Analysis of faults and errors moderation for deep learning operators executed on GPUs

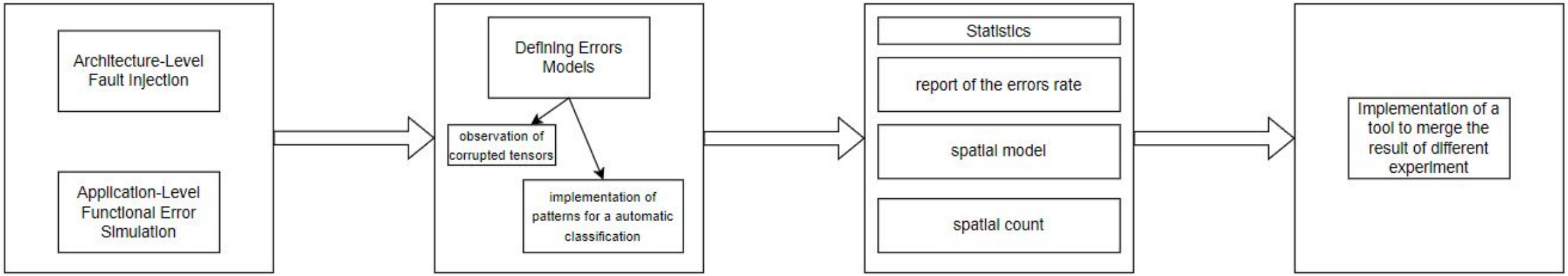
PRESENTED BY BALZARINI FILIPPO AND BETTIATI MATTEO

Introduction

In those years we are facing a growing interest in employing convolutional Neural Networks (CNNs) for perception functionalities in a wide range of application domains, including safety- and mission-critical ones. In this context the CNNs are generally executed on Graphic Processing Units (GPUs), because of its Single Instruction Multiple Data (SIMD) architecture, capable of speeding up the highly data-parallel elaborations that characterize these applications.

The core of the project is to analyze how the CNNs reacts to faults injections in order to answer the question "is the downstream system able to correctly carry out its task with the produced, possibly corrupted, output?".
The injection analyzed in this project has been generated at the architecture level. Then, the corrupted tensors generated by the CNN have been analyzed and classified, in order to predict the result of a soft error.

Working Process



Fault injection

We consider the Single Event Upset (SEU) fault model, whose effect is a bit-flip in a value stored in a register of the computing platform. Because the faults are rare events, the assumption of single faults holds very well.
Faults may cause the application:
• to crash or raise exception from the operating system that blocks the execution
• to hang leading to a non-termination
• to terminate by product an erroneous result, without any alert, this is called Silent Data Corruption (SDC)

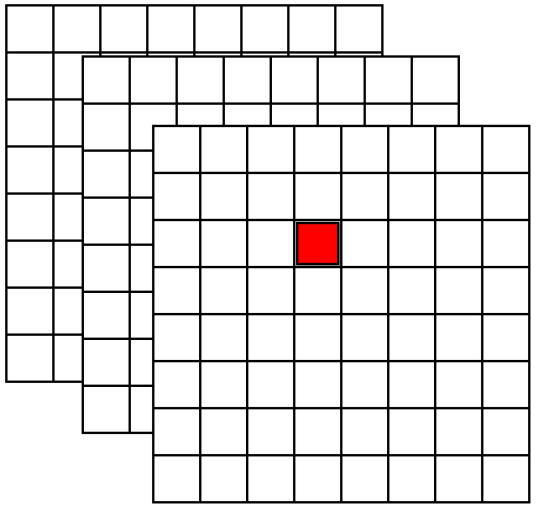
The fault injections have been made in several ways, both for instruction injection or simpler value injections,
such as:
• **fp**: floating point instruction
• **gp**: general purpose extraction
• **ld**: load instruction
• **rv**: injection random value
• **fsb**: injection flip single bit

Define Error Models

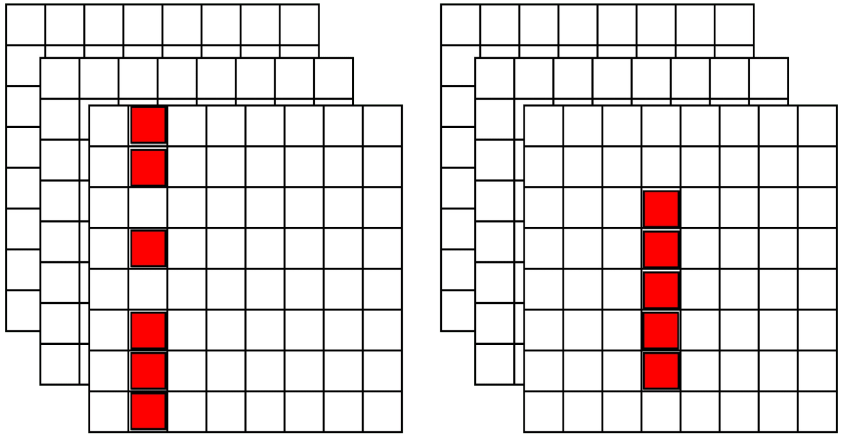
Single Feature Map:

Most of the faults that cause less than 16 erroneous values in the output tensor affect only a single feature map. This is due to the fact that the GPU kernel is implemented to group threads working on the same feature map in a single block; thus such multiple errors may be due to the corruption of predicated instructions. In this subfamily of spatial distributions we identify:

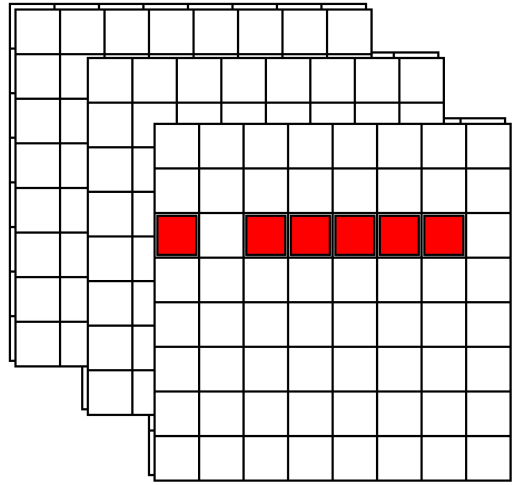
- **Single point**: a single value is corrupted.
- **Same row**: multiple corrupted values lie in the same row.
- **Same column**: multiple corrupted values in the same column.
- **Random**: no regular pattern.



Single Point



Same Column



Same Row

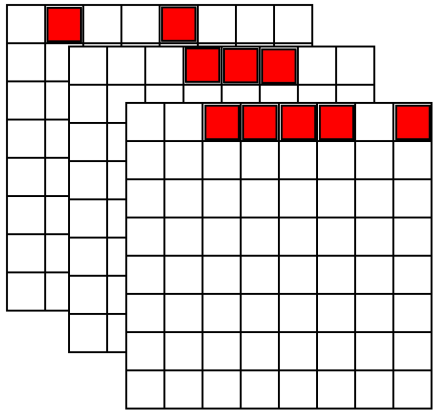
Single Feature Map

Define Error Models

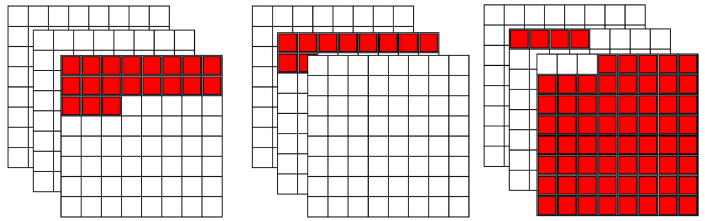
Multiple Feature Map:

Most of the faults that cause more than 16 erroneous values in the output tensor spread the corrupted values among several feature maps; this is due to the fact that, as mentioned, such operators heavily exploit shared memories. In this subfamily of spatial distributions we identify:

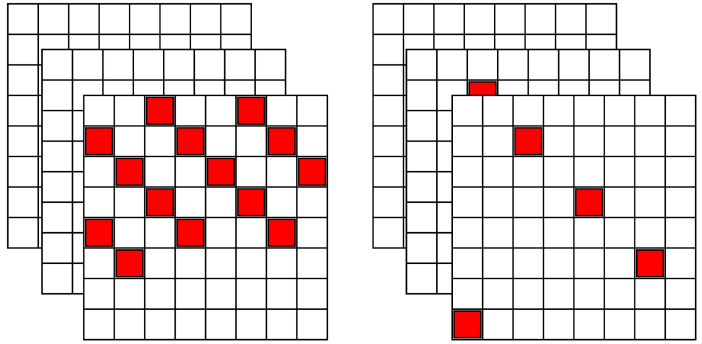
- **Bullet Wake**: the same location is corrupted in all (or in multiple) feature maps.
- **Shattered glass**: like one or more Bullet wake errors, but in one or multiple feature maps the corruption spreads over a row (or part of the row).
- **Consecutive**: serial values corrupted, without correct values in between. Corrupted values can occupy more than a rows.
- **SkipX** : the distance between corrupted values is always composed by X correct values. The X value it's the number of correct tensors between the first and the second corrupted values. The distance between the first corrected values and the first corrupted it's negligible.
- **Random**: no regular pattern.



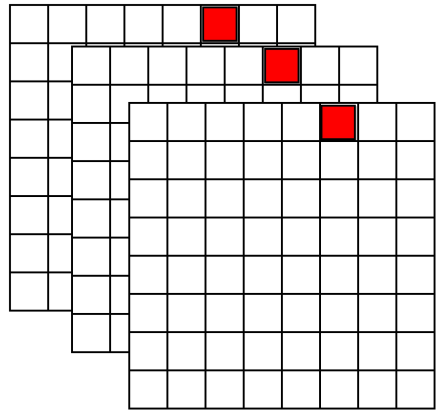
Shattered glass



Consecutive Error



Skip X

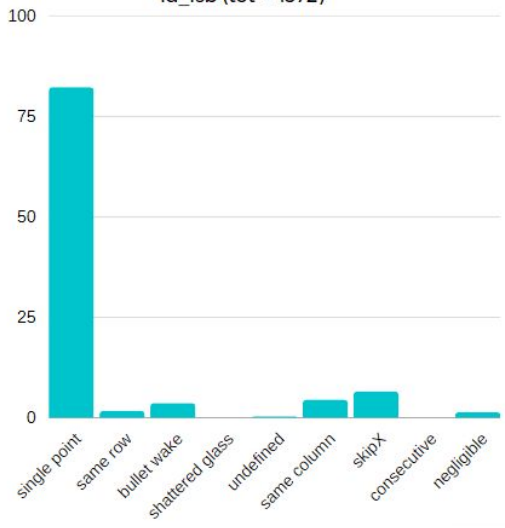


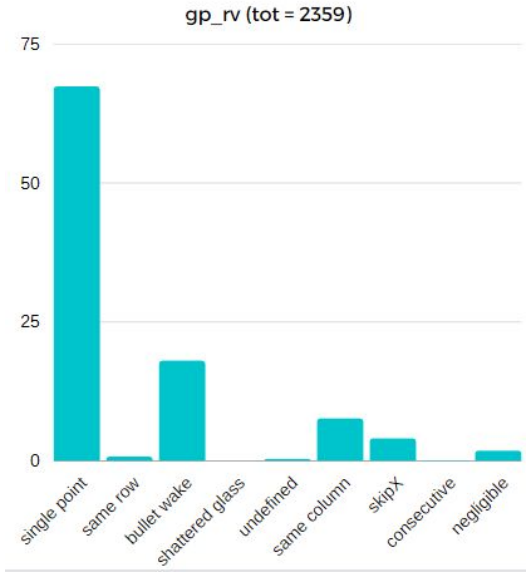
Bullet wake

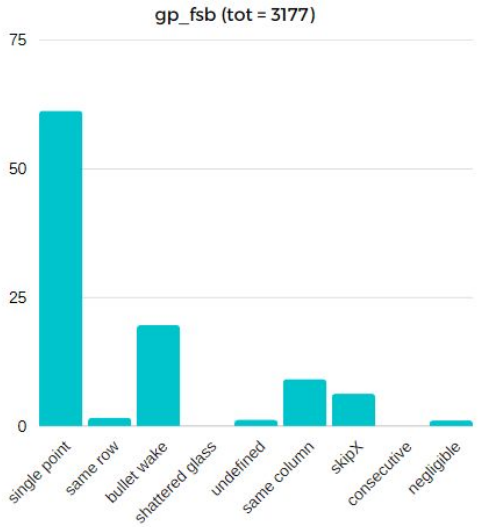
Multiple Feature Map

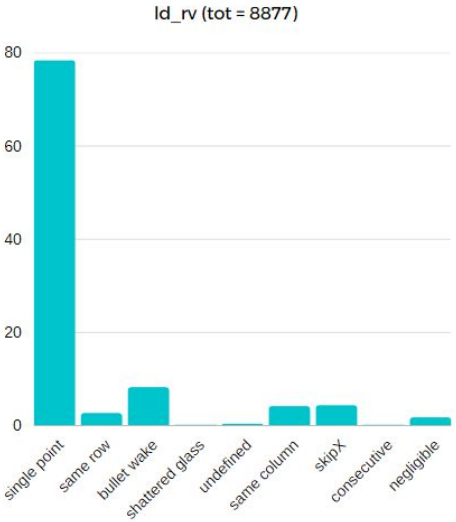
Statistics

The script also collect the load of data in order to create some statistics file and report the result of the injection on the corrupted tensors.
Precisely it generates three different type of files.
The first one it’s a csv file and it’s generated by a small script we made called "writeData" which you can find in the repository. The file created contains the result of an entire experiment. All the data are separated depending on the injection.
Down here you can find all the results of the experiment sorted by type of injection.


ld_fsb (tot = 4372)
100
75
50
25
0
single point
same row
bullet wake
shattered glass
undefined
same column
skipX
consecutive
negligible


gp_rv (tot = 2359)
75
50
25
0
single point
same row
bullet wake
shattered glass
undefined
same column
skipX
consecutive
negligible

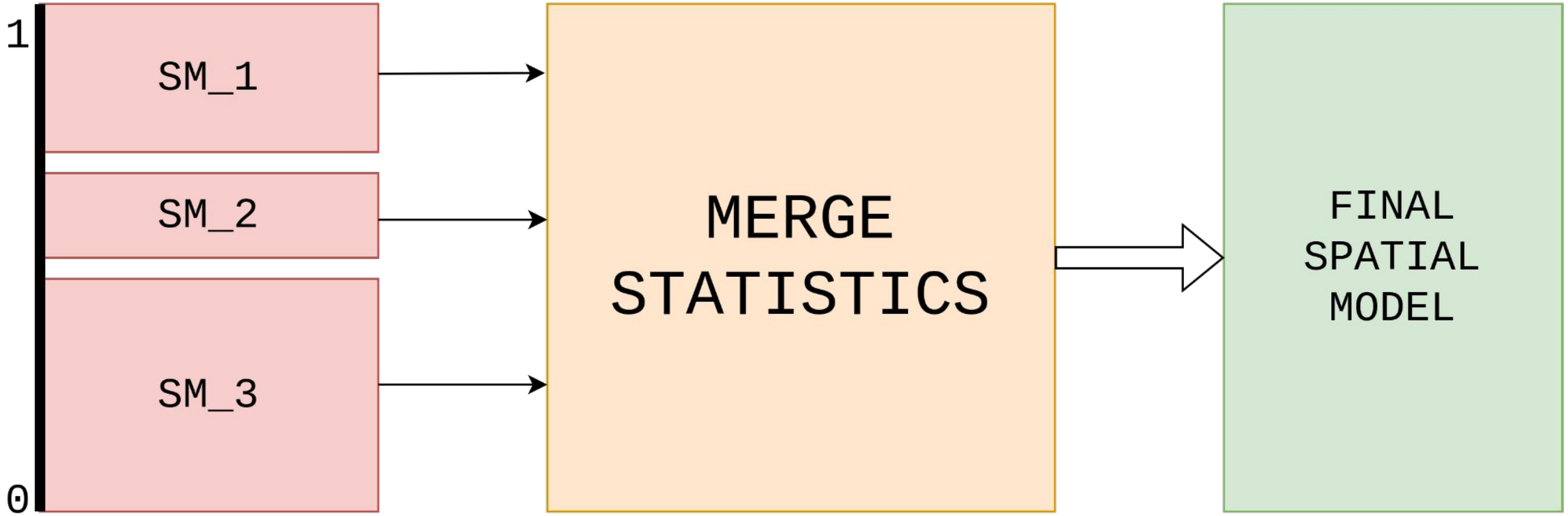

gp_fsb (tot = 3177)
75
50
25
0
single point
same row
bullet wake
shattered glass
undefined
same column
skipX
consecutive
negligible


ld_rv (tot = 8877)
80
60
40
20
0
single point
same row
bullet wake
shattered glass
undefined
same column
skipX
consecutive
negligible

Merge statistics

As already mentioned, the automatic classification of an experiment creates a spatial and a count statistics model. Merge Statistics is a script able to merge the results of spatial error models given a set of spatial models and their weights. Using merge statistics, different results can be melded together in order to obtain a single spatial model that describes the correct distribution of the error classes in the experiments. The script is useful for stick together different experiment's results without losing information. Weights should be chosen, for example, based on the dimension of the experiments, given two experiment's results, if the first experiment collected data from 200 corrupted tensors, and a second experiment, that collected only 100 corrupted tensors, it would be useful to assign 0,67 (200/300) to the first experiment and 0.33 to the second.

Merge statistics



Conclusion

The project's greatest strength is that thanks to the analysis made up here, we can predict with a close approximation which damage can a hardware error inflict on the software.

The idea behind the project is to analyze a large amount of data in order to be capable to prevent soft errors or at least know the damage they could create.

The script also confirmed all data provided in the first version of classes and thanks to the Merge Statistics tool, the use of classes is now more fluid. Future work will bring further optimization to the script that classified errors, bringing a completely autonomous algorithm for the classification of errors that doesn't just classify errors with known patterns, but will be also able to discover new patterns itself.