Correcting Detectable Uncorrectable Errors in Memory

Grzegorz Pawelczak
University of Bristol
Department of Computer Science
Bristol, UK
g.pawelczak@bristol.ac.uk

Simon McIntosh-Smith
University of Bristol
Department of Computer Science
Bristol, UK
s.mcintosh-smith@bristol.ac.uk

1 INTRODUCTION

With the expected decrease in Mean Time Between Failures (MTBF), Fault Tolerance (FT) has been identified as one of the major challenges for exascale computing. One source of faults are soft errors caused by cosmic rays, which can cause bit corruptions to the data held in memory. Current solutions for protection against these errors include Error Correcting Codes (ECC), which can detect and/or correct these errors. When an error that can be detected but not corrected occurs, a Detectable Uncorrectable Error (DUE) results, and unless checkpoint-restart is used, the system will usually fail. In our work we present a probabilistic method of correcting DUEs which occur in the part of the memory where the program instructions are stored. We devise a correction technique for DUEs for the ARM A64 instruction set which combines extended Hamming code with Cyclic Redundancy Check (CRC) code to provide near 100% Successful Correction Rate (SCR) of DUEs.

2 PREVIOUS WORK & METHODOLOGY

In their previous work, Gottscho et al. [1] proposed *Software-Defined Error-Correcting Codes* (SWD-ECC) which are capable of correcting DUEs by leveraging the properties of the underlying ECC and the *side information* about the data stored. In their research they have focused their efforts on the (39,32) Single Error Correction and Double Error Detection (SECDED) code, where two bit-flips inside a codeword cause a DUE. As Figure 1 shows, when a DUE occurs there can be multiple candidate codewords which could be the correct codeword, and since all the candidate codewords are equally likely, it is impossible to determine which is the correct one. However, by using the side information about the data stored, it is possible to make some of these codewords more likely than others. The original work provides an example where the data protected is the MIPS program instructions, and so all the valid codewords that do not represent a valid instruction can be discarded. This allowed them to reduce the search space of valid instructions to an average of 12, and along with an observation that some instructions are more likely than others, they were able to correct 34% of DUEs.

We extend this work by investigating how well this approach performs with the ARM A64 instruction set, where every instruction is 32-bits, and so we also use the (39,32) SECDED. We also introduce further heuristic techniques to reduce the number of candidate instructions that represent a valid instruction, improving the SCR.

The first technique we investigate is adding an Error Detecting Code (EDC) in order to reduce the number of candidate valid codewords. We use CRC32C as the EDC, where every N SECDED codewords are covered by a single CRC32C checksum (see Figure 2).

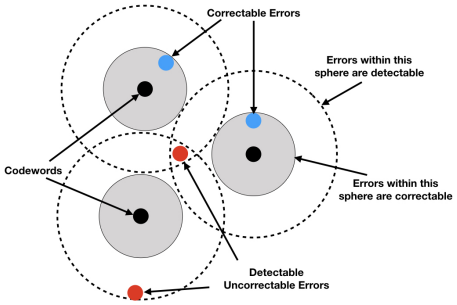

Figure 1: ECC allows for correction as there is no overlap of the correctable spheres between the codewords. However when a DUE occurs, these spheres overlap, meaning it is impossible to determine the original codeword.

With this approach, when a DUE occurs within a codeword we can find all valid SECDED codewords that also produce a valid CRC32C checksum. We use CRC32C due to its high Hamming Distance (HD) length [2] and because modern architectures, such as x86 or AMRv8a, often provide hardware support for the computation of the checksum via intrinsics.

Another technique we investigate is filtering out codewords which are valid instructions, which would however be illegal operations. For example, consider an instruction which requires a different privilege level, or a branch instruction to an invalid address; both are valid but illegal instructions, and therefore should not be considered as likely candidates. Determining whether an instruction is legal is a challenging task and requires a thorough knowledge about the instruction set. In some cases it might not be possible to decide whether an instruction is legal without simulating the program. Our current approach is limited to only checking that all operands which contain an address represent a valid address, which accounts for 12% of instructions in our test data.

The Figure 3 shows the heuristic approach that is used when accessing an instruction. In order to determine whether a codeword represents a valid instruction we use VIXL[1], a Runtime Code Generation Library by ARM which provides a disassembler.

To evaluate the performance of our techniques, the binary programs from the `/bin/` directory for the openSUSE Linux distribution (120 programs with 168 unique opcodes and a total of 3268027 instructions) are used as the training programs in order to find the most frequent instructions. We use the mini-apps from the UK Mini-App Consortium[2] as the test programs (7 programs with 118 unique opcodes and a total of 515666 instructions). We inject DUE faults

SC'17, November 12–17, 2017, Denver, Colorado

[1] VIXL: AArch64 Runtime Code Generation Library, https://github.com/armvixl/vixl
[2] UK Mini-App Consortium, https://uk-mac.github.io/

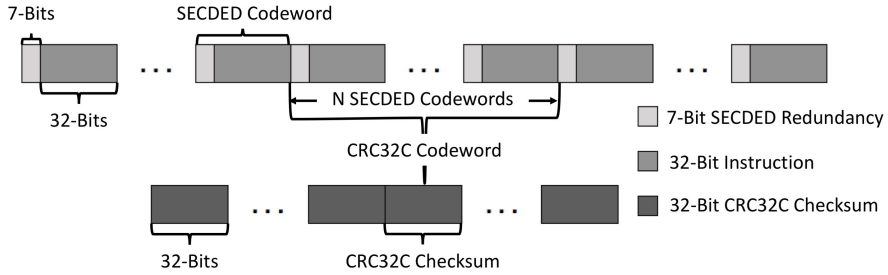

Figure 2: For every N 39-bit SECDED codewords, a 32-bit CRC32C checksum is stored.

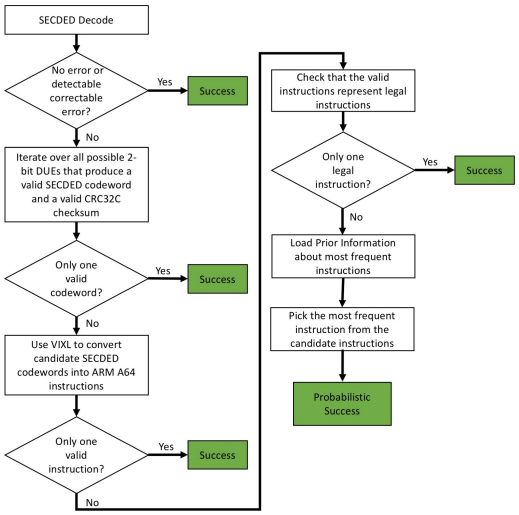

Figure 3: This flowchart shows the heuristic approach that is used when accessing an instruction from memory.

by selecting a random instruction from the `.text` section of the test program, and randomly flipping two unique bits in the instruction SECDED codeword.

3 RESULTS & CONCLUSIONS

The results in Table 1 show that we can successfully correct DUEs occurring in the instructions with a high success rate. The performance of the original method on the ARM A64 instruction set has improved compared to the MIPS instruction set. The number of valid codewords is the same for each faulty codeword because SECDED is a linear code, however on average when taking instruction validity into account, the number of candidate codewords is lower for ARM A64, but this is dependent on each individual codeword. The additional test for instruction legality has further reduced the number of candidate instructions and thus improved the SCR.

The hybrid scheme of SECDED ECC and CRC32C EDC has vastly improved the SCR, however it is dependent on the number of SECDED codewords per CRC Codeword. When $N \leq 134$, this approach can always correct a SECDED DUE, because the HD of CRC32C at that codeword length is 6 therefore it can always detect an error of up to 5 bits. The candidate SECDED codewords have

Table 1: Performance results for the new techniques.

Correction Method	Average Candidate Instructions	Successful Correction Rate
Original Gottscho et al. [1] method	7.96	43.4%
Testing for legal instructions	7.16	48.1%
Error Detection with CRC32C (N $\leq$ 134)	1.00	100.0%
Error Detection with CRC32C (134 < N)	$\approx$ 1.00	$\approx$ 100.0%

an error of distance of either 0 (for the correct codeword) or 4 (for the incorrect codewords) from the correct codeword and so CRC will always be able to detect the incorrect codewords, leaving only the correct codeword as a candidate. If $134 < N$, then the HD is 4, meaning that there is a small chance of multiple candidate codewords occurring (the estimated probability is $2^{32-1} \approx 4.6 \times 10^{-10}$ [3]). This probability is likely to be even smaller in reality, because a collision in CRC checksums must also take into account that it must represent N valid instructions.

Addition of the CRC32C EDC results in no runtime overhead because the checksum is only accessed in an event of a DUE. In addition, the checksums do not need to be recalculated during the execution of the program as the instructions should not change. If this approach was used for data that does change, the extra layer of EDC could add significant performance overhead, as Read-Modify-Writes (RMW) would need to be performed to update the checksums. In order to improve the performance of RMWs, a smaller N should be used. However, this would increase the storage overheads. For example, the storage overheads for $N = 2$, $N = 134$ and $N = 2048$ are 41.03%, 0.6% and 0.04% respectively.

In this work we have only used the 39-bit SECDED, and future work would involve investigating the performance of other ECC techniques such as 72-bit SECDED or Reed-Solomon Codes. Currently when there are multiple candidate codewords, we pick the candidate codeword which represents the most frequent instruction from the training set. Future work would involve investigating other approaches, such as machine learning, in order to make decisions about which instruction features to take into account. We also plan to extend this work to other data types, not just program instructions.

REFERENCES

[1] Mark Gottscho, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. 2016. Software-Defined Error-Correcting Codes. *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2016* (2016), 276–282. https://doi.org/10.1109/DSN-W.2016.67

[2] Philip Koopman. 2002. 32Bitt Cyclic Redundancy Codes for Internet Applications. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN '02)*. Washington, DC, USA, 459–472. http://dl.acm.org/citation.cfm?id=647883.738239

[3] B Hall P Koopman, K Driscoll. 2015. *Selection of Cyclic Redundancy Code and Checksum Algorithms to Ensure Critical Data Integrity*. Technical Report. U.S. Department of Transportation, Federal Aviation Administration.