Bare-Metal Redundant Multi-Threading on Multicore SoCs under Neutron Irradiation

A. Serrano-Cases[1], A. Martínez-Álvarez[2], R. Possamai Bastos[3], and S. Cuenca-Asensi[2]

***Abstract*—A software technique is presented to protect commercial multi-core microprocessors against radiation-induced soft errors. Important time overheads associated with conventional software redundancy techniques limit the feasibility of advanced critical electronic systems. In our approach, redundant bare-metal threads are used, so that critical computation is distributed over the different micro-processor cores. In doing so, software redundancy can be applied to Commercial Off-The-Shelf (COTS) micro-processors without incurring high-performance penalties. The proposed technique was evaluated using a low-cost single board computer (Raspberry Pi 4) under neutron irradiation. The results showed that the Redundant Multi-Threading versions detected and recovered all the Silent Data Corruption (SDC) events, and only increased HANG sensitivity with respect to the unhardened original versions. In addition, higher Mean Work to Failure (MWTF) estimations are achieved with our bare-metal technique than with the state-of-the-art bare-metal software-based techniques that only implement temporal redundancy.**

***Index Terms*—COTS, Neutron Radiation, Triple Modular Redundancy, Redundant Multi-Threading, Single-Board-Computer, System on Chip.**

I. Introduction

Today, an increasing number of industrial domains are adopting Commercial Off-The-Shelf (COTS) processors, to implement critical electronic systems. The application of these systems ranges from mainstream aerospace and military sectors to emergent markets, such as high-performance computing, autonomous vehicles, and medical devices. The inclusion of the new COTS processors within those systems are due to their improved flexibility and performance and their reduced costs. Compared to (rad-hard) hardened processors, commercial devices offer potentially significant performance-related advantages, due to their inherent parallel computing structures. In addition, semiconductor process innovations have considerably reduced energy consumption. Unfortunately, even though these alternatives to other specifically designed circuits are promising, the manufacturing process of electronic components and higher operating frequencies makes them more vulnerable to radiation-induced faults. It is therefore critical to equip commercial processors with safety-critical capabilities that ensure fault-tolerance computing, which implies fault detection and computation-process recovery capabilities, even in the presence of faults.

The hardening of electronic devices has generally involved the use of spatial and/or temporal redundancy. Thus, if there are sufficient copies of the data, radiation-induced faults may be detected and/or corrected as the computation process is replicated. The protection techniques can be implemented at different layers of the electronic set up: hardware, software, or even hybrid combinations.

Redundancy can be introduced with hardware techniques that apply summary functions (CRC, DMC, EDAC,...) to verify data integrity or that replicate some components to create redundant hardware blocks. Focusing on the last group, the Dual and Triple redundant Core lockstep (DCLS/TCLS) [1], [2] techniques replicate the whole processor for system output comparisons every clock cycle. This strategy enables the detection of any mismatch during the execution of the code. Additionally, TCLS offers system recovery capabilities, using faultless cores to mask incorrect computation of the faulty core. Although both techniques present high fault coverage rates, several drawbacks may also be mentioned. The former (DCLS) introduces high performance and hardware overheads to maintain the computing context and to enable fault recovery capabilities (roll-back). In comparison, the latter (TCLS) requires an extra core and implies a conservative over-verification to keep the three replicated cores under synchronization. Unfortunately, these strategies imply customized software design and hardware that increase overall system development costs.

In contrast, techniques based on software modifications aim to execute protected software on unreliable hardware, primarily COTS devices. The software techniques to some extent mirror the hardware; software-structure and data replication at different software levels -programs, functions, loops, and instructions- ensure reliable computation. Although software modifications involve lower impacts on development costs than the hardware counterpart, their application, nevertheless, presents relevant overheads in terms of performance and resource usage.

In this investigation, a portable software-based technique is presented that improves multi-core COTS processors by enabling programmed soft-error fault tolerance capabilities. It is an extension and improvement of our previous bare-metal approaches [3], and [4], which follows a two-thread duplication scheme with comparison & re-execution (multi-

The research reported in this paper has been partially supported through the following projects: MultiRad (funded by Région Auvergne-Rhône-Alpes, France); IRT Nanoelec (French National Research Agency ANR-10-AIRT-05 project funded through the Program d'investissement d'avenir); UGA/LPSC/-GENESIS platform and PID2019-106455GB-C22 (funded by the Spanish Ministry of Science and Innovation).

[1]A. Serrano-Cases works at the Barcelona Supercomputing Center (BSC) (e-mail: alejandro.serrano@bsc.es).

[2]A. Martínez-Álvarez and S. Cuenca-Asensi are staff members at the Computer Technology Department, University of Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain (e-mail: amartinez@dtic.ua.es; sergio@dtic.ua.es).

[3]Rodrigo Possamai Bastos works at the Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, 38000 Grenoble, France (e-mail: rodrigo.bastos@univ-grenoble-alpes.fr).

thread DWC-R). The novelty of the current work resides in re-targeting and adapting them to COTS devices with a higher number of cores, which better matches the current reality of COTS. Moreover, this adoption was intended to improve some drawbacks of previous proposals, such as the need to preserve the execution context or the time overhead introduced by the recovery mechanism. The software is modified with the new technique, to reproduce redundancy, taking advantage of the available cores on the micro-processor to run multiple instruction flows (*i.e.*, threads). It can therefore detect and mask transient faults in memory cells, avoiding system failures, by comparing the replicated data and running majority voting on the outcome. If an abnormal and uncorrectable state is detected, the system is stopped, so that state will not be propagated. The system then awaits a reset signal to be triggered, prompting recovery to a fault-free state. Compared to other software-based solutions which implement a fine-grain Triple Modular Redundancy (TMR) [5], our technique operates at the function/loop level, reducing the injection of redundant code and avoiding any need for continuous verification. Thus, the probability of Silent Data Corruption (SDC) is significantly reduced. In addition, the use of redundant threads speeds up computation and lowers the time overheads associated with single-thread schemes [5], [6]. On the contrary, its sensitivity to potential issues related to thread synchronization is higher, which may risk increased hang events. Careful consideration must therefore be given to availability requirements when using this technique.

The implementation presented in this study was built on a low-cost computer, also known as a Single Board Computer (SBC). More precisely, the Raspberry Pi v4 (RPI4) was equipped with a state-of-the-art ARM Cortex-A72 processor (BCM2711) manufactured by Broadcom. The system was evaluated under radiation at the Subatomic Physics & Cosmology Laboratory (LPSC, Grenoble, France) using *mono-energetic neutrons*.

II. Related works

The ubiquity of modern multi-core processors has been consolidating the emergence of new multi-instruction flow computing paradigms. With respect to the aforementioned group of hardware techniques, which are based on DCLS/TCLS, modern shared memory multi-core devices can now efficiently perform multiple copy executions of the same instruction flows within separate units.

The technique known as Redundant Multi-Threading (RMT) was first proposed in [7], to leverage redundant computing resources, for detecting and recovering soft errors through simultaneous multi-threading. In this study, the concept of the Sphere of Replication (SoR) is also defined as the set of resources, hardware, or software, which are replicated. So the dynamics of the SoR imply that data inputs must be replicated and, in contrast, data outputs must be checked to ensure their integrity. Since then, the effects of soft errors have been addressed through different RMT approaches that involve compilers [8], [9], operating systems [10], and application-based techniques [11], [12]. All these approaches shared the evident drawback in performance/size overheads and susceptibility to errors, which was produced because of the additional software layers involved in the techniques. Evidence of higher susceptibility to errors which resulted in a higher rate of Operating System crashes and SDC events were presented in [13] and [11]. Those proposals were only tested on simulators with promising results, but were never assessed on real devices.

Automatic mitigation tools such as Trikaya [6] and COAST [5] generate hardened applications from high-level source code. Those tools are designed to protect bare-metal single-threaded code, by means of temporal redundancy (the code is repeatedly executed) and spatial redundancy (some data are replicated). On the one hand, the Duplication With Comparison (DWC) technique is applied with the Trikaya tool at a functional level. It consists of running the critical code twice on the same core and comparing the results; if any discrepancy is observed, a rollback with re-execution is performed for an eventual vote on the result. The use of just one execution thread penalizes performance and increases the fault-exposure time of sensitive data. In addition, the recovery time is higher with respect to a multi-threaded approach (re-execution vs voting mechanism). A DWC single-thread version of the benchmark was added to the radiation campaign, to evaluate the differences between them, as will be discussed in Section V. A multi-threaded version of this technique (*i.e.*, multi-thread DWC-R) was studied at different protection granularity levels in [3]. Furthermore, authors in [4] explored the combination of a two-thread DWC-R version with a custom IP for monitoring the control-flow. The technique produced a performance overhead of 2.5× and offered improvements of one order of magnitude in the cross-section of total errors.

On the other hand, the COAST tool injects redundant instructions and checkers at assembly level, according to the rules described in [14]. COAST has been used to apply TMR in different architectures [5] and benchmarks [15] following a single-thread execution scheme. The results showed that a percentage of SDCs passed undetected to the TMR. The performance overhead obtained was over 3× in the majority of the cases and different improvement levels were observed in the cross-section.

Hardware-based redundancy techniques are much less common than their software counterparts, because they usually need architectural modifications or custom module additions. In [16], two Field Programmable Gate Array (FPGA)-based Intellectual Property (IP)-cores were used to implement a lockstep, a checkpoint with rollback recovery, and on-demand configuration memory scrubbing, for the protection of periodic tasks running on two LEON processors. The overhead was significantly reduced with respect to state-of-the-art TMR-based techniques. Similarly, a rollback/recovery mechanism using the 2-core ARM cortex-A9 programmable resources of an FPGA was implemented in [2]. The test application was manually split into several blocks, running simultaneously on both cores. Each block was delimited by means of a verification point, which when triggered, activated customized hardware, so that program context and data were saved in a dual-port private memory where the data could be compared

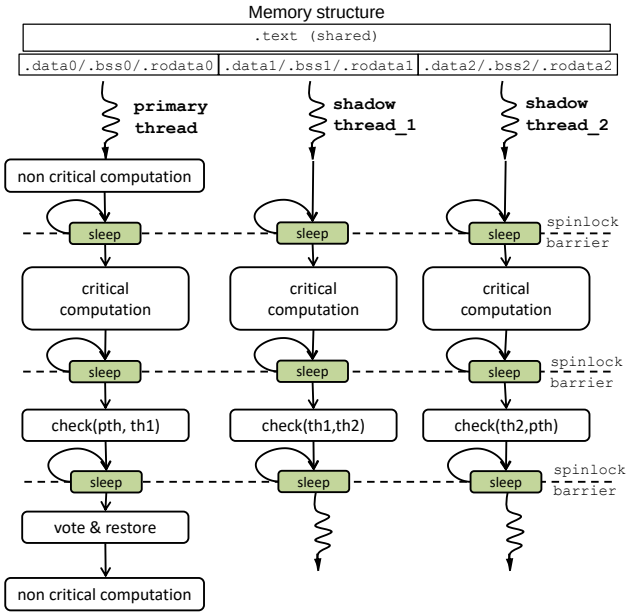

Fig. 1. Control flow of Triple Modular Redundancy (TMR) with redundant multi-threading (RMT).

and analysed to detect mismatches. A modification of this approach can be found in [17], where the verification process was performed directly in software. The architecture included some customized hardware IP to support the synchronized execution of redundant blocks of code on multi-core processors. In addition, several hardware (watchdog timer) and software mechanisms (exception handlers) were implemented to protect the system from hangs and unexpected events, accounting for up to 73% of the detected errors. The proposal was tested in bare-metal and with a real-time OS.

Only a few of the multi-threading based techniques has been tested in accelerated radiation experiments [2], [6], [11], [17] and some of them will be studied in Section V. A more comprehensive survey on multi-threading and lockstep based mitigation techniques can be found in the surveys of both [18] and [19].

III. Bare-metal Multi-Threading approach

Our software technique, Redundant Multi-Threading (RMT), makes use of common resources available on state-of-the-art COTS to apply redundancy and to increase system reliability with minimal additions. We followed the well-known TMR technique, adding the following characteristics: replica computing was performed in parallel on different resources and redundancy was defined in C++ in portable code.

In this sense, Fig. 1 shows the basic elements that are needed to enable the execution of redundant threads on bare-metal multi-core systems. On the one hand, the data structures (initialized `.data`, uninitialized `.bss` and read-only `.rodata`) are replicated, to enable the parallel execution of all the threads. On the other hand, there is only one common application code (`.text` section) that is shared between all the threads. The method is similar to a multi-threaded program on top of an operating system. However, in our bare-metal approach, the threads are statistically scheduled/mapped during the compilation process, to guarantee the simultaneous execution of all the threads and one thread per core. An operating system or any other supporting software layer , which might otherwise add complexity to the system, is therefore unnecessary. In addition, a custom barrier mechanism was designed to synchronize the threads and to ensure proper communication between them. The barriers were based on common resources available on modern multi-core processors: special shared variables (spinlocks) and atomic instructions, allowing access to mutually exclusive regions. In general, any thread that crosses the synchronization barrier remains asleep until the last thread reaches this point and wakes up the other threads, so that the execution of the program continues.

As can be seen in Fig. 1, before the critical computation (*i.e.*, the region of the code that must be protected), there is an initial barrier to any synchronization of the three threads. The first to cross the barrier are the shadow threads that then enter sleep mode. Once the preparation of the redundant execution is finished in the primary thread, it then crosses the barrier and the shadow threads are activated, so that they execute the critical code in parallel with the primary thread. Another barrier, after the critical region, ensures that all the threads will have finished the computation before voting on the results. At that point, the results are checked and a two-phase voting process ensues (with a barrier in between). First, the output variables of each thread are checked against its neighbor's output and one status bit is updated within a shared variable. Second, once all the checks have been performed, the three bits of the shared variable are tested in the primary thread, followed by majority voting. Any corrupted variables are also restored in the process, if needed.

Our approach presents two main limitations that should be assessed. First and in a similar way to other RMT techniques, our approach is susceptible to common mode failures, as the application code is not replicated. However, the execution of the threads is staggered when RMT is applied to multi-core processors, because of the serialized memory access and the use of local decoding buffers that can store many instructions. As a consequence, the possibility of common mode failures is reduced. Furthermore, only if all the results of each thread are identically erroneous will any error affecting the .text section go unnoticed. This situation is unlikely, as the memory space and the data management instructions are separately managed for each thread.

A second collateral effect is related to the synchronization of threads. As mentioned earlier, at each sync point all threads must have crossed the barrier for execution to continue, therefore any error which causes just one single thread to hang before crossing the barrier will mean that the rest of threads remain in sleep mode. Hence, synchronization barriers increase the probability of hangs, as more threads are running in parallel. The problem can be addressed through the addition of common monitoring mechanisms such as watchdogs and exception subroutines.

The Board Support Package (the files that guide the compilation process) was modified, and a custom C++ library was developed, for easy adoption and implementation of the approach. Including the library, the user only has to add minor modifications to its source code, *e.g.*, changing the data type of variables and including some macros to define the regions of code. The variable replicas, the synchronization, and the voting mechanisms are automatically added during the compilation process.

Figure 2 shows how the hardening instrumentation was applied to the original C++ code. Data protection was selectively and explicitly applied to each variable in use (`A`, `B`, and `Res`). To do so, three macros were employed. The first, `BSS_TRIPLICATION`, replaces the classical definition of the variable by three replicas allocated to different private .bss sections (*i.e.* .bss0, .bss1 and .bss2). Our library provides a macro for each type of data section (*e.g.* .bss, .data, .rodata, etc...). It also creates a shared 3-bit variable, transparent to the user, to store the status of the replicas after the critical computation. Each triplication macro is followed by a `define`. It defines an alias of the pointers created by the `PTR` macro to mask the replica access. The third macro, `REPLICA_VAR`, is activated within the function where the variable will be used and ensures that each replica is computed inside an exclusive memory area (*i.e.*, our Sphere of Replication is extended to the memory). It selects the correct replica according to the core identifier (`core_ID`) and assigns it to a pointer. The pointer will be allocated within the local stack or a (local) register of the core. Any other variable within the SoR (*i.e.*, i, j, k) is automatically triplicated in its own core address space. As can be seen in the `test` function implementation (fig. 2), the alias of the (globally defined) variables gives access to the replicas using the indirection defined in `PTR`.

The `main` function is instrumented adding the needed synchronizations points (`barrier`): before starting the critical code, before local checks on the results, and before the global voting process. Since four cores are available in Cortex-A72 processor, cores 0 to 2 are employed to run the shadow threads and core 3 is devoted to the primary thread. Different voting macros are available, depending on the code and the kind of protection that is needed. For instance, `VOTE` macro can be used to return just one correct copy of the result, while `VOTE_RESTORE` is a command to continue the computation using the three replicas. Additional code, labeled as "non critical stuff", is needed for reporting to the host during the radiation experiments.

```
/** Variable annotation & allocation **/
BSS_TRIPLICATION(matrix, A)
#define A, PTR(A)
BSS_TRIPLICATION(matrix, B)
#define B, PTR(B)
BSS_TRIPLICATION(matrix, Res)
#define Res, PTR(Res)

/** Initialization of inputs **/
void initMat(int seed)
{
  REPLICA_VAR(matrix, A)
  REPLICA_VAR(matrix, B)
  // random initialization of matrices
}

/*********  Critical code  **********/
void test()
{
  REPLICA_VAR(matrix, A)
  REPLICA_VAR(matrix, B)
  REPLICA_VAR(matrix, Res)
  register int i, j, k;
  for (i = 0; i < SIZE; i++)
  for (j = 0; j < SIZE; j++)
  {
      Res [i][j] = ZERO;
      for (k = 0; k < SIZE; k++)
        Res[i][j] += A[i][k] * B[k][j];
  }
}

/*********  Main  **********/
int main()
{
  GETID           // get the core ID
  if (core==3) {
    // do non critical stuff
  }
  barrier();      // start of critical code
  initMat(s);
  test();
  barrier();      // sync to start checking
  TMR_CHECK(Res) // distributed checking
  barrier();      // sync to start voting
  VOTE(Res)       // or vote & restore
  if (core=3){
    // do non critical stuff
  }
}
```

Fig. 2. Hardening instrumentation of the matrix multiplication algorithm (inserted macros and functions are highlighted in blue).

IV. Radiation Experiments

The radiation experiments were carried out at the Subatomic Physics & Cosmology Laboratory (LPSC, Grenoble, France) in February 2022 using a GENEPI2 (GEnérateur de NEutrons Pulsé Intense), a high intensity, pulsed neutron generator. GENEPi2 is a mono-energetic neutron generator with a maximum flux that exceeds the natural flux of 14-MeV neutrons at 40,000 ft by a factor of 10^{10}. The relevance of the 14-MeV neutron test for characterization of the Single Event Upset (SEU) sensitivity of digital devices is discussed in [20]. The same authors also proposed a model to extrapolate 14-MeV neutron results to other neutron and proton energies. Since then, several works have used mono-energetic neutrons, to evaluate the SEU sensitivity of multi-core processors [21], FPGAs [22] and memories [23].

A total fluence higher than $4.0 \times 10^{10} neutron/cm^2$ was chosen to achieve statistical significance for the different experiments, by accumulating event counts from multiple runs. Three radiation campaigns were performed during February and July 2022 with an average flux of 1.13×10^7, 9.20×10^6 and $4.71 \times 10^6 neutron/cm^2/s$ respectively.

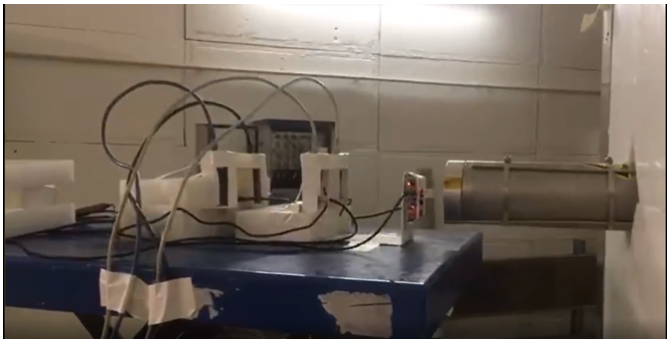

Fig. 3. Experimental setup of the High Intensity Pulsed Neutron Generator GENEPi2 facility

A Raspberry Pi v4 (RPI4) equipped with a BCM2711 SoC was selected as the Device Under Test (DUT), to assess our technique under radiation. The SoC is based on a quad-core ARM Cortex-A72 multiprocessor (ARMv8 architecture). This device was placed on an external beamline and subjected to an open-air irradiation process (fig.3).

The DUT was connected using Ethernet and TTL/UART to another RPI4 board used as the control computer outside the radiation chamber. During the experiments, the DUT was continuously emitting a *keep-alive* message to the control computer via UART. This message only included the number of runs executed and some auxiliary bytes of information. The time interval between successive messages was adjusted for each benchmark, to reduce the weight of the communication tasks with respect to the total execution time. In doing so, one notification was sent at approximate intervals of one minute.

The control computer was in charge of uploading the different benchmarks to the DUT using the Ethernet link and the Pre-Execution Environment (PXE) protocol. In addition, it recorded all the DUT output messages, and triggered the corresponding power cycle when the DUT entered into a non-response state.

During the experiments, three different anomalous states of the DUT were considered as possible events:

- A **HANG** event is detected when the DUT is unable to emit a *keep-alive* message within a period of time three times longer than the normal report.
- **Silent Data Corruption** (SDC) occurs when the code runs correctly, but the computation is incorrect. This event is detected by checking the computation outcome using a golden output.
- **Detect** (SDC detected) refers to the case in which a program that is running with hardening capabilities detects an error within the computed data by checking the output of the replicas.

The applications running on the DUT were instrumented, following the method defined in [24], to check the results against a golden output, so that any mismatch could be notified as soon as it happened.

The well-known matrix multiplication algorithm was the main benchmark that was selected to assess our hardening technique under mono-energetic neutrons. The benchmark was configured to test two different computing units: the integer ALU and the floating point ALU, by supporting two different data types: 64-bit (long long - L) integers, and 64-bit (double - D) floating points. We performed the experiment using two matrix sizes: 32×32 and 128×128. The benchmark was also tested in three different versions, depending on the hardening technique applied: Original (mm), Duplication With Comparison & Re-execution (mmDWC), Redundant Multi-Threading (mmRMT). Note that the DWC-R described in [4] and [6] used only one core to perform two identical executions of the critical section. Only if a mismatch was detected, was a third execution implemented. The mem benchmark was also provided for the assessment with the sole purpose of measuring the sensitivity of the external Double Data Rate (DDR) memory.

Table I shows the static characterization of the application overheads: resource utilization (number of cores), memory footprint (data), and performance (exec. Time). As can be seen, resource utilization in the hardened versions using the RMT technique increased, as was expected, by a factor of 3. In contrast, the same overheads as the unhardened versions were observed in the conventional technique (mmDWC), which uses a single core. All of them showed an increase in memory structures, in terms of memory footprint. In the case of the mmDWC, only two output matrices were used to fit the original and the redundant calculation. In contrast, the RMT versions showed an increase of $3\times$, because all the matrices had to be replicated so that computation (inputs and results) could proceed. Moving to the performance overhead, it can be seen that the mmDWC showed, as expected, the worst performance, because it was sequentially computing the replica outcomes. In the case of the mmRMT versions, the synchronization mechanism for the verification of replica outcomes produced the overhead. Note that the fewer the number of matrix elements and the smaller the matrix size, the greater the impact of the verification routine (see mmRTM-L32).

V. Results and discussion

A. Cross-section

The results of the radiation experiments are shown in Table II. The total fluence values accumulated for each benchmark (Φ) are listed in the third column and the number of events observed (#SDC, #DetRec, #HANG) and the corresponding cross-section (σ) for each type of event are shown in the other columns. Upper and lower error margins can be seen above and below the cross-sectional measurements. They

TABLE I
Time and Size Overheads

benchmark	#cores	data	exec. Time
mmDWC-L128	$1\times$	$1.33\times$	$2.19\times$
mmRMT-L32	$3\times$	$3\times$	$1.59\times$
mmRMT-L128	$3\times$	$3\times$	$1.34\times$
mmRMT-D128	$3\times$	$3\times$	$1.34\times$

were calculated using the inverted chi_square distribution as described in [25], at a confidence level of 95% considering a fluence uncertainty of $\pm 10\%$. An additional category labeled as Total is included. It represents the total harmful events in the experiment, *i.e.*, SDC + HANG. Notice that all SDC events were detected and corrected in the RMT versions, so the Total and the HANG cross-sections were equal in those cases.

The first benchmark (mem) was used to report a baseline of the architectural board sensitivity, mainly external memory resources (caches were disabled in bare-metal). Mem code writes a vector of 64KB to memory and reads it back repeatedly to detect any error. Only three events were reported for this benchmark during the assessment time, providing a cross-section of 3.47×10^{-11} and 1.74×10^{-11} for SDC and HANG, respectively.

Focusing on the original Matrix Multiplication benchmark, the L32 versions presented a reduced memory footprint when compared to the mem version. However, its superior usage of processor resources justifies the increase in the cross-section. An increase of 68% in the total cross-section of the unhardened version (mm-L32) was observed, while the increase in the total cross-section of the redundant version (RMT-L32) was around 55%, due to its ability to detect soft errors. Remarkably, the RMT-L32 version detected and recovered all the SDC events (now labeled as Det/Rec), but at the cost of an increase in the HANG sensitivity, at around 1.8× from 4.38×10^{-11} to 8.09×10^{-11}. The cause was related to the additional resources needed to run the redundant threads (3 cores vs 1 core) and the synchronization constraint imposed by the technique. As mentioned in Section III, if any one thread at any one time fails to reach the synchronization point, then the synchronization barriers limit the progress of the threads and the whole program hangs. Consequently, a very limited improvement in the total cross-section was shown for the RMT.

When RMT was applied to process matrices of higher sizes (D128 and L128), improvements in the levels of SDC were detected. In both cases, integer and floating point data types, all SDC events, were detected and corrected. Predictably, there was also a worsening of the HANG cross-section which reached an increment of 4.2× (RMT-L128) and 2× (RMT-D128) with respect to the unhardened version (mm-L128). Even so, the integer version offered a slightly improved overall cross-section, achieving a 2.6× improvement in the case of the floating version.

Finally, the DWC technique was tested and compared with our RMT approach. As can be appreciated, the DWC-L128 benchmark presented a lower detection & recovery cross-section (2.46×10^{-11}) than its RMT counterpart (3.18×10^{-11}), which may be attributed to the different data volumes of each technique. The original inputs (matrices A and B) were processed twice for the DWC benchmark and two replicas of the result were produced (*i.e.*, 2.25× of memory overhead), while all the inputs and outputs were triplicated in the multi-threaded (RMT) version. It means a lower Det/Rec cross-section of DWC version (2.46 vs 3.18 and 2.88). On the contrary, DWC offered a worsening of 6.2× in the HANG cross-section (1.37×10^{-11} vs 8.44×10^{-11}). So, even when triplicating the numbers of cores in use, our approach was less sensitive to HANG errors than the other state-of-the-art techniques. Consequently, the total cross-section of both, RMT-L128 and RMT-D128, outperformed the cross-setion of DWC-L128 by 1.45 times and 3.2 times.

Taking into account that HANG errors can generally be managed with conventional monitoring mechanisms, it is worth analysing the SDC cross-section separately. Although no SDC events were observed for the hardened versions, the cross-section can be calculated as $1/\Phi$, (*i.e.*, assuming the worst-case where an error could be observed in the very next instant of the experiment). In this case, the RMT versions outperformed the σ_{SDC} of the corresponding unmitigated benchmarks by 7.6× (L32), 6.3× (L128), and 12.2× (D128). These improvements clearly exceeded those provided by DWC, which only achieved up to 0.4× (L128). They also revealed that the floating point units were less sensitive to radiation than the integer units.

B. Failure Rate Metrics

In Table III, the Failures in Time (FIT), are presented, which refer to the number of expected failures per one billion hours of operation for a device. The FIT values were normalized using the sea level flux of $13n/(cm^2 \cdot h)$ as suggested in the JESD89 standard [26]. The table follows the same structure as Table II and therefore reproduces the same headings. It can be observed that the DWC offers the worst FIT, because of the high impact the HANGs have on it. The same reason applies to the rest of the hardened versions which slightly increases the total FIT. Only the mm-RMT-D128 version reduced the total FIT from 6.18 to 3.86.

The cross-section and the FIT metrics do not take into account the performance or time overheads which are inherent to the software-based technique. Therefore, the MWTF metric [27] is often used to compare different protection techniques, as it captures the trade-off between the error rate and the time overhead. MWTF can be calculated from the results of a radiation experiment as follows:

$$MWTF = (Flux \cdot \sigma \cdot ExecutionTime)^{-1}$$

where the $ExecutionTime$ is the time needed to compute a single benchmark workload in seconds, σ is the dynamic cross-section, and $Flux$ is the radiation flux.

Table III also shows the MWTF improvements with respect to the corresponding unmitigated version (mm-L32 and mm-L128). Considering the totality of all errors, the MWTF was not increased in the multi-threaded version. Despite the parallelism of the RMT technique, there was a time overhead inherent to the checking and synchronization tasks which reduce the amount of work computed before the fault. Furthermore, the synchronization mechanism is sensitive to radiation and is often triggered, due to the short duration of the benchmarking process, thereby raising the number of HANG errors. Both factors contributed to a reduction in the overall MWTF, in such a way that only the mmRMT-D128 version can match the value obtained by the original version.

TABLE II
NEUTRON BEAM TEST RESULTS

			#events			$\sigma \times 10^{-11}(cm^2)$			
bench	cores	$\Phi(n/cm^2)$	#SDC	#Det/Rec	#HANG	SDC	Det/Rec	HANG	Total
mem-L64K	1	5.76×10^{10}	2	–	1	$3.47\ ^{12.1}_{0.0}$	–	$1.74\ ^{10.4}_{0.0}$	$5.21\ ^{15.6}_{1.7}$
mm-L32	1	1.60×10^{11}	7	–	7	$4.38\ ^{9.4}_{1.9}$	–	$4.38\ ^{9.4}_{1.9}$	$8.77\ ^{15.0}_{5.0}$
mmRMT-L32	3	1.73×10^{11}	N/A	9	14	$^{**}0.578\ ^{3.5}_{0.0}$	$7.96\ ^{15.0}_{3.5}$	$8.09\ ^{13.9}_{4.6}$	$8.09\ ^{13.9}_{4.6}$
mm-L128	1	2.19×10^{11}	10	–	3	$4.58\ ^{8.70}_{2.3}$	–	$1.37\ ^{4.1}_{0.5}$	$5.95\ ^{10.5}_{3.2}$
mmDWC-L128	1	5.92×10^{10}	N/A	1	5	$^{**}1.69\ ^{10.1}_{0.0}$	$2.46\ ^{14.8}_{0.0}$	$8.44\ ^{20.3}_{3.4}$	$8.44\ ^{20.3}_{3.4}$
mmRMT-L128	3	1.38×10^{11}	N/A	6	8	$^{**}0.752\ ^{4.4}_{0.0}$	$3.18\ ^{6.9}_{1.1}$	$5.80\ ^{11.6}_{2.2}$	$5.80\ ^{11.6}_{2.2}$
mmRMT-D128	3	2.66×10^{11}	N/A	4	7	$^{**}0.375\ ^{2.3}_{0.0}$	$2.88\ ^{7.2}_{0.7}$	$2.63\ ^{5.6}_{1.1}$	$2.63\ ^{5.6}_{1.1}$

** No errors observed, so for comparison purposes this is calculated given one error (assuming the worst-case)

However, looking at nothing other than the SDC contribution, it can be seen that the MWTF was increased with our approach from 3.9× (mmRMT-L128) to 5.3× (mmRMT-D128). These values were in the same range as those offered by other software-based state-of-the-art proposals, as is described in the following subsection.

TABLE III
FAILURES METRICS: FAILURES IN TIME AND MEAN WORK TO FAILURE INCREASE REGARDING THE UNHARDENED VERSIONS

	FIT			ΔMWTF	
bench	SDC	HANG	Total	SDC	Total
mm-L32	4.78	4.78	9.56	1.0	1.0
mmRMT-L32	0.00	9.65	9.65	4.81	0.69
mm-L128	4.75	1.43	6.18	1.0	1.0
mmDWC-L128	0.00	10.94	10.94	1.01	0.26
mmRMT-L128	0.00	6.47	6.47	3.88	0.63
mmRMT-D128	0.00	3.86	3.86	5.34	0.98

C. Comparison with related works

Among all the results of the software-based approaches tested under radiation there were only a few that stood comparison with our proposal. Table IV summarizes the main features and results of three of the state-of-the-art techniques [14], [6], [17]. Beams of a different nature and energy (low energy protons, atmospheric and mono-energetic neutrons) were used for testing in all of the studies, making any direct comparison of the cross-section impossible. However, valuable information on the performance of the technique can be appreciated in the relative improvements reported (marked in bold). Please note that in all the works they refer to cross-section and MWTF as the contribution of the SDC events exclusively. Thus σ_{HANG} was calculated from the total fluence and event counts provided by the authors.

The Matrix multiplication benchmark running on similar ARM architectures was the only benchmark considered for comparative purposes. Note that the benchmarks follow the same coding name, mmXX-YZZ, where XX refers to the technique applied, Y refers to the data type, and ZZ to the size of the square matrices. They are grouped in four sets related to the mitigation technique applied and from which the radiation data were obtained.

The VAR technique [14] defines a set of rules to triplicate data and instructions at assembly level and running on a single-core. Authors in [15] use the COAST tool to produce TMR-processed portable versions automatically from several source code benchmarks, reporting σ_{SDC} improvements from 4× up to two orders of magnitude. As can be seen, they tested the benchmark running on bare-metal with and without activating the memory caches, resulting in σ_{SDC} (MWTF$_{SDC}$) enhancements of 4.3× (1.2×) and 31.7× (10.1×) respectively.

The Macro Lockstep approach (MLockstep) [17] incorporates a rollback/recovery mechanism to synchronize the redundant execution of two cores. Although no single-core version was evaluated in this work, the data were completed with the results reported in [28] at the same facility and using an identical setup. In this way, the improvements relating to the baseline version were calculated with a value of 16.7× for the bare-metal version and 7.3× for the version running on an Operating System.

Finally, the single-core version of the DWC technique was evaluated in our own work. We followed the description in [6] to implement the DWC version, so that it could be considered equivalent to the COAST produced benchmark. As can be seen, it obtained the worst results: 0.4× and 1.01× for σ and MWTF respectively.

Regarding the RMT approach, it can be appreciated that our results were the best for the bare-metal/no-cache configurations. The cross-section was reduced by 7.6× (RMT-L32) and 6.3× (RMT-L128) vs 4.3× (VAR) and 0.4× (DWC), while the MWTF was increased 4.8×/3.9× vs 1.2×/1.01×. These results remain true even considering two detrimental factors in our experiments: the larger sizes of the matrices employed (32 and 128 vs 30) and the protection of the external memory (not included in other techniques). Furthermore, the increase

TABLE IV
COMPARISON TABLE OF RELATED WORKS

Tech, [work] arch, particles	benchmark configuration	Φ $(ptcl/cm^2)$	#SDC	#Det/Rec	#HANG	σ_{SDC} $(\times 10^{-11} cm^2)$		σ_{HANG} $(\times 10^{-11} cm^2)$		MWTF$_{SDC}$ $(\times 10^{10})$	
VAR [15] ARM CortexA9 1 core, atmospheric neutrons	mm-I30	8.30×10^{10}	13	N/A	14	16.0	–	*16.9	–	0.117	–
	mmTMR-I30	8.10×10^{10}	3	26	19	3.70	↓ **4.3**	*23.5	↑ **1.4**	0.144	↑ **1.2**
	mm-I30, Cache	3.70×10^{10}	28	N/A	3	76.0	–	*8.11	–	0.624	–
	mmTMR-I30, Cache	8.30×10^{10}	2	163	165	2.40	↓ **31.7**	*199	↑ **24.5**	6.31	↑ **10.1**
MLockstep [17] ARM CortexA9 2 cores, protons	*mm-I20, Cache	1.20×10^{12}	80	N/A	13	6.67	–	*1.08	–	N/A	–
	mmML-I20, Cache	4.90×10^{11}	2	142	131	0.40	↓ **16.7**	*26.7	↑ **24.8**	N/A	N/A
	mmML-I20, Cache, OS	5.50×10^{11}	5	65	183	0.91	↓ **7.3**	*33.3	↑ **30.8**	N/A	N/A
RMT [our work] ARM CortexA72 3 cores, mono-energetic neutrons	mm-L32	1.60×10^{11}	7	N/A	7	4.38	–	4.38	–	42.5	–
	mmRMT-L32	1.73×10^{11}	N/A	9	14	**0.578	↓ **7.6**	8.09	↑ **1.8**	204	↑ **4.8**
	mm-L128	2.19×10^{11}	10	N/A	3	4.58	–	1.37	–	0.830	–
	mmRMT-L128	1.38×10^{11}	N/A	6	8	**0.725	↓ **6.3**	5.80	↑ **4.2**	3.22	↑ **3.9**
DWC, 1 core	mmDWC-L128	5.92×10^{10}	N/A	1	5	**1.69	↓ **0.4**	8.44	↑ **1.5**	0.842	↑ **1.01**

* Data obtained from [28], where authors used the same setup and facility as in [17]
** No errors observed, so for comparison purposes this is calculated given one error (assuming the worst-case)
* Calculated with data provided by [17]

in the σ_{HANG} was similar (1.4× vs 1.8× and 1.5×) for a similar matrix size (32 vs 30).

It is worth noting that unlike our approach, the data in the external memories are not protected with VAR and MLockstep, so the Sphere of Replication is limited to the processor cores and the internal memories (caches). In fact, external memories were left out of beam during the e [5] and [17]. In addition, when cache memories were enabled, any sensitivity to errors significantly increased, as may be observed in the SDC, Det/Rec and HANG event counts. Enabling the cache memory offers more opportunities for the technique to mitigate SDCs and to scale the cross-setion by 31.7× (VAR) and 16.7× (MLockstep). Also the σ_{HANG} increased by one order of magnitude (around 24× in both cases) which clearly exceeds the modest worsening of the RMT versions (1.8× and 4.2×).

VI. CONCLUSIONS

A new and portable software technique has been presented to protect COTS multi-core microprocessors against soft errors. The technique takes advantage of the parallel computing resources of modern COTS micro-processors to split critical computation into multiple instruction flows and to design a redundant and reliable computation. The technique has been evaluated under radiation using 14MeV neutrons. The results have shown that single event faults can be detected and corrected with this technique, improving the results of the state-of-the-art software-based mitigation techniques for bare-metal applications. In fact, all the Redundant Multi-Threading versions that have been presented were capable of detecting and recovering all the SDC events, and they only increased the HANG sensitivity moderately, compared to other single and multi-threading techniques.

No other hardware is required for the design of this redundant and reliable computation method. It was implemented in this study using a C++ library, making it a highly portable and transparent for the user.

REFERENCES

[1] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (TCLS) ARM® cortex®-r5 processor for safety-critical and ultra-reliable applications," in *2016 46th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, jun 2016, pp. 246–249.

[2] A. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt, N. Added, E. L. A. Macchione, V. A. P. Aguiar, N. H. Medina, and M. A. G. Silveira, "Lockstep dual-core ARM a9: Implementation and resilience analysis under heavy ion-induced soft errors," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1783–1790, aug 2018.

[3] S.-C. A. R.-C. F. C.-A. S. et al, "Multi-threaded mitigation of radiation-induced soft errors in bare-metal embedded systems," *J Electron Test*, vol. 36, pp. 47–57, aug 2020.

[4] M. Peña-Fernández, A. Serrano-Cases, A. Lindoso, S. Cuenca-Asensi, L. Entrena, Y. Morilla, P. Martín-Holgado, and A. Martínez-Álvarez, "Hybrid lockstep technique for soft error mitigation," *IEEE Transactions on Nuclear Science*, vol. 69, no. 7, pp. 1574–1581, jul 2022.

[5] B. James, H. Quinn, M. Wirthlin, and J. Goeders, "Applying compiler-automated software fault tolerance to multiple processor platforms," *IEEE Transactions on Nuclear Science*, vol. 67, no. 1, pp. 321–327, jan 2020.

[6] H. Quinn, Z. Baker, T. Fairbanks, J. L. Tripp, and G. Duran, "Software resilience and the effectiveness of software mitigation in microcontrollers," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2532–2538, dec 2015.

[7] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," *SIGARCH Comput. Archit. News*, vol. 28, no. 2, p. 25–36, may 2000.

[8] Y. Zhang, J. W. Lee, N. P. Johnson, and D. I. August, "Daft: Decoupled acyclic fault tolerance," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, feb 2010, pp. 87–97.

[9] K. Mitropoulou, V. Porpodas, and T. M. Jones, "Comet: Communication-optimised multi-threaded error-detection technique," in *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, oct 2016, pp. 2.3.1–2.3.10.

[10] B. Döbel and H. Härtig, "Can we put concurrency back into redundant multithreading?" in *Proceedings of the 14th International Conference on Embedded Software*, ser. EMSOFT '14, Art. no. 19. Association for Computing Machinery, oct 2014.

[11] G. Rodrigues, F. Rosa, A. de Oliveira, F. L. Kastensmidt, L. Ost, and R. Reis, "Analyzing the impact of fault tolerance methods in ARM processors under soft errors running linux and parallelization APIs," *IEEE Transactions on Nuclear Science*, vol. 64, no. 8, pp. 2196–2203, may 2017.

[12] D. P. Hukerikar S., Teranishi K., "Redthreads: An interface for application-level fault detection/correction through adaptive redundant

multithreading," *International Journal of Parallel Programming*, vol. 46, p. 225–251, oct 2018.

[13] J. S. Monson, M. Wirthlin, and B. Hutchings, "Fault injection results of linux operating on an FPGA embedded platform," in *2010 Int. Conference on Reconfigurable Computing and FPGAs*. IEEE, dec 2010, pp. 37–42.

[14] E. Chielle, F. Rosa, G. S. Rodrigues, L. A. Tambara, J. Tonfat, E. Macchione, F. Aguirre, N. Added, N. Medina, V. Aguiar, M. A. G. Silveira, L. Ost, R. Reis, S. Cuenca-Asensi, and F. L. Kastensmidt, "Reliability on arm processors against soft errors through sihft techniques," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2208–2216, aug 2016.

[15] B. James, M. Wirthlin, and J. Goeders, "Investigating how software characteristics impact the effectiveness of automated software fault tolerance," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1014–1022, may 2021.

[16] M. Violante, C. Meinhardt, R. Reis, and M. Sonza Reorda, "A low-cost solution for deploying processor cores in harsh environments," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 7, pp. 2617–2626, 2011.

[17] P. M. Aviles, A. Lindoso, J. A. Belloch, M. Garcia-Valderas, Y. Morilla, and L. Entrena, "Radiation testing of a multiprocessor macrosynchronized lockstep architecture with freertos," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 462–469, nov 2022.

[18] I. Oz and S. Arslan, "A survey on multithreading alternatives for soft error fault tolerance," *ACM Comput. Surv.*, vol. 52, no. 2, art. no. 47, pp. 1–38, mar 2019.

[19] E. W. Wächter, S. Kasap, X. Zhai, S. Ehsan, and K. McDonald-Maier, "Survey of lockstep based mitigation techniques for soft errors in embedded systems," in *2019 11th Computer Science and Electronic Engineering (CEEC)*, jan 2019, pp. 124–127.

[20] F. Miller, C. Weulersse, T. Carrière, N. Guibbaud, S. Morand, and R. Gaillard, "Investigation of 14 mev neutron capabilities for seu hardness evaluation," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2789–2796, feb 2013.

[21] P. Ramos, V. Vargas, M. Baylac, F. Villa, S. Rey, J. A. Clemente, N.-E. Zergainoh, J.-F. Méhaut, and R. Velazco, "Evaluating the see sensitivity of a 45 nm soi multi-core processor due to 14 mev neutrons," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2193–2200, jul 2016.

[22] J. C. Fabero, H. Mecha, F. J. Franco, J. A. Clemente, G. Korkian, S. Rey, B. Cheymol, M. Baylac, G. Hubert, and R. Velazco, "Single event upsets under 14-mev neutrons in a 28-nm sram-based fpga in static mode," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1461–1469, mar 2020.

[23] J. A. Clemente, G. Hubert, J. Fraire, F. J. Franco, F. Villa, S. Rey, M. Baylac, H. Puchner, H. Mecha, and R. Velazco, "Seu characterization of three successive generations of cots srams at ultralow bias voltage to 14.2-mev neutrons," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1858–1865, mar 2018.

[24] H. Quinn, W. H. Robinson, P. Rech, M. Aguirre, A. Barnard, M. Desogus, L. Entrena, M. Garcia-Valderas, S. M. Guertin, D. Kaeli, F. L. Kastensmidt, B. T. Kiddie, A. Sanchez-Clemente, M. S. Reorda, L. Sterpone, and M. Wirthlin, "Using benchmarks for radiation testing of microprocessors and fpgas," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2547–2554, dec 2015.

[25] ESA/ESCC, "Single event effects test method and guidelines. escc basic specification no. 25100," Oct 2014.

[26] JEDEC, *JESD89A, Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices.*, JEDEC Solid State Technology Association, 2006.

[27] G. A. Reis, J. Chang, N. Vachharajani, S. S. Mukherjee, R. Rangan, and D. I. August, "Design and evaluation of hybrid fault-detection systems," in *32nd International Symposium on Computer Architecture (ISCA'05)*, jun 2005, pp. 148–159.

[28] A. Lindoso, M. García-Valderas, L. Entrena, Y. Morilla, and P. Martín-Holgado, "Evaluation of the suitability of neon simd microprocessor extensions under proton irradiation," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1835–1842, jul 2018.