a popular research topic. Previous works have explored program resilience against SDC on CPU and GPGPU programs rather than GPU-accelerated machine learning programs. In this work, we aim to understand and harden machine learning program resiliency on two Rodinia mini-apps. We hope that this can bring some insights into future research that is related to machine learning system resiliency.

CCS Concepts: • **Parallel Computing, Program Resilience**;

Additional Key Words and Phrases: datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Yafan Huang & Md Hasanur Rahman. 2021. Understanding and Hardening the Resiliency of Machine Learning Applications in CPU and GPU. In *Course Report'21: CS:4980:0003 Fall21 Topics in Computer Science II, Aug 23–Dec 10, 2021, Iowa City, IA.*

1 INTRODUCTION

Due to the scaling of transistor sizes, low operating voltage, and large-scaled computing systems, modern processors are more susceptible to high-energy particle strikes. Such phenomena leads to a higher probability of bit-flips in stored data or execution units, called "soft errors". Silent data corruption (SDC), which is recognized as one of the most severe results of soft errors compared with benign and crash, produces wrong program output without any noticeable exceptions. As soft errors may happen anywhere in a hardware device and there are countless error propagation routines, tracking SDCs inside hardware is extremely energy-cost. Thus, researchers seek to understand error propagation at the software level. Fault injection (FI) is the fundamental practice to simulate soft errors and thus understand the resilience of programs. By injecting faults into different level, such as LLVM-IR or SASS, the program resiliency can be revealed with a pretty low cost and flexible design.

Machine learning algorithms such as deep neural networks and their downstream applications have attracted enormous research attention, thanks to the powerful computational capability and flexible programmability of modern processors. However, in some safety-critical machine learning applications, such as autonomous driving system (ADS), soft errors can be a very dangerous issue, especially where correct prediction is strictly required. Take Figure 1 for example, incorrect classification in ADS caused by soft errors can lead to few possible fatal consequences (e.g. misclassifying a "stop" sign as a "go ahead" sign). Besides ADS, there are plenty of machine learning usage scenarios, such as natural language processing and training on large distributed systems. Previous works also have already demonstrate that training time in machine learning models can be significantly increased due to soft errors. As a result, it is crucial to understand the resiliency of machine learning programs and systems, and selectively protect the most vulnerable parts at a particular tolerance overhead.

Fig. 1. Soft error outcome on the classifier models. Images at the first row are the input images, those at the second row are the faulty outputs.

In this work, we aim to i) profile the program resiliency by conducting comprehensive FIs on CPUs, ii) find challenges to study error propagation in GPU-accelerated machine learning programs, iii) proposes a greedy mitigation strategy, which leverages selective instruction duplication (SID) technique, for GPU-accelerated programs.

Our finished tasks are as below:

- Conduct comprehensive instruction-level fault injections on two CPU machine learning micro-kernels.
- Analyze the results and provide a basic understanding of machine learning program resiliency.
- Reproduce SID workflow, hence demonstrate its efficiency in hardening resiliency against soft errors.
- Identify the challenges in make program error resilient in GPU programs, and how to mitigate the error propagation. Also, we aim to provide insights for future researches on resiliency on machine learning systems.

Our future works are listed as follows:

- Explore the large machine learning systems resiliency against soft errors.

The rest of this reports are organized as follows. In Section 2, we found and briefly summarized three representative works on this topic. In Section 3, we introduce the basic ideas on how selective instruction duplication works. In section 4, we provide the experimental setup including benchmarks we used and fault model we designed. In section 5, we describes our research direction step by step. Finally, we summarize our observations and discuss some of the potential future works.

2 RELATED WORKS

Researchers have studied a lot about error propagation for applications in CPU and GPGPU. But there is only a handful of study in GPU-accelerated machine learning programs that are related to soft

error propagation.

2.1 SC'16: Understanding Error Propagation in GPGPU Applications

This paper[6] investigates error propagation characteristics across general purpose GPU kernel boundaries. It tracks memory data across kernel boundary, rather than within a kernel. After each kernel run, it compares the current saved memory data with that from a golden run to mark any difference to detect the soft error, which is often called as silent data corruption (SDC). But the main limitation of this paper is that it doesn't consider error propagation inside kernel, which is often believed to be hardest task because there are so much interleave dependencies among different threads.

2.2 SC'21: G-SEPM: Building an Accurate and Efficient Soft Error Prediction Model for GPGPUs

This paper[7] estimates the resilience characteristics without performing the extensive FI (fault injection) campaigns. It leverages certain features such as instruction-type, bit-position, bit-flip direction for the machine learning model. For example, feature such as bit-position is influential in making good prediction of a faulty site because the higher the position for a bit to be flipped is, the more significant effect on error propagation it has. However, one of the key limitations of machine learning (ML) models, let alone this work, is that we need a lot of training samples. Yet, for acquiring more training samples for this work, we need to conduct extensive FI campaigns to detect the error sites. Moreover, faulty sites can be benchmark specific, meaning that we need to do rigorous study to make the ML model more generic. Figure 2 gives the overview of G-SEPM workflow.

Fig. 2. The overview of G-SEPM workflow.

2.3 DSN'17: Evaluation and Mitigation of Soft-errors in Neural Network Based Object Detection in Three GPU Architectures

This paper[3] proposes Algorithm-based fault tolerance(ABFT) strategy for DarkNet to detect and correct soft errors in matrix multiplication operations. This work rearranges the input and convolutional kernels to make it general matrix to matrix multiplication (GEMM). This work is only limited to those structures where there are lots of matrix multiplications. Figure 3 demonstrates the radiation setup by which soft errors are injected.

3 SELECTIVE INSTRUCTION DUPLICATION

In this section, we describe the workflow of a general instruction-level error mitigation strategy, selective instruction duplication (SID), in details.

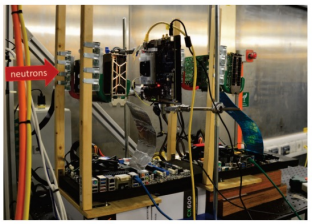


Fig. 3. The radiation setup.

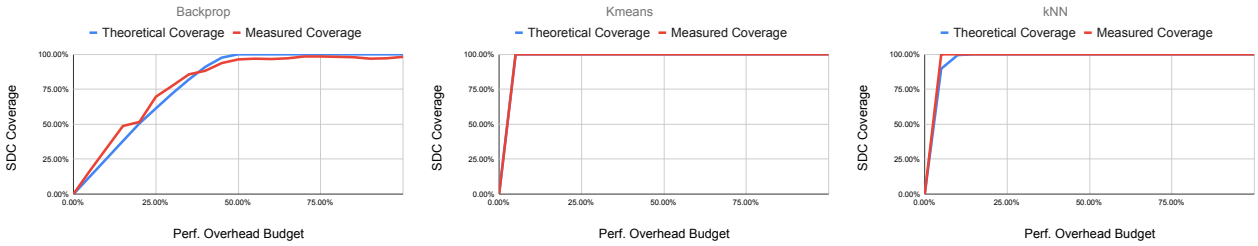
Although hardware mitigation based methods can guarantee a high coverage, they are costly in energy perspective, and are only adopted in mission-critical systems. Software mitigation methods thus attract the researchers' attention due to its efficiency and flexibility. Selective instruction duplication, of which example is shown in Figure 5, is a general software strategy to build protection against soft errors by only duplicating the most vulnerable instructions[1]. As shown in Figure 5(c), while instructions c and d are the most vulnerable parts in original program, we can only duplicate them and then insert a comparison instruction (we called cmp_flag) at the compile time . If any soft error occurs in instruction c or d, the cmp_flag can then detect it via comparing the original value with it of the duplicated one at a run-time. The program will then shut down when any mismatch is found. Comparing with full duplication, SID can achieve a comparable protection with a quite low overhead – the only prerequisite is we need to identify the most vulnerable instructions.

To identify the most vulnerable instructions, SID formulates protection as a 0-1 knapsack problem. For every instruction, the cost is the dynamic cycle (i.e. number of the execution cycle), while the benefit represents the SDC probability if soft errors happens in this instruction. Give a performance overhead budget, knapsack can then find the most vulnerable instruction list, after which instructions will be duplicated in the SID phase. Overall, SID can be divided into three parts in our practice:

- *Per-instruction FI*: We perform 100 FI to each instruction and obtain its per-instruction SDC probability. Note that each static instruction has numerous dynamic cycle in running time, and the FI sites we choose are based on random sampling.
- *Knapsack Engine*: Given the cost and benefit of each instruction, knapsack engine can provide the optimal instruction list which guarantees a maximum overall SDC coverage within a certain performance overhead budget.
- *Duplication Phase*: We duplicate instructions based on vulnerable list provided by knapsack engine. A protected program can then be generated.

4 EXPERIMENTAL SETUP

In this section, we present the experimental setup including benchmarks and fault model. Note that all our experiments are conducted on a Linux machine with an Intel Xeon Golden processor and 32 GB memory, and FI experiments are conducted with 20-thread parallelism.



(a) Backprop (b) Kmeans (c) kNN

Fig. 4. Theoretical protection efficiency (blue line) vs measured protection efficiency (red line) in CPU machine learning programs

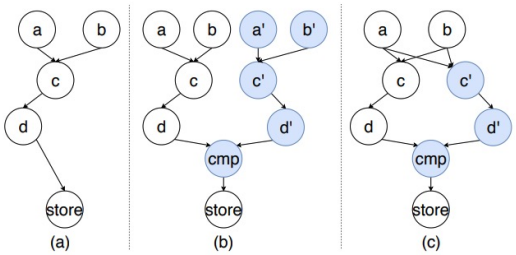


Fig. 5. An example of instruction duplication method. (a)Original program. (b)Full instruction duplication. (c)Selective instruction duplication.

4.1 Benchmarks

In this work, we select three machine learning programs from Rodinia benchmark suite[2]. The benchmark information is described in the following table. Among them, kNN and Kmeans are statistic machine learning applications, while Backprop contains the calculation of the back-propagation phase in a 2-layer fully connected neural networks. All of these three benchmarks contain both CPU & GPU version and are compiled into LLVM IR. However, the technique used in this work are not bounded to LLVM infrastructure.

Table 1. Characteristics of Benchmarks

Benchmark	Suite	Description	No. of Static Instructions
kNN	Rodinia	Find the k-nearest neighbours from an unstructured data set	349
Backprop	Rodinia	A machine-learning algorithm that trains the weights of connected nodes on a layered neural network	1860
Kmeans	Rodinia	A clustering algorithm used extensively in data-mining and elsewhere	1018

4.2 Fault Model

We use a fault model that has been commonly used in previous fault tolerance studies. Specifically, we only consider soft errors (i.e. hardware transient faults) in program computing units. Faults in memory or caches are not considered in this work, as we assume that error-correcting code (ECC) has already covered those faults. Furthermore, we ignore faults in the instruction's encoding as they can be detected through other means, such as error-correcting codes. Finally, we assume that the program does not jump to arbitrary, illegal addresses due to execution faults, as this problem can be mitigated by control flow checking techniques. However, the program may take a legal but wrong branch. That is, the execution path is legal, but the branch selection may be wrong due to propagated faults.

5 OUR RESEARCH DIRECTIONS

For a good comparison, we need to know error propagation characteristics in both CPU and GPU applications. In this section, we first estimate the error propagation in CPU programs. Using the initial study with CPU applications, we then shift our view to the GPU-accelerated ML applications to study the challenges to make them error resilient, and how we can mitigate the soft errors.

5.1 Estimation of Error Propagation in CPU Programs

We firstly present the initial study results of these benchmarks on CPU, to further understand the error propagation pattern and validate the efficiency of existing SID methods. In our fault model, the error propagation can result in SDC, so we use SDC coverage to evaluate the SID efficiency. The results are shown in Figure 4. For each benchmark, we perform SID based on the steps mentioned in Section 3. The blue line shows a theoretical SDC coverage under different protection overhead (the maximum dynamic cycle in knapsack total dynamic cycle), which is calculated on every instruction. At the same time, the red line shows a measured SDC coverage and is obtained by performing 1000 random FIs on selected input of each benchmark. Note that a larger number of random FI will produce a more accurate result. We use 1000 here as it gives a tolerable time cost and is in line with previous works. We observe that the measured protection efficiency is very close to the theoretical ones. In kNN and Kmeans, only protecting instructions within less than 5% overhead can already achieve a 100% protection efficiency. Those preliminary studies demonstrate the efficiency of SID methods.

5.2 Challenges to Make Programs Error Resilient in GPU

When it comes to GPU-accelerated programs, there are thousands of threads involved, which makes it really difficult to track error propagation using memory dependency. Moreover, due to copyright issues, NVIDIA SASS instruction sets are seldom explored. In GPU, program shares data across threads in a thread block using shared

memory. Also, program shares data across threads in a kernel using device global memory. Therefore, large amount of interleave dependency information is required, which in effect needs large amount of dynamic profiling. Again, extensive fault injections are needed to obtain a statistically significant resilience profile, which may be impractical because of the amount of time involved.

5.3 Implications on Error Mitigation

Applications are required to be error resilient before the deployment. Without strategically identifying the most vulnerable parts, the error sites go undetected. As a result, applications does not often meet resiliency target in GPU-accelerated ML applications. Again, it is implausible to track every error site of a GPU-accelerated ML application because of the need of large dynamic profiling, and interleave dependencies of thousands of threads.

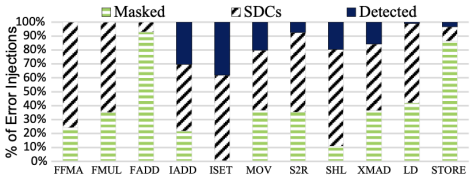


Fig. 6. The distribution of fault injection results of each instruction type

5.4 Proposed Error Mitigation

As discussion in section 3, instruction duplication is a very common technique that has been applied to CPU programs to make those error resilient. Unfortunately, this has been less studied for GPU-accelerated ML applications. Due to difficulty of finding each fault site, we rather apply instruction duplication technique to the only most vulnerable fault sites. So it is non-trivial to GPU-accelerated ML applications. In this way, we can protect majority percentage of error sites.

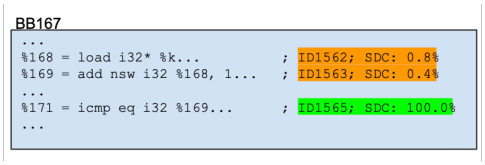


Fig. 7. SDC distribution via memory dependencies

We observe that certain instructions are more vulnerable to SDCs than others. For example, as shown in Figure 6, floating point instructions are more vulnerable than respective integer instructions. One of the reasons is floating-point instructions are generally used for calculation which silently affects the program's correctness and go undetected, while integer operations are generally used for control-flow decision and have severe effect when the error occurs, hence the error often goes detected. Therefore, we can cover most of the error sites if we choose to protect only those most vulnerable instructions by selective instruction duplication technique.

Moreover, we observe that instructions follow same error distribution via memory dependency. As can be seen from Figure 7, instruction ID 1563 has the same SDC rate as instruction ID 1562. With this observation, we can add extra set of instructions which are directly dependant on the instruction set previously protected by our selective instruction duplication technique. As a result, we can efficiently protect most the vulnerable fault sites by our combining selective instruction duplication technique and observation of memory dependency.

6 FUTURE WORKS

In this section, we list the future works we would like to explore. During this paper reading course, we read several papers of which topics are on accelerating machine learning systems via fancy parallelism strategies[4, 5]. Combining with the system resilience, we found no one has ever explored soft errors effect and mitigation on large machine learning systems. As the processing unit in GPU continues to become scaling and the machine learning systems are scaling larger, soft errors are very likely to happen more frequently than before. Under such circumstances, if there are some comprehensive initial studies to validate the severity of this issue, soft error on machine learning systems could be a very exiting topic to study!

REFERENCES

[1] Chun-Kai Chang, Guanpeng Li, and Mattan Erez. 2019. Evaluating Compiler IR-Level Selective Instruction Duplication with Realistic Hardware Errors. In *9th IEEE/ACM Workshop on Fault Tolerance for HPC at eXtreme Scale, FTXS@SC 2019, Denver, CO, USA, November 22, 2019*. IEEE, 41–49. https://doi.org/10.1109/FTXS49593.2019.00010

[2] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009, October 4-6, 2009, Austin, TX, USA*. IEEE Computer Society, 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[3] Fernando Fernandes dos Santos, Lucas Draghetti, Lucas Weigel, Luigi Carro, Philippe O. A. Navaux, and Paolo Rech. 2017. Evaluation and Mitigation of Soft-Errors in Neural Network-Based Object Detection in Three GPU Architectures. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2017, Denver, CO, USA, June 26-29, 2017*. IEEE Computer Society, 169–176. https://doi.org/10.1109/DSN-W.2017.47

[4] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. 2019. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2019, Washington, DC, USA, February 16-20, 2019*, Jeffrey K. Hollingsworth and Idit Keidar (Eds.). ACM, 300–314. https://doi.org/10.1145/3293883.3295712

[5] Peng Jiang, Changwan Hong, and Gagan Agrawal. 2020. A novel data transformation and execution strategy for accelerating sparse matrix multiplication on GPUs. In *PPoPP '20: 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, California, USA, February 22-26, 2020*, Rajiv Gupta and Xipeng Shen (Eds.). ACM, 376–388. https://doi.org/10.1145/3332466.3374546

[6] Guanpeng Li, Karthik Pattabiraman, Chen-Yong Cher, and Pradip Bose. 2016. Understanding error propagation in GPGPU applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, John West and Cherri M. Pancake (Eds.). IEEE Computer Society, 240–251. https://doi.org/10.1109/SC.2016.20

[7] Hengshan Yue, Xiaohui Wei, Guangli Li, Jianpeng Zhao, Nan Jiang, and Jingweijia Tan. 2021. G-SEPM: building an accurate and efficient soft error prediction model for GPGPUs. In *SC '21: The International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, Missouri, USA, November 14 - 19, 2021*, Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin (Eds.). ACM, 54:1–54:15. https://doi.org/10.1145/3458817.3476170