ERROR-DETECTING RLIRFO DATA STRUCTURES FOR THE WIN

Darío de la Fuente García **Félix Áxel Gimeno Gil** **Juan Carlos Morales Vega**

March 10, 2019

ABSTRACT

Improvements in the speed and capacity of DRAM have made memory corruption by cosmic rays more likely. In order to ameliorate this issue, we present a recursive stack (Recursive Last In Recursive First Out) that can verify its state by storing all its previous states and the previous states of the previous states up to a desired level of recursion.

***K*eywords** Error detection · Data structures · Cosmic rays · Stacks · Stacks of stacks · Stacks of stacks of stacks · Template Metaprogramming

1 Introduction

13.8 billion years ago, an event that was key for the existence of Sigbovik [1] and your reality [2] itself happened. Yes, we are talking here about the Big Bang, the creation of our universe [3]. Due to several reasons (such as your own existence), the creation of the universe cannot be considered a bad move, despite being different opinions in that topic [4]. However, the universe that was created is nowadays a source of errors in our electronic devices.

In the past, errors due to cosmic rays did not happen since the size of the chips and the transistors were much bigger than the size of a particle, hence a collision was not able to change the state of a bit in memory. However, with the miniaturization of those components, the energy of a cosmic ray has become able to change the state of a bit, causing these types of soft errors to appear [5][6].

To be a bit more precise, the effect of a cosmic ray at the Earth surface is not due to the cosmic ray itself, but rather due to the effect of the subproducts of the ray after interacting with the atmosphere, mainly protons, neutrons and nuclei. Neutron showers have proven to be the main cause of errors due to cosmic rays [7][8].

2 Motivation

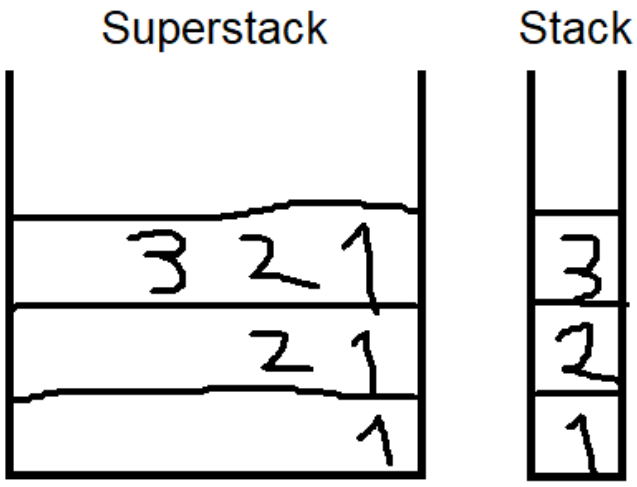
As shown in [9][10][11], the probability of getting an uncorrectable error is quite low, but that is for a small system and the calculations were made back in 2010, when we had 32-45 nm architectures. Nowadays, having 7 nm architectures in some devices, the problem is considerably worse. Of course, the problem is still going to become even worse in the future with more advanced architectures.

Since we cannot go back in time and prevent the universe from being created (as of now), we need an alternative.

Stacks [12][13] (not to be confused with stacks [14] and stack [15]) are one of the most important [16] data structures [17].

3 Idea

Now that we have described the problem, let us explain how to solve it using recursive stacks. The basic idea behind the method is to store all previous states of the stack to be able to search for errors. These states are stored in another stack that we can call the "superstack". The following superb drawing helps for understanding the structure:



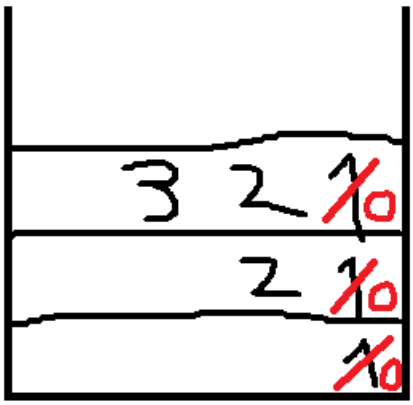
Now, it is easy to check for errors in that structure:

In case there has been a push operation, the first stack of the superstack performs a pop operation and it is compared with the second stack of the superstack. If they are the same, we assume that no error has happened.

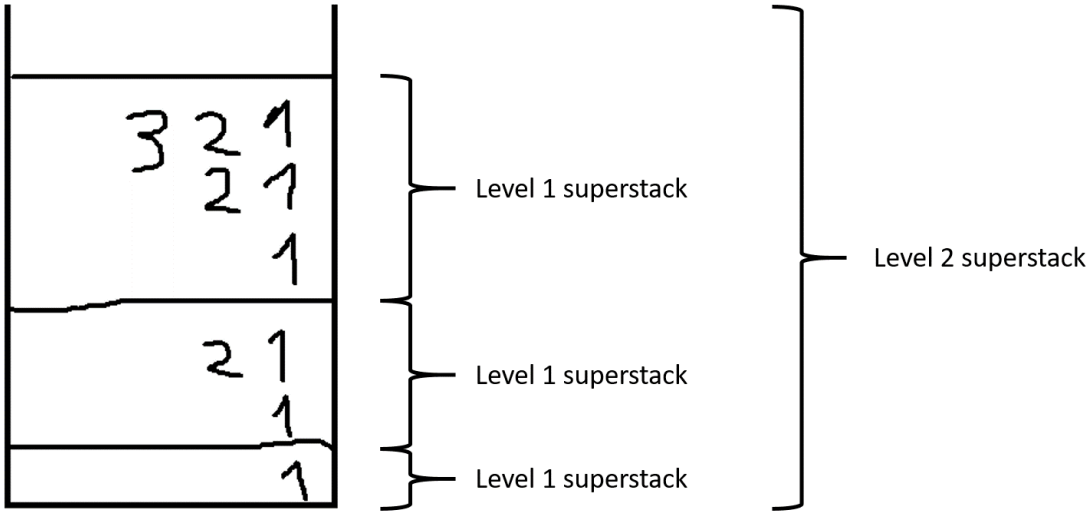
In case of a pop operation, the previous value is added to the top stack in the superstack and then compared with the second one.

As we can see, the push check is straightforward, but the pop check presents a small issue: the top stack does not know which is the popped value (and we cannot just copy the top value of the second top stack). To solve this, we propose a structure pair<stack<T>, optional<T>> for the entries of the superstack, instead of just stack<T>. In the case of a push operation, the optional will take the nullopt value, while in the case of a pop operation, it will take the value of the popped value. This structure also helps to detect if the performed operation was a push or pop very straightforwardly. Moreover, the decision of using an optional here can be considered as an act of good will to promote the features of C++ 17. Cool!

This process is repeated through the full stack until every stack is compared with the previous one. So far so good, but, can we be completely sure that there have been no errors? Not really. What if several evil neutrons decide to team up and corrupt the memory in the specific, following way?

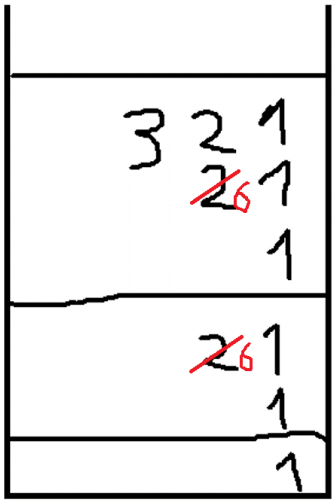


Now we do not have any way of detecting that error. Except that we do have. We can consider the previous superstack as an entry of a bigger superstack. In this way, if an army of evil neutrons decides to perform an all-out-attack over a level 1 superstack, the errors can still be detected in the new level 2 superstack. The structure looks like this:



Checking the errors of a superstack of superstacks is even easier than the previous case. Since a superstack already contains all previous states, there is no need to keep an optional value and the cases for push and pop works exactly in the same way: a pop operation is performed over the top level 1 superstack of the level 2 superstack and the result is compared with the second level 1 superstack.

However, we still need to perform the level 1 superstack check over the top level 1 superstack since there can be errors that cannot be detected at level 2 but they can at level 1:

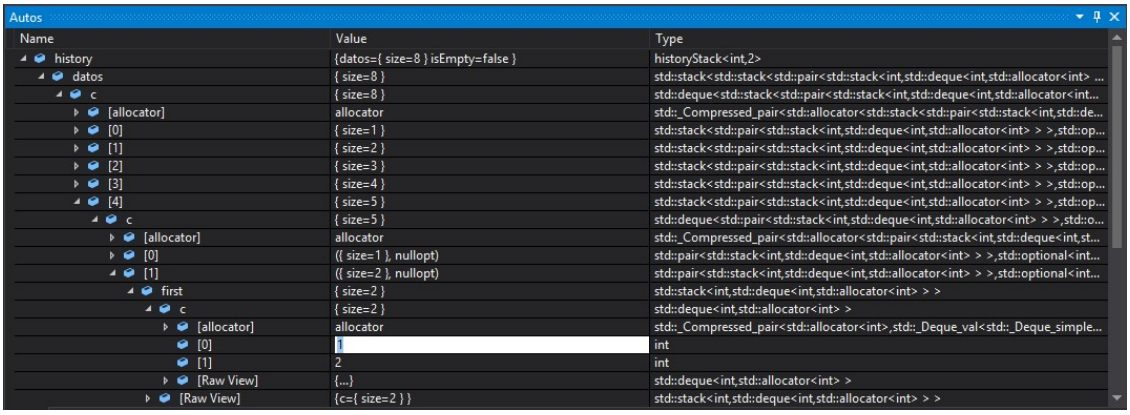


This is starting to look strong, but what if we want to keep some information for longer than what it takes a solar mass black hole to fully evaporate via Hawking radiation? [20] Yes, you know where this is going. In general, we define a level n superstack as a stack of level n-1 superstacks. Detecting an error in a level n superstack is done as explained before: first check the level n superstack and then call the level n-1 checker over the top level n-1 superstack recursively. The inception level n can be freely chosen and the structure is generated using a bit of ~~magic~~ template metaprogramming, as you can see in the code [22].

One could argue that the more inception, the more it will increase the chances of an error happening, but since statistics is a lie, nobody can really prove it.

4 Methodology

We have modeled the cosmic rays using a model that we call "Hand of God". This advanced method consists on using the debug functionalities of the compiler of your choice (Visual Studio 2017 in our case). We set a breakpoint and change random values by hand.



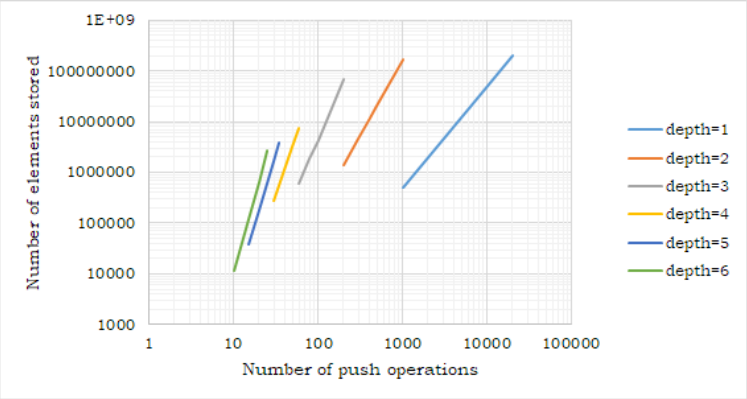
It worked in every case.

5 Benchmarks

We prepared a benchmark that initialised a stack of depth k and performed on it n push operations. The results can be found in the next graph:

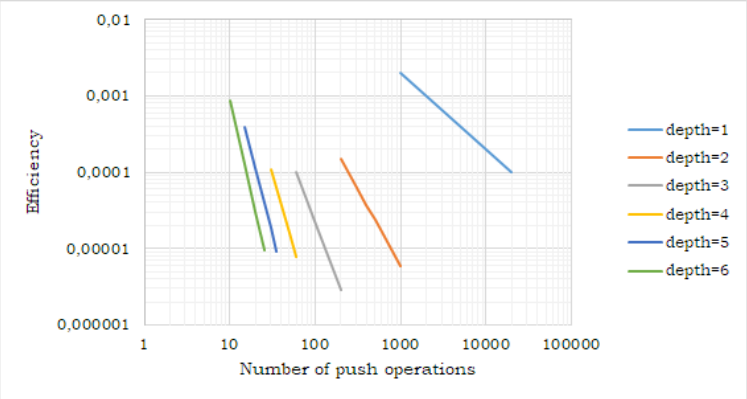
The initial idea was to make longer benchmarks, unfortunately when attempting to do so the virtual machine in which the code was running ran out of memory, thrashing started to happen and the virtual machine had to be force shut down. For that reason, none of the benchmarks go over 3 seconds. Since this is a log-log graph, by looking at the slope of the lines we can deduce easily that the time complexity of performing n operations on a recursive stack of depth k is $\theta(n^{k+1})$. Since this is polynomial time, we can conclude that ~~a mathematician in the wrong field~~ computer scientists will think this is efficient.

In order to better understand this phenomenon, we computed the actual number of elements these benchmarks were producing:



As we can see, at least for levels of recursion of 1, 2 and 3, it is no surprise that the code will explode a virtual machine with 4 GB of RAM assigned to it. As for levels 4, 5 and 6, we suspect that even though the number of elements stored is smaller, there are a lot of smaller level stacks that are being memory aligned and wasting a lot of RAM this way.

By dividing the number of operations by the number of elements generated, we can compute the efficiency:



This result may look absolutely terrible for the untrained eye. However, one must remember that this is absolutely necessary to protect against the evil army of cosmic rays that are out to get you and the correctness of your programs.

Special thanks to all the virtual machines that had to be force shut down. Your sacrifice shall not be forgotten.

6 Potential moral implications for saving erased data

Nobody cares.

7 Future work

Obvious future work includes implementing RLIRFO in languages such as Brainfuck, Unsafe Rust, Safe Rust, Piet and Malbolge. Another imp ovement would be torfork an open source C compiler to use RLIRFO data structure instead of LIFO in call stacks.

Also, in this paper we have talked about data structures that contain its past state. With a similar implementation (probably) we could create data structures that contain themselves, and with that we could create a paradoxical data structure [21] and crash the universe.

8 Conclusion

We have discussed a chaotic-good algorithm to detect errors in a stack that are caused by cosmic rays or other highly probable causes. The data structure that we have discussed can be checked in github. The error detecting algorithm

was tested using the "Hand of God" method, yielding the expected results. We have also evaluated the performance of the algorithm and concluded that computer scientists won't find anything wrong with it. We expect this amazing data structure to be used in the future in almost every device to avoid memory errors in a chaotic way.

Neutrons will not win this crusade.

References

[1] http://sigbovik.org

[2] Your Reality https://www.youtube.com/watch?v=CAL4WMpBNs0

[3] Big Bang https://en.wikipedia.org/wiki/Big_Bang

[4] https://en.wikiquote.org/wiki/The_Hitchhiker's_Guide_to_the_Galaxy#Chapter_1

[5] Attack of the Cosmic Rays! https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2

[6] Serious Computer Glitches Can Be Caused By Cosmic Rays https://science.slashdot.org/story/17/02/19/2330251/serious-computer-glitches-can-be-caused-by-cosmic-rays

[7] Soft Errors https://en.wikipedia.org/wiki/Soft_error

[8] Single Event Upset https://en.wikipedia.org/wiki/Single_event_upset

[9] Do gamma rays from the sun really flip bits every once in a while? https://stackoverflow.com/a/4109288

[10] http://lambda-diode.com/opinion/ecc-memory

[11] T.J. O'Gorman The effect of cosmic rays on the soft error rate of a DRAM at ground level https://doi.org/10.1109/16.278509

[12] Stack https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

[13] Stack https://en.wikipedia.org/wiki/Stack_(C%2B%2B)

[14] Stack https://en.wikipedia.org/wiki/Stack_(mathematics)

[15] Learn to Stack https://youtu.be/Xqpf1FxU2q4?t=291

[16] What Is a Full Stack Developer https://www.youtube.com/watch?v=UtDpYVf9jKU

[17] List of data structures https://en.wikipedia.org/wiki/List_of_data_structures

[18] Double Byte Error Detecting Codes for Memory Systems https://doi.ieeecomputersociety.org/10.1109/TC.1982.1676056

[19] Population dynamics https://wiki.puella-magi.net/Population_dynamics

[20] https://www.quora.com/How-long-would-it-take-for-an-Earth-mass-black-hole-to-evaporate-due-to-Hawking-radiation

[21] https://en.wikipedia.org/wiki/Russell%27s_paradox

[22] RLIRFO https://github.com/juancarlosmv/RLIRFO