Selective Validations for Efficient Protections on Coarse-Grained Reconfigurable Architectures

Jihoon Kang

The Graduate School
Yonsei University
Department of Computer Science

Selective Validations for Efficient Protections on Coarse-Grained Reconfigurable Architectures

A Masters Thesis
Submitted to the Department of Computer Science
and the Graduate School of Yonsei University
in partial fulfillment of the
requirements for the degree of
Master of Science

Jihoon Kang

June 2013

This certifies that the masters thesis of Jihoon Kang is approved.

Thesis Supervisor: Kyoungwoo Lee

_____

Thesis Committee Member #1: Yosub Han

_____

Thesis Committee Member #2: Bernd Burgstaller

_____

The Graduate School
Yonsei University

June 2013

Table of Contents

Abstract

Selective Validations for Efficient Protections on Coarse-Grained Reconfigurable Architectures

Jihoon Kang
Dept. of Computer Science
The Graduate School
Yonsei University

Coarse-Grained Reconfigurable Architectures or CGRAs are drawing significant attention since they promise both performance with parallelism and flexibility with reconfiguration. Soft errors or transient faults are becoming a serious design concern in embedded systems including CGRAs since soft error rate is increasing exponentially as technology scaling. A recently proposed software-based technique with TMR (Triple Modular Redundancy) implemented on CGRAs incurs extreme performance overhead mainly due to expensive voting mechanisms for the outputs from the triplication of every operation and energy consumption. In this thesis, we propose selective validation mechanisms for efficient modular redundancy techniques in the datapaths on CGRAs. Our techniques selectively validate the results at synchronous operations rather than every operation in order to reduce the expensive performance overhead from the validation mechanism. We also present an optimization technique to further improve the performance and the energy consumption by minimizing synchronous operations where validating mechanism needs to be applied. Our experimental results demonstrate that our selective validation based TMR technique with our optimization on CGRAs can improve the performance by 41.0% and the energy consumption by 26.2% on average over benchmarks as compared to the recently proposed software-based TMR technique with the full validation.

Keywords : CGRA, Soft Error, Selective, Reliability, Error Protection, DFG(Data Flow Graph), Validation, TMR, Mapping, Fault Coverage

I. Introduction

Coarse-Grained Reconfigurable Architecture or CGRA is receiving lots of attentions. It is necessary to achieve not only high performance but also power efficiency in recent embedded systems. CGRA is in general composed of grid based processing elements (PEs) and each PE consists of a FU (Functional Unit) and a few registers as shown in Figure 1. CGRA is a promising alternative as an accelerator since this simple architecture can improve the performance massively by executing application loop kernels on PEs in parallel with the inherent efficacy of power consumption. Further, CGRA is programmable, i.e., able to reconfigure architectures by switching CGRA configuration for a new application in the short amount of time. Thus, CGRAs have been used to accelerate complex applications where high performance is required with the power efficiency [1], [2].

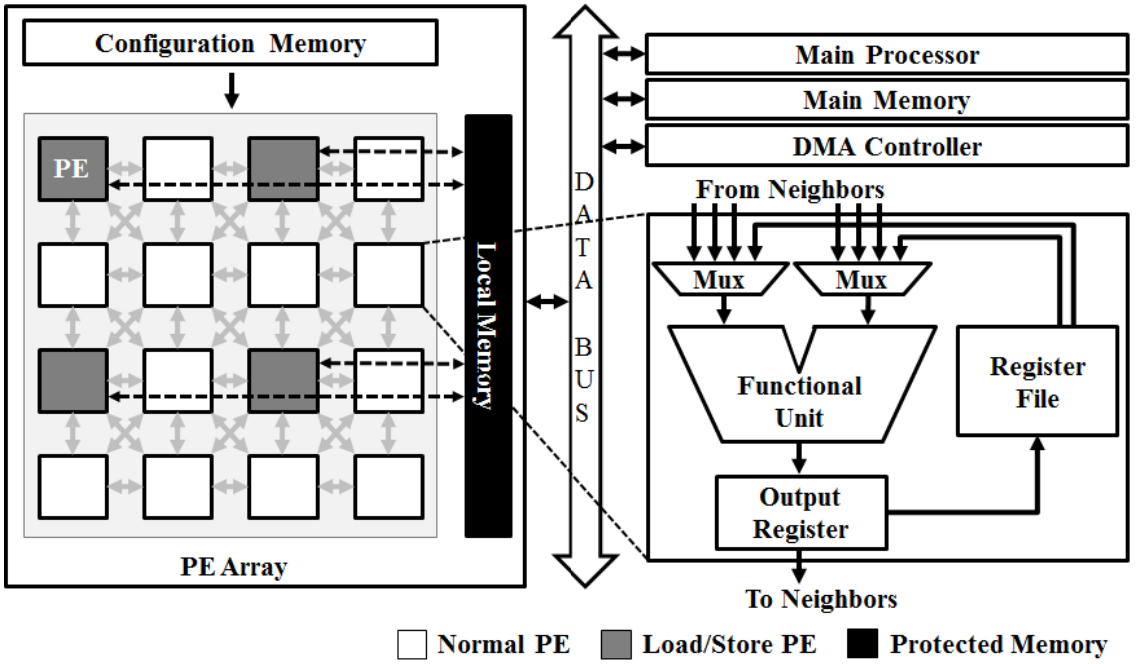
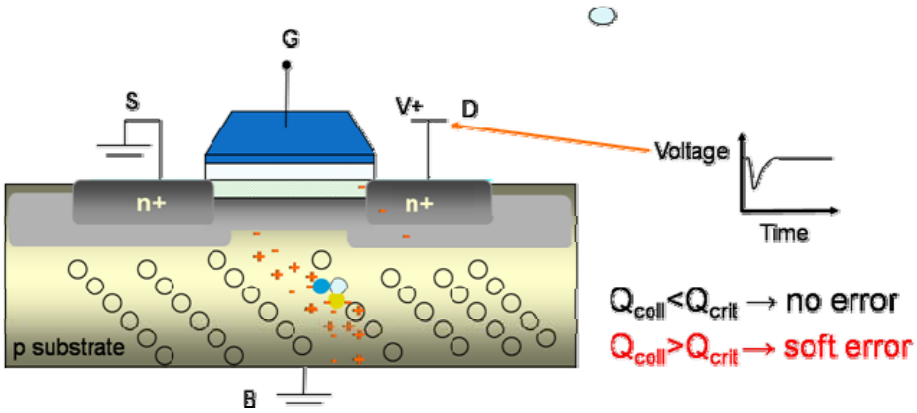
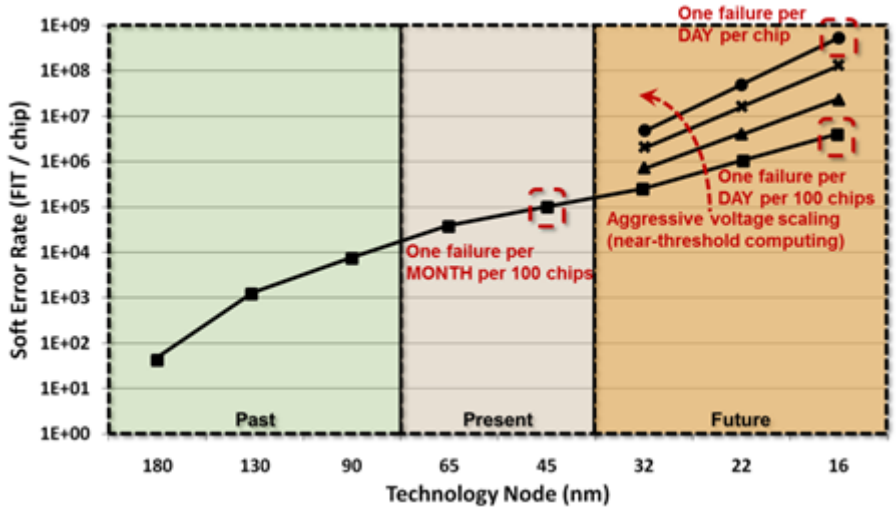


Fig. 1. CGRA (4 × 4) architecture (Example)

Soft error and its concern are on significant increase in embedded system designs. Several decades of technology scaling has brought us where transistors are extremely susceptible to even small fluctuations in supply voltage levels, slight noise in the power, signal interference, and even induced radiation [3], [4], [5]. Any of these effects can temporarily toggle the logic value of a transistor, so it is called a transient fault or soft error as shown in Figure 2.



(a) Soft Error [37]



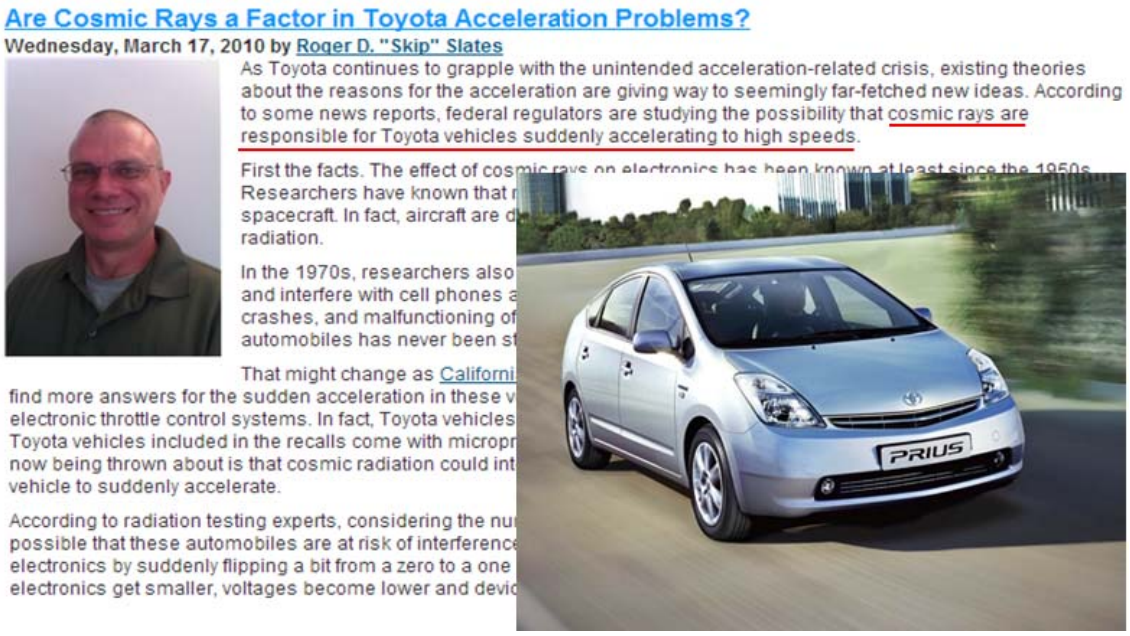
(b) SER is increase as the technology scaling [38]

Fig. 2. Concept of soft error and its concerning

Soft errors induced by terrestrial radiation are becoming a significant concern in architectures designed in newer technologies. If left undetected, these errors can result in catastrophic consequences or costly maintenance problems in different embedded applications. When high energy neutrons such as terrestrial cosmic radiation, alpha particles that originate from impurities in the packaging materials, strike a sensitive region in a semiconductor device, they generate a dense local track of electron hole pairs. This may be collected by a p-n junction resulting in a current pulse of very short duration termed a single event upset (SEU) in the signal value. A SEU may cause a bit flip in some latch or memory element thereby altering the state of the system resulting in a soft error. Additionally, a SEU may occur in an internal node of combinational logic and subsequently propagate to and be captured in a latch. Soft errors in memories (both static and dynamic) have traditionally been a much greater concern than soft errors in logic circuits since memories contain by far the largest number and density of bits susceptible to particle strikes. Soft errors will be an increasing burden for embedded system designers as the number of on-chip transistors continues to grow exponentially. The raw error rate per latch or SRAM bit is projected to remain roughly constant or decrease slightly for the next several technology generations. Thus, unless we add error protection mechanisms or use a more robust technology, a microprocessor's error rate will grow in direct proportion to the number of devices we add to a processor in each succeeding generation However, error correction is expensive in terms of power consumption and performance overhead.

Such a soft error is not permanent and non-destructive, i.e., resetting the device can resume the normal operation. However, a single soft error can be as critical as a permanent error. Indeed, soft errors have been already revealed to cause significant fiscal damages [6], [7], [8]. For instance, SUN blamed soft errors for the crash of their million-dollar line SUN flagship [7] as shown in Figure 3 and Hewlett Packard acknowledged that a large installed base of a 1024-CPU server system in Los Alamos National Laboratory has been crashing [8] due to soft errors caused by cosmic ray and energetic particle. Further, abrupt unattended acceleration of vehicle of Toyota might be caused by soft errors induced by comic ray [29] as shown in Figure 4. As the popularity of CGRA usages is increasing on many embedded applications such as human health systems, automobiles, airplanes, and data server systems [9], a single soft error may lead to catastrophic consequence, and even a human life as shown in Figure 5.

Fig. 3. Sun flagship server (example, not directly related) [39]



Are Cosmic Rays a Factor in Toyota Acceleration Problems?

Wednesday, March 17, 2010 by Roger D. "Skip" Slates

As Toyota continues to grapple with the unintended acceleration-related crisis, existing theories about the reasons for the acceleration are giving way to seemingly far-fetched new ideas. According to some news reports, federal regulators are studying the possibility that cosmic rays are responsible for Toyota vehicles suddenly accelerating to high speeds.

First the facts. The effect of cosmic rays on electronics has been known at least since the 1950s. Researchers have known that r
spacecraft. In fact, aircraft are d
radiation.

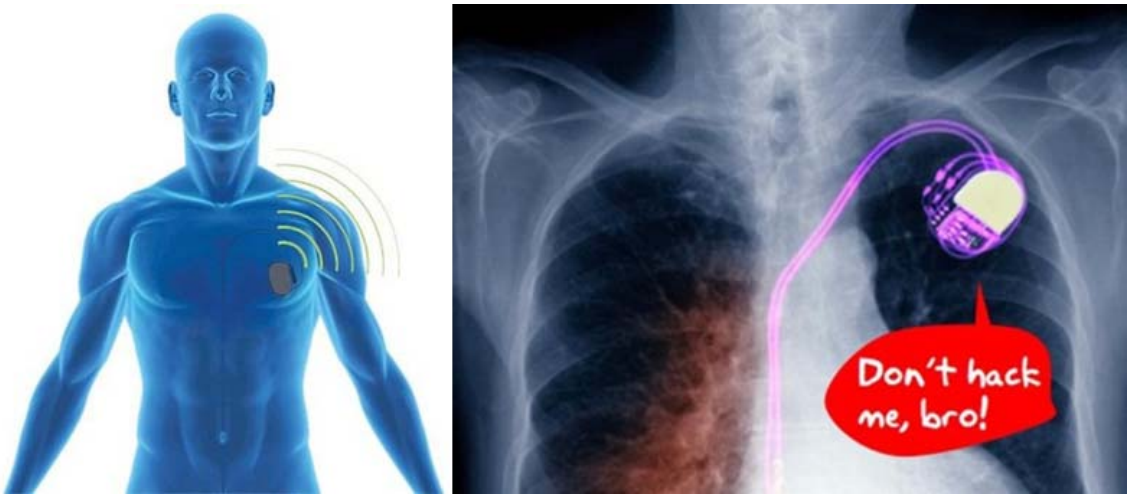
In the 1970s, researchers also
and interfere with cell phones a
crashes, and malfunctioning of
automobiles has never been st

That might change as Californi
find more answers for the sudden acceleration in these v
electronic throttle control systems. In fact, Toyota vehicles
Toyota vehicles included in the recalls come with micropr
now being thrown about is that cosmic radiation could int
vehicle to suddenly accelerate.

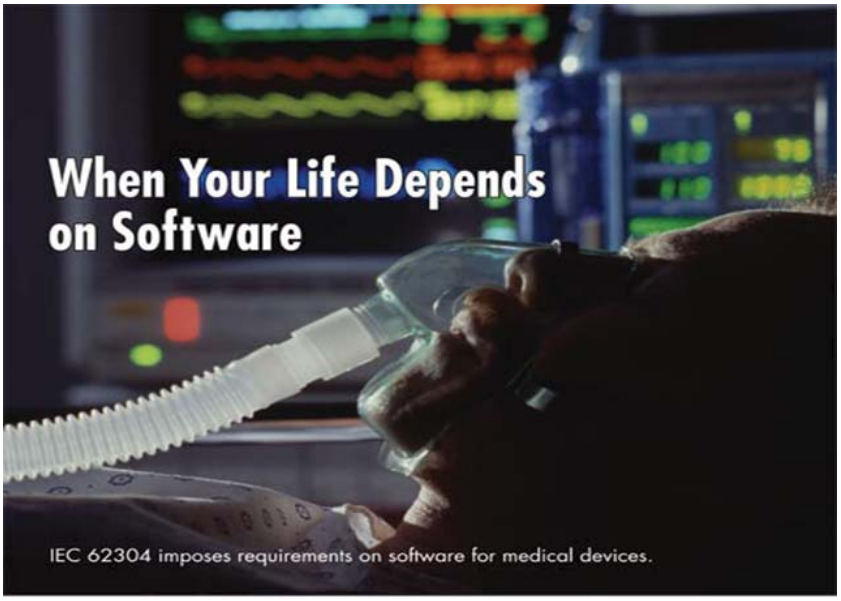
According to radiation testing experts, considering the nu
possible that these automobiles are at risk of interference
electronics by suddenly flipping a bit from a zero to a one
electronics get smaller, voltages become lower and devic

Fig. 4. Unintended acceleration of Toyota PRIUS [40]

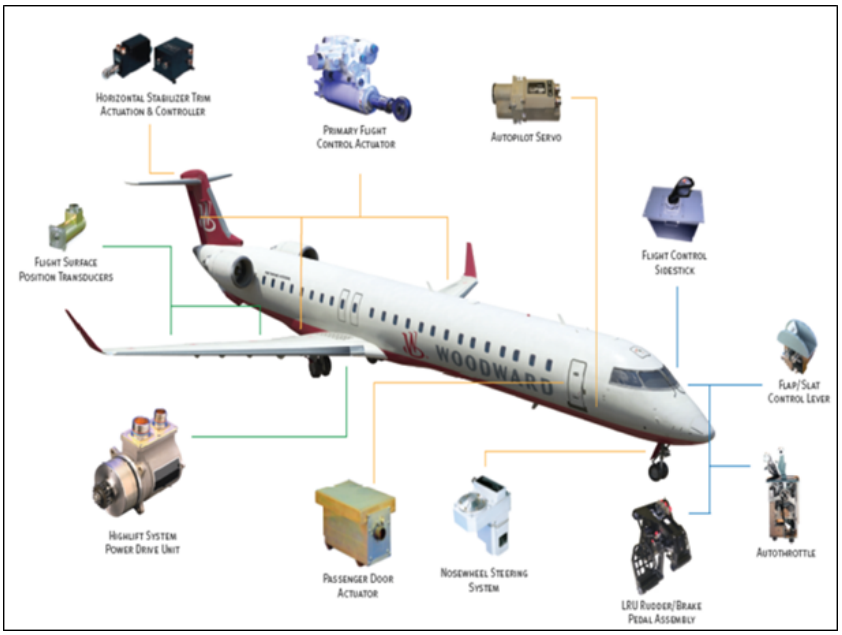
(a) Explosion of Fukushima Daiichi nuclear power plant [41]



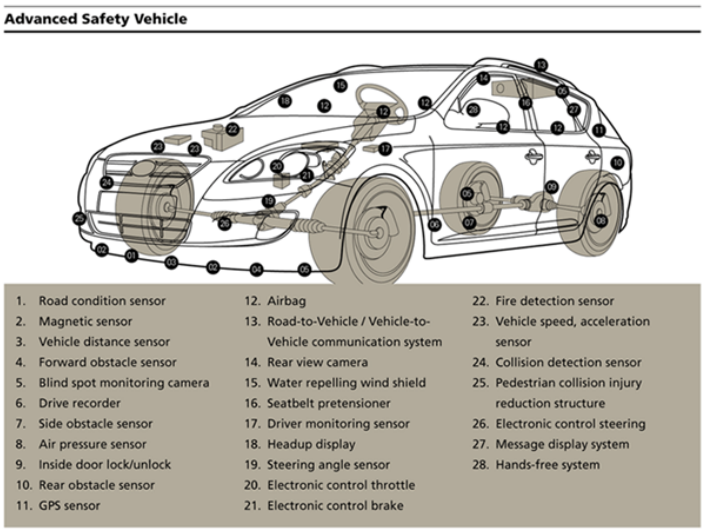
(b) Soft error of implanted devices [42]



(c) Human can depend on medical devices [43]



(d) Airplane can be used widely close to human life [44]



(e) Vehicle engine control unit is becoming an important issue [45]

Fig. 5. Soft errors can lead to catastrophic consequences in embedded systems

Soft-errors are protected by error detection and correction codes (EDC and ECC). However, soft errors in logics are becoming also critical and take up more than 50% in overall soft errors in embedded systems [31]. Thus, researchers have presented several redundancy based techniques at various levels of design space abstraction, based on dual modular redundancy (DMR), triple modular redundancy, and check pointing. However, these redundancy techniques without optimization incur high overheads in terms of power, performance, and area. For example, TMR typically uses three functionally equivalent replicas of a logic circuit and a majority voter, but the overheads of hardware and power for conventional TMR exceed 200% [32]. [33] has been proposed a systematic approach for automatically introducing data and code redundancy into an existing program written using a high level language. The transformations aim at making the program able to detect most of the soft errors aecting data and code, independently of the Error Detection Mechanisms (EDMs) possibly implemented by the hardware. In [34], a new paradigm for designing logic circuits with concurrent error detection (CED) is described. The key idea is to exploit the asymmetric soft error susceptibility of nodes in a logic circuit. Rather than target all modeled faults, CED is targeted towards the nodes that have the highest soft error susceptibility to achieve cost effective tradeoffs between overhead and reduction in the soft error failure rate. Under this new paradigm, propose one particular approach that is based on partial duplication and show that it is capable of reducing the soft error failure rate significantly with a fraction of the overhead required for full duplication.

Figure 6 illustrates the possible outcomes of a single-bit fault. Outcomes labeled 1-3 indicate non-error conditions. The most insidious form of error is silent data corruption (SDC) (outcome 4), where a fault induces the system to generate erroneous outputs. To avoid SDC, designers often employ basic error detection mechanisms, such as parity. With the ability to detect a fault but not correct it, we avoid generating incorrect outputs, but cannot recover when an error occurs. In other words, simple error detection does not reduce the error rate, but does provide fail-stop behavior and thereby avoids any data corruption. We call errors in this category detected unrecoverable errors (DUE). We subdivide DUE events according to whether the detected error would affect the final outcome of the execution. We call benign detected errors false DUE events (outcome 5 in Figure 6 and others true DUE events (outcome 6). In most situations, it is impossible for a processor to determine at the time an error is detected whether it is benign. The conservative approach is to signal all detected errors as processor failures. A direct approach to reducing error rates involves adding error correction or recovery mechanisms to a design, eliminating outcomes 3 through 6 from Figure 6. Unfortunately, these mechanisms come at a significant cost in power, performance, and area [35]. In [36], present two low overhead techniques that provide scalable fault coverage as a function of the available area and power budgets. The first technique introduce the register value cache, an architectural mechanism, that provides twice the fault coverage of ECC when applied to the register le and costs less to implement in terms of both area and power. The second technique present makes use of time delayed shadow latches for fault detection. It identifies high fan-in nodes in the microprocessor core for placing these detectors and achieves up to 40% fault coverage. In conjunction, the two proposed fault tolerance techniques can provide approximately 84% fault coverage while incurring less than 5.5% area overhead and about 14% power overhead.

To make CGRAs robust against soft errors, several hardware based techniques have been proposed [9], [10], [11], [12], but they are expensive in terms of area, power, and performance. Most of hardware based techniques modify existing architectures to implement redundancy based DMR [13] and TMR [14] and they incur high costs in every design aspect. To resolve these drawbacks from hardware based techniques, researchers move attention to software based techniques that are of no area overhead [15], [16]. Recently, an interesting software based technique has been proposed but it still incurs high performance overhead mainly due to expensive voting and comparison mechanisms of TMR and DMR, respectively [16]. In fact, Lee et al. [16] has demonstrated that software implemented TMR and DMR on 6X8 CGRAs incur up to 700% and 167% performance overheads, respectively.

In order to address soft error resilient CGRAs with the least performance overhead, we propose software implemented redundancy techniques on CGRAs with selective validation mechanisms. First, we identify the expensiveness of validation mechanisms for TMR and DMR on CGRAs, respectively. Indeed, the voting overhead takes up approximately 62.5% of the total overhead in TMR and it is true since CGRAs are good at data intensive computation rather than control intensive computation such as voting operations [16]. Second, we present selective validation schemes for software based TMR and DMR on CGRAs rather than the complete validation. The main idea behind our proposals is to selectively apply voting mechanisms just before synchronous points where applications can be affected by corrupt data induced by soft errors and fail to deliver the correct results. Also, we present the comparable fault coverage of our approach as compared to the previously proposed software-based TMR technique with the full voting on CGRAs. In addition, we propose an optimization technique to reduce the synchronous points so that we can further reduce the performance overhead due to the complex voting by decreasing the number of voting mechanisms.

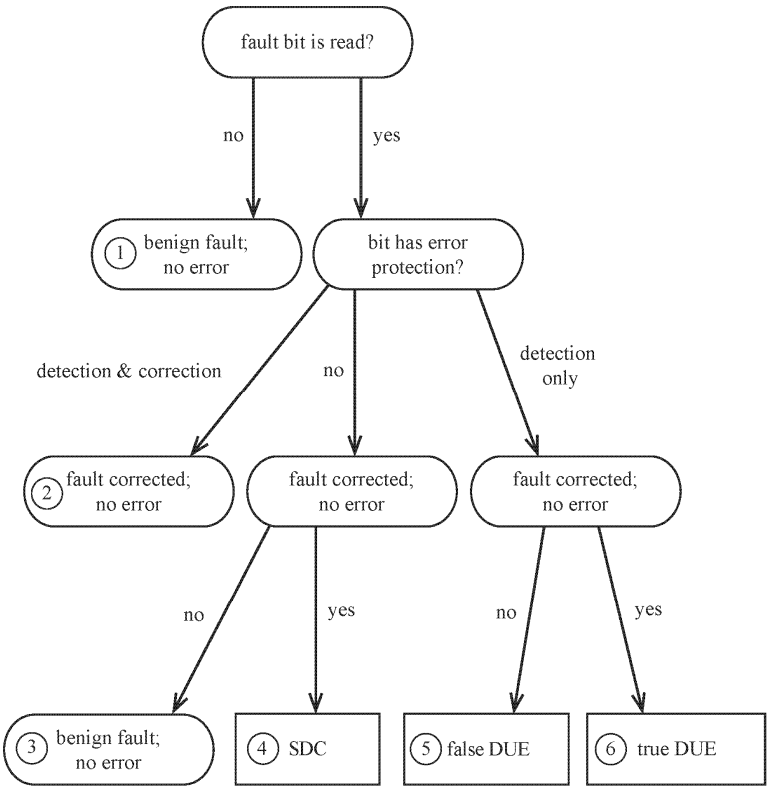


Fig. 6. Classification of possible outcomes of a faulty bit in a microprocessor (SDC = silent data corruption. DUE = detected unrecoverable error)

The contribution and results of this work include :

- Our software based TMR technique with the selective voting can improve the runtime by 38.3% and the energy consumption by 18.1% on average over benchmarks as compared to a previously proposed TMR technique with the full voting.

- Our software based DMR with the selective comparison can improve the runtime by 14.3% and the energy consumption by 3.6% on average over benchmarks as compared to a previously proposed DMR technique with the full comparison.

- Our optimization techniques can further improve the runtime by 41.0% and the energy consumption by 26.2% as compared to a previously proposed TMR technique with the full voting mechanism and by 17.8% and 14.0%, respectively, as compared to a previously proposed DMR technique with the full comparison mechanism by minimizing the occurrence of the validations with the loop unrolling scheme.

- Our software based protection techniques with the selective validation mechanism show the fault coverage as comparable as recent proposals with the full voting mechanism by quantitative analysis.

Ⅱ. Related Work

Soft errors are becoming a critical design concern as technology scaling continues and CGRA is being employed in critical applications such as aircrafts, space missions, and financial systems [9]. Thus, the reliability on CGRAs against soft errors is emerging as an important research topic but the literature is relatively small. Most of these studies proposed redundancy techniques such as DMR and TMR by exploiting identical blocks or processing elements for the replications to protect the datapaths on CGRAs against soft errors.

Previously proposed fault tolerant techniques for CGRA are widely classified with hardware and software based techniques as shown in Table 1. Hardware based techniques (grey shade ones in Table 1) incur significant area overhead for implementing fault tolerant techniques. To resolve this area overhead, software based techniques (yellow shade ones in Table 1) that is not occurred area overhead are propose. Although software based techniques reduce the area overhead, they incur performance overhead.

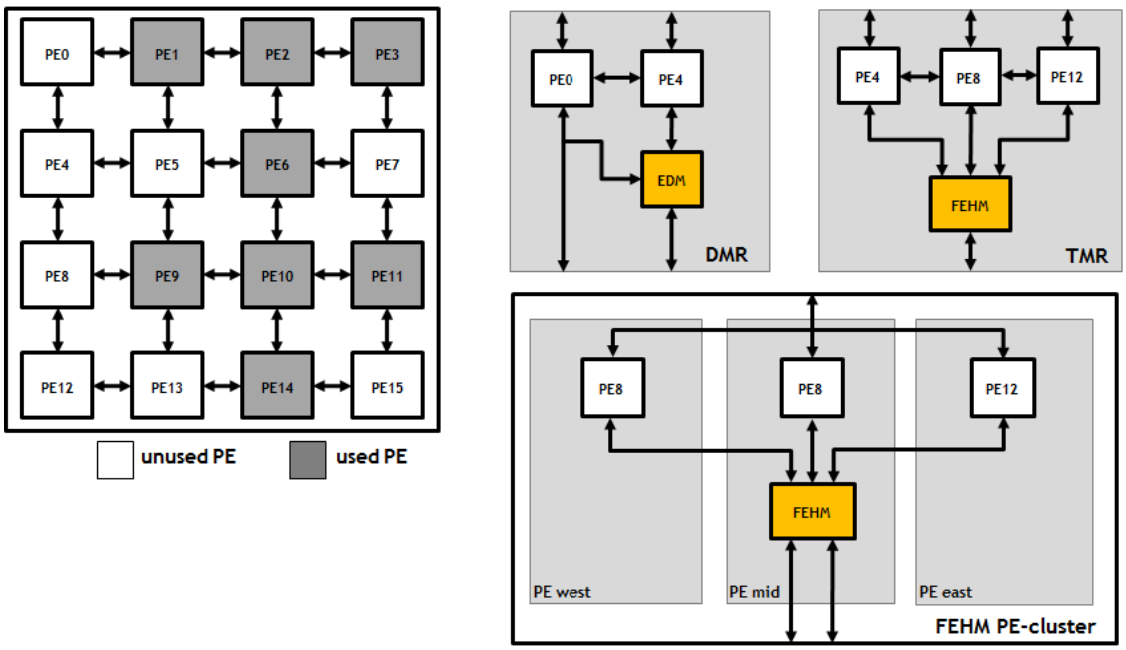
Alnajiar et al. [10] proposed dynamic operation modes in CGRA architecture to provide the various levels of reliability under the performance constraint. However, their technique incurs 26.6% area overhead mainly due to additional hardware redundancy and causes performance degradation because cluster-based architectures cannot fully use hardware resources. To reduce this hardware overhead, Jafri et al. [9] presented an alternative hardware-based redundancy technique, residue mode less costly than DMR to detect soft errors. They implemented self-checking residue mode for multiplication and addition operations on DART architecture [17], but it cannot be applied to logical operations. Recently, Eisenhardt et al. [18] proposed the remapping engine process designed and suitable for permanent faults, not soft errors.

On the other hand, researchers have investigated different approaches from the previously proposed techniques that redesign and modify architectures of processing elements on CGRAs to reduce the hardware cost. Kim et al. [19] observed that not all processing elements are exploited at the execution time mainly because some of processing elements are used for the routing of operands between producing and consuming operations. Based on this observation, Schweizer et al. [12] proposed techniques exploiting unused FUs for replications to increase the reliability with the minimal hardware overhead as illustrated in Figure 7. They proposed FEHM (Flexible

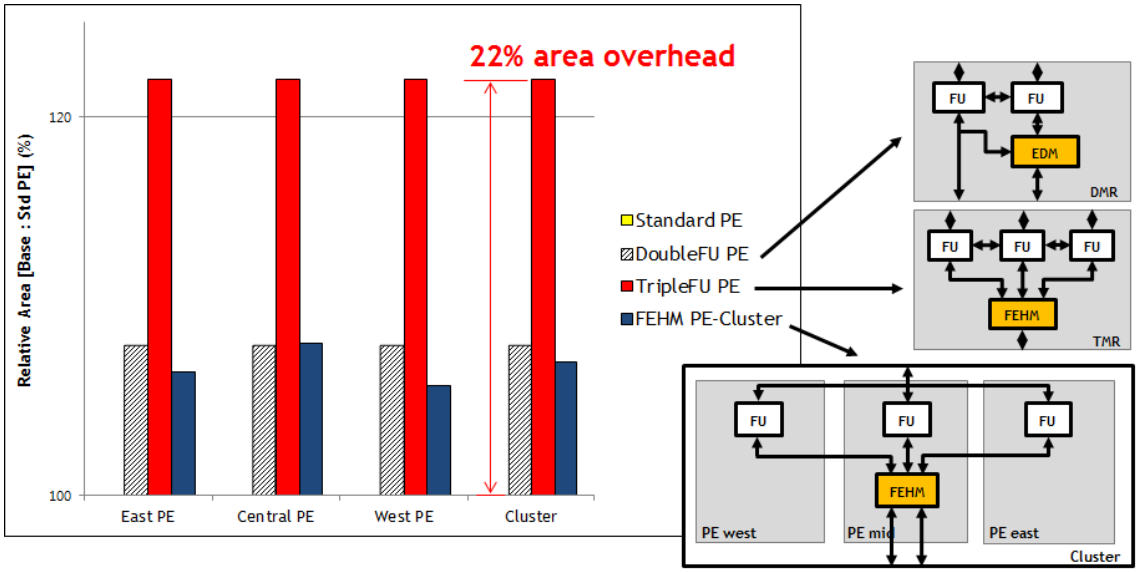
Error Handling Module) that supports DMR and TMR schemes on specific target architectures as illustrated in Figure 7(a). However, data intensive application cannot map all the operations to processing elements due to insufficient unused FUs and their experimental results show that there are still significant area overhead as illustrated in Figure 7(b). To resolve this limitation, they introduced multiple contexts to be mapped on CGRA by using the concept of temporal redundancy [11]. However, the increased number of the contexts incurs 12% performance degradation and there still remains unresolved hardware overhead. In short, previously proposed hardware based techniques incur additional area cost since they need to modify existing CGRA architectures to implement redundancy techniques such as TMR and DMR.

[Paper] & Key Idea	Experiment		drawback & Comment
Alnajiar et al. [10] dynamic operation modes in CGRA architecture to provide the various levels of reliability under the performance constraint.	set up	- compare to the number of gate (using tool : RTL)	- flexible protection mechanism - **area-overhead (implement voter)** - performance degradation (cant use resource fully)
	result	- area : 26.6% increase	
Jafri et al. [9] self-checking residue mode for multiplication and addition operations on DART architecture	set up	- compare to original FU, DMR FU and self-checkingFU	- less area overhead - **area-overhead (implement self-checking)** - performance degradation - only detection - not cover specific fault
	result	- area : 18% decrease compare to DMR - performance : 400% decrease compare to DMR	
Schweizer et al. [12] exploiting unused FUs for replications to increase the reliability with the minimal hardware overhead. FEHM (Flexible Error Handling Module) - supports DMR and TMR schemes.	set up	- compare to TMR and Clustering PE include FEHM	- supports DMR and TMR schemes with modified FEHM - considering power - **area-overhead (implement FEHM)** - cant apply to data intensive application
	result	- area : 12.8% decrease compare to TMR -power : 1.6~18.6%decrease compare to TMR	
Schweizer et al. [11] to resolve previous ([Schweizer, 2011]) limitation, multiple contexts to be mapped on CGRA by using the concept of temporal Redundancy	set up	- mapping to CGRA usedFFT application - estimate required area as context memory is increased - Estimate time between read and store	- enable to apply permanent, transient, timing fault - enable to apply any application - performance degradation compare to TMR (increase context) - **still exits area-overhead**
	result	- area : 31% decrease compare to TMR - performance: NR/TMR 26%/12% decrease	
K. Singh et al. [15] Selective apply combined scheme to code that can cause a soft error.	set up	- reliability is the percentage of time that FT matrix multiplication can run on raw architecture without system reset	- no area overhead (sw-based technique) - limited RAW architecture - **performance overhead** - not consider TMR voting
	result	- reliability : 89.2% (108out of 1000 reset)	
Lee et al. [16] replication &voter is implemented on PE to reduce area overhead;to reduce the critical path, addconditional execution & column wide bus;thermal impactoptimal mapping on PE is proposed.	set up	- comparebase arch with proposed arch based on RTL - ACS (time consuming operations) module in viterbi decoder map CGRA	- software technique - area-efficiency (minimal hardware overhead) - **performance overhead** (replication& voter map on PE)
	result	- area : 12% increase compare to base - performance: decrease TMR(700%)/DMR(167%)	

Table 1. Previously proposed techniques for CGRAs



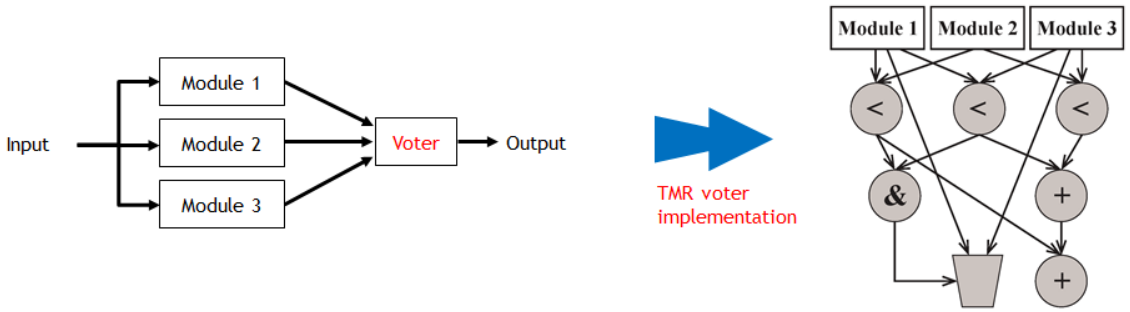
(a) FEHM (Flexible Error Handling Module)



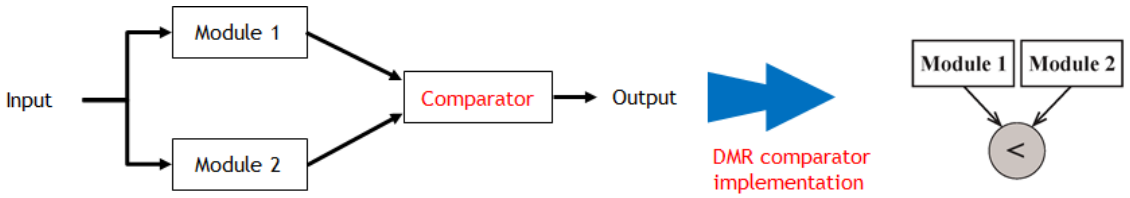
(b) Experimental results

Fig. 7. Hardware based techniques for fault tolerant CGRAs [12]

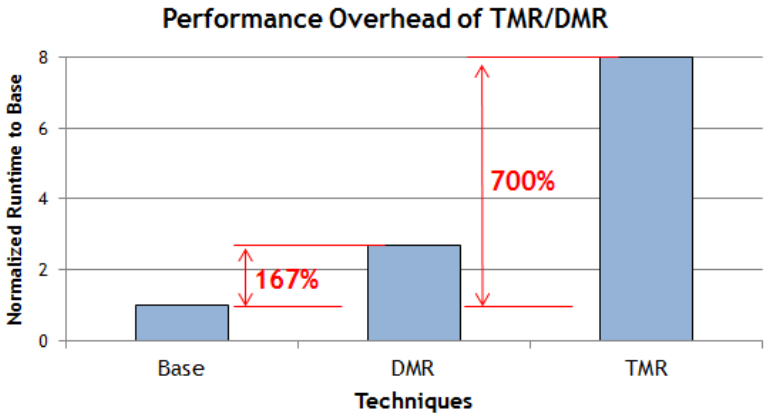
In order to overcome this area overhead, recent researchers have investigated software based techniques to implement redundancy techniques without hardware modifications [15], [16]. Singh et al. [15] presented fault tolerance techniques for an existing Raw architecture [20] by exploiting the selective redundancy and checkpoint schemes. However, their scheme is inapplicable to any CGRA architecture since it is designated only for RAW architecture and also causes performance degradation. As a general software-implemented technique applicable to any CGRA architecture, Lee et al. [16] proposed software based TMR and DMR techniques by mapping software implemented replicas of operations and validation mechanisms onto processing elements in CGRA architectures as shown in Figure 8. However, they still incur high performance overhead mainly due to additionally mapped processing elements for the complex voting and comparison mechanisms. These software techniques offer limited amount of area cost than hardware techniques but result in significant performance degradation.



(a) Software implemented TMR techniques on CGRAs



(b) Software implemented DMR techniques on CGRAs



(c) Experimental results

Fig. 8. Software based techniques for fault tolerant CGRAs [16]

We propose novel selective validation schemes to improve performance without any hardware modification. Our proposals can remove the area cost by exploiting software based techniques and fulfill the performance improvement by selectively applying the validation mechanisms only on synchronization points before store operations. This approach makes sense since modified outputs from processing elements (which will be written back to the memory) can affect the application kernel and its final output at the end unless they are fixed before the memory update [21].

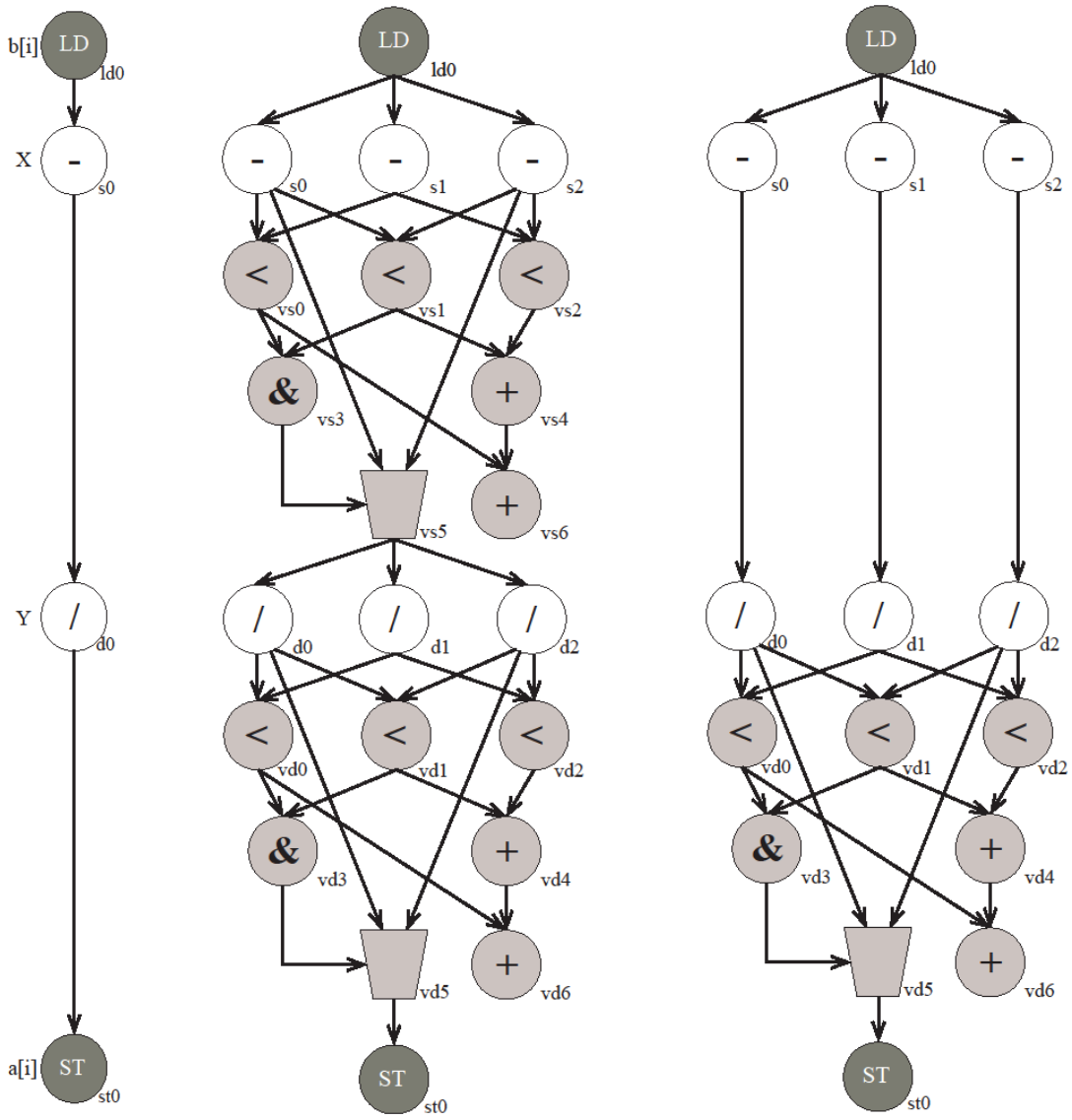
Ⅲ. Motivation

CGRA is essentially an array of processing elements or PEs connected through a mesh-like or interconnection as illustrated in Figure 1. A PE generally consists of a functional unit (e.g., ALU, shifter, multiplier, etc.) and a small register file for storing temporary variables and constant values. The PE array consists of heterogeneous PEs and basic operations such as arithmetic and logical operations are performed by every PE whereas the costly operations such as multiply and memory access operations are performed only by some PEs. Like Field-Programmable Gate Arrays (FPGAs), the functionality of PEs and the data flow among PEs are controlled by configuration. However, as the configuration size for CGRA is small since CGRAs are controlled in a word-level operation, CGRAs can be reconfigured very fast, even in every cycle [22], unlike FPGA configured in a bit-level operation.

Figure 9 shows an example of software-implemented TMR for a kernel part. Figure 9(a) presents C-like pseudo code for the kernel part (a[i] = (b[i] - X) / Y). To implement this kernel, the original DFG (Data-Flow Graph) is composed of load, subtract, divide, and store operations as shown in Figure 9(b). Figure 9(c) draws its TMR implementation as a DFG form. In this example, normal operations (except memory operations) such as subtract and divide must be triplicated and their results are validated by the voting mechanism. For instance, a subtract operation (the original node, s0) is triplicated (two additional nodes s1 and s2 for the triplication) and the voting requires three compare (vs0, vs1, and vs2), one logical and (vs3), two add (vs4 and vs6), and one select (vs5) operations (7 additional nodes for the voting). A TMR implementation requires 9 additional nodes per normal operation, which is translated into the huge impact on the performance. For this simple kernel, TMR implementation has increased the number of nodes from 4 to 22 and the number of edges from 3 to 35 (compare Figure 9(b) and Figure 9(c)). These increased numbers of nodes and edges increase the complexity of the operations to be mapped onto PEs causing more challenges to the compilation scheduling. Eventually, they degrade the performance due to highly required PEs and tightly induced data dependency among them.

```
for ( i = 0;  i < iteration;  i ++ ) {
  /* X and Y are constant */
  a[i] = ( b[i] - X ) / Y ;
}
```

(a) Example of a kernel



(b) Base (c) TMR with full voting (d) TMR with selective voting

Fig. 9. Generated DFGs of Base (No Redundancy), Software implemented TMR with the full voting [16], and TMR with the selective voting for a kernel

In order to observe the performance overhead of the voting mechanism, we have run a simple experiment. First off, we have evaluated the performance in terms of runtime for base kernels (i.e., without any redundancy) of benchmarks. Secondly, the runtime has been estimated for software based TMR implementation on CGRA and its performance overhead has been calculated in percentage by dividing the difference between runtime of the base and that of the TMR by that of the base ($O_{TMR} = (R_{TMR} - R_{Base}) \div R_{Base}$ where O_{TMR} is its performance overhead, and R_{Base} and R_{TMR} are runtime for the base and TMR with the full voting, respectively). Then, we have implemented the DFGs of triplicated operations of benchmark kernels without the voting, and evaluated the runtime and its performance overhead ($O_{TMR\_no\_vote} = (R_{TMR\_no\_vote} - R_{Base}) \div R_{Base}$ where $O_{TMR\_no\_vote}$ is its performance overhead and $R_{TMR\_no\_vote}$ is runtime for TMR without the voting.). Last, we estimate the voting overhead as the difference between O_{TMR} and $O_{TMR\_no\_vote}$, and Figure 10 draws the portions of the voting overhead ($O_{TMR} - O_{TMR\_no\_vote}$) and the triplication overhead ($O_{TMR\_no\_vote}$). Figure 10 shows that the performance overhead caused by the voting mechanism in software-implemented TMR takes up about 64.8% on average over the benchmarks. This high performance degradation results from the high data dependency and complexity of the voting mechanism. In this thesis, we investigate selective validation techniques to reduce this expensive performance cost in redundancy protections on CGRAs.

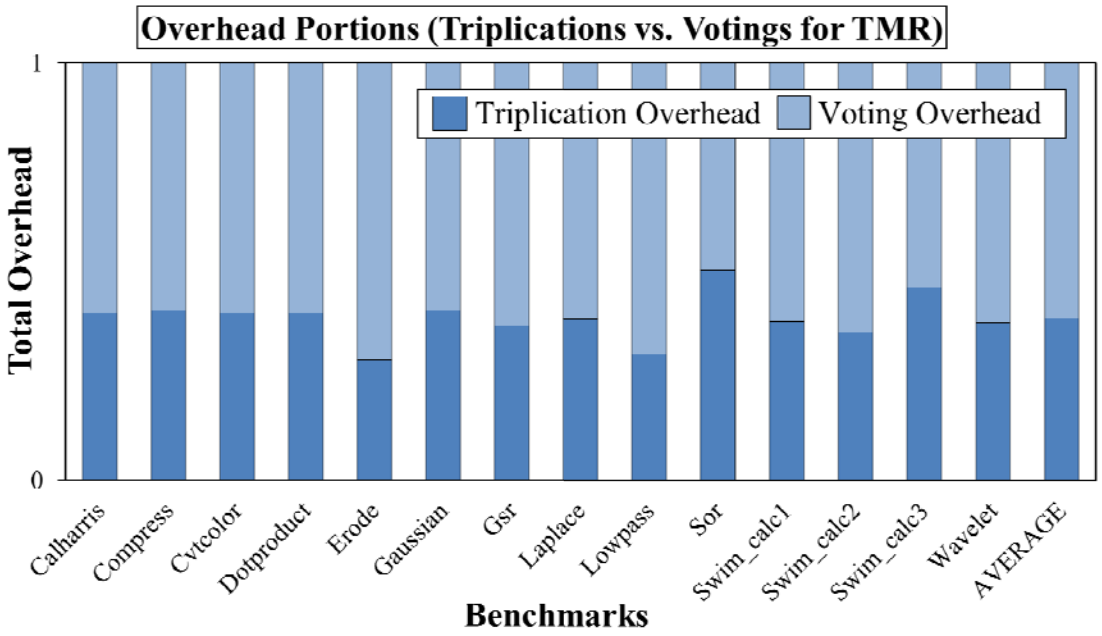


Fig. 10. Runtime overhead of voting overhead takes up about 64.8% in software implemented TMR techniques on CGRAs.

Ⅳ. Our Approach

A. Selective Validation Mechanism

In order to improve the reliability with minimal performance overhead, we present the selective validation techniques for TMR and DMR on CGRAs. Our goal is to protect the datapath of CGRAs such as FUs from soft errors. Traditional hardware-based protection methodologies for memory subsystem are inexpensive as compared to maintaining double- or triple-redundant execution cores [21]. Therefore, we suppose that memory of CGRA architectures is protected against soft errors by traditional fault tolerant techniques such as parity check, ECC (Error Correction Code), and scrubbing. Thus, we do not replicate the memory operations such as “load” and “store” and do not validate the output of memory operations, which can reduce the performance overhead.

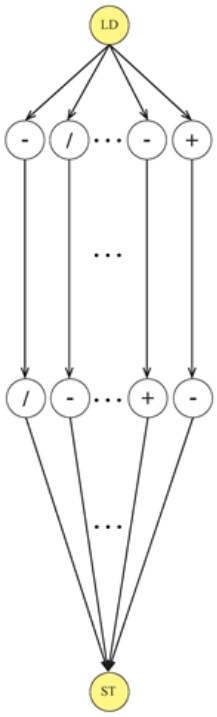
Our main goal is to reduce the number of validations for improving performance without losing the reliability as compared to the existing redundancy techniques. Note that the main benefit of CGRAs is to map the kernel part of applications to accelerate the performance that is data intensive kernel as in operations in the loop. Thus, the control part of applications is not suitable for being mapped onto the PEs in CGRAs since it incurs unnecessary performance overhead. Indeed, the validation mechanisms such as the voting mechanism for TMR and the comparison for DMR are sort of control intensive operations which are inappropriate for operations mapped onto PEs in terms of the performance. The main idea behind our approach is to perform validation operations just before synchronization points where the program can be affected and result in incorrect output or even catastrophic consequences if the data is incorrect after synchronization points [21], [23]. For example, store operations have been committed to the memory with erroneous states and these erroneous results can eventually cause incorrect outputs of an application. Thus, the program will be executed correctly if corrupted data is not stored in the main memory. Indeed, the concept of this selective validation approach has been introduced through the technique named SWIFT (Software-Implemented Fault-Tolerant) [21], [23]. In this technique, all instructions other than memory instructions are replicated and the validation checks are introduced only at certain synchronization points to ensure that the data produced by the original and replicated operations are identical or correctable. Note that their technique can achieve the reliability by 97% of that of TMR with the full validation [23].

Assume that the number of operations are 100 as shown in Figure 11(a). In TMR

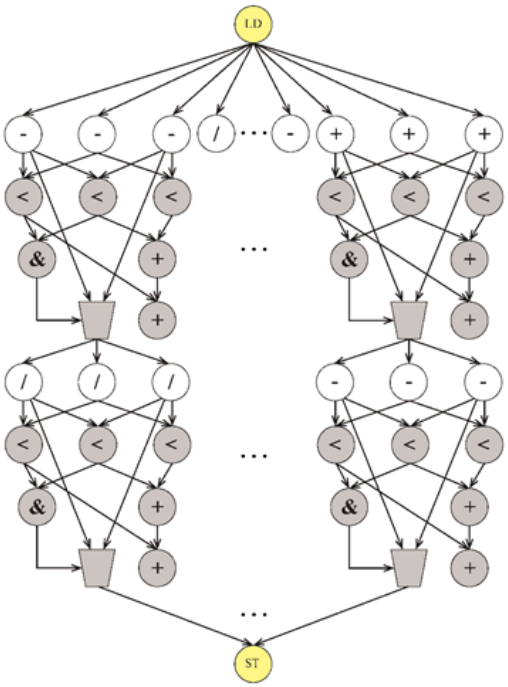
with full voting, 300 operations are required to triplication and 700 operations are added for voting mechanism. In short, the total number of operations we need is 2000 (300+700+300+700) as shown in Figure 11(b).

On the other hand, 300 operations are required to triplication like Figure 11(b). and then no operations are added for voting mechanism because our approach vote at synchronization points such as store operation. In short, the total number of operations we need is 1300 (300+0+300+700) as shown in Figure 11(c).

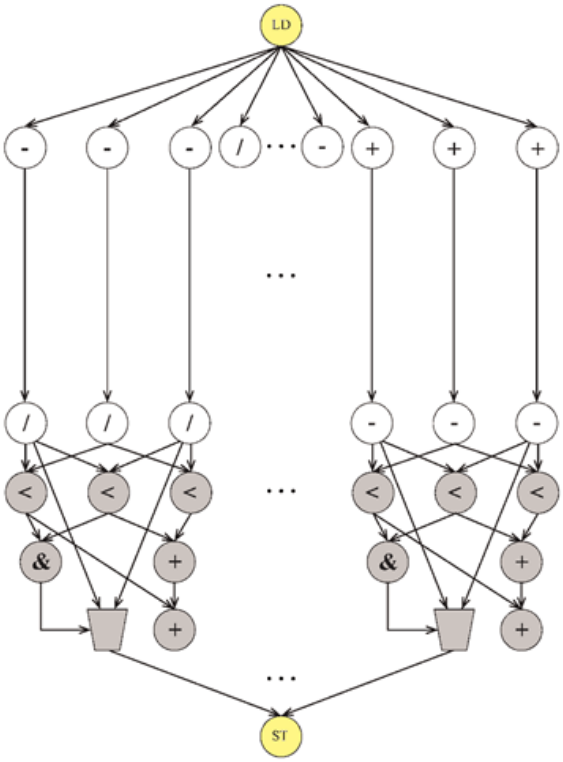
Figure 12 is the DFG from Swim_calculation which one of the benchmark suites. Figure 12 clearly shows efficiency of our selective validation for improving performance. Figure 12(a) is base with no protection techniques. Figure 12(b) is TMR with full validations. Original operations (A1, A2, A3, A4 and A5) are triplicated and voted but ST1 is not validate because memory of CGRA architectures is protected against soft errors by traditional fault tolerant techniques such as parity check, ECC (Error Correction Code), and scrubbing. However our selective validation techniques as shown in Figure 12(c) check the validations only at certain synchronization points(ST2) to ensure that the data produced by the original and replicated operations are identical or correctable. Selective validation techques incur less performance overhead than conventional TMR technique.



(a) Base

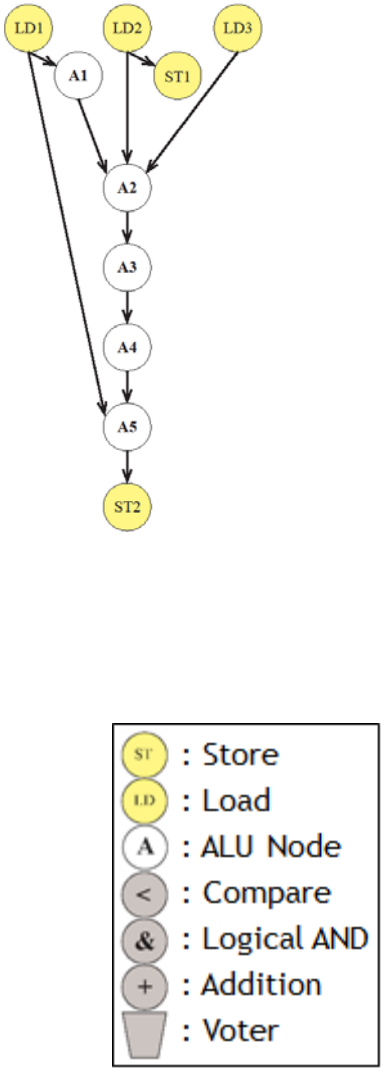


(b) TMR with full voting

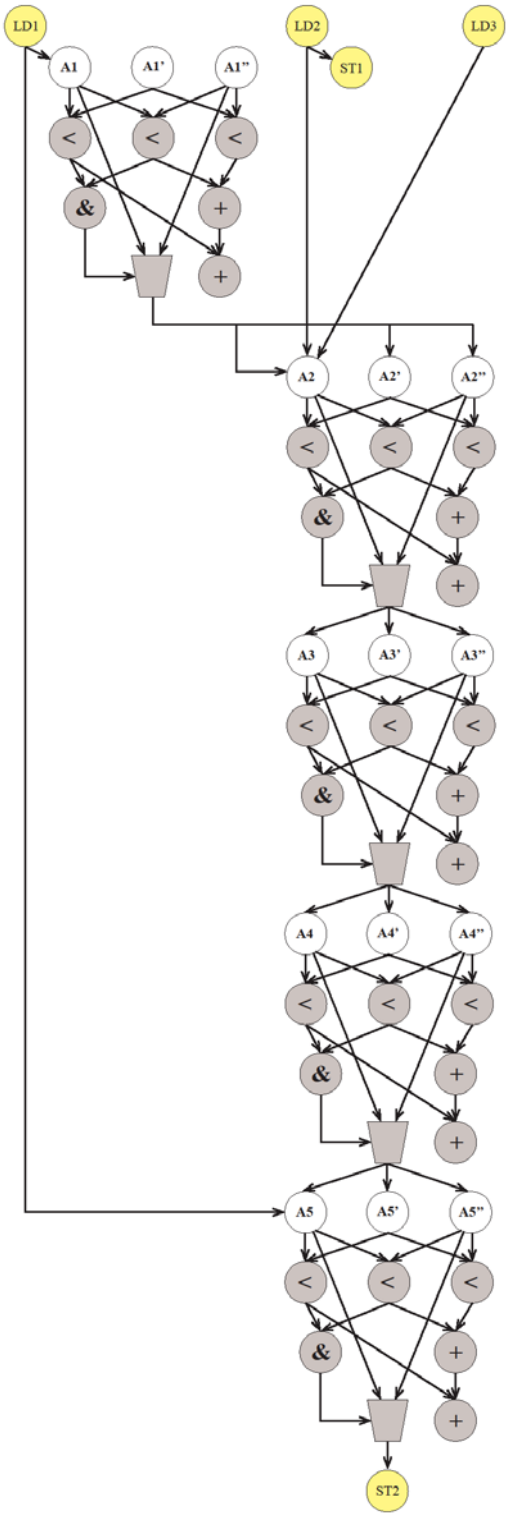


(c) TMR with selective voting

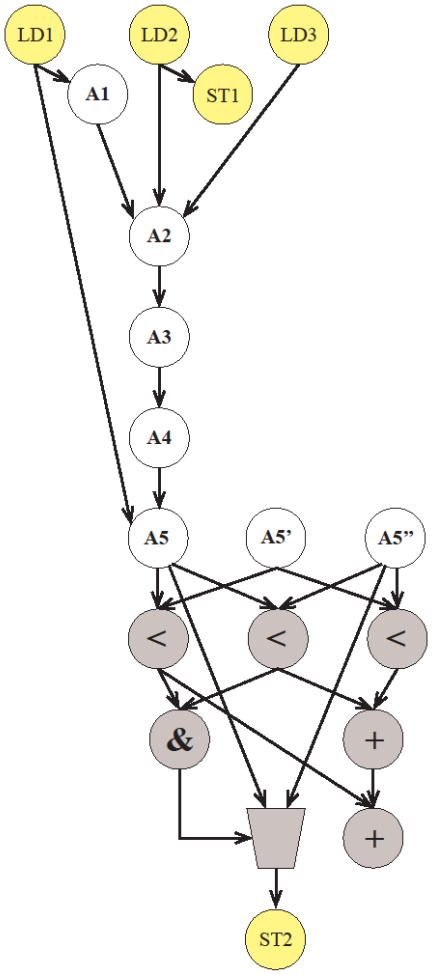
Fig. 11. Examples of reducing expensive voting



(a) Base
in Swim_calculation benchmark



(b) TMR with full voting
in Swim_calculation benchmark



(c) TMR with selective voting in Swim_calculation benchmark

Fig. 12. Expensive Voting mechanisms (Benchmark: Swim_calculation)

B. Compilation Flow and Performance Analysis

Figure 13 shows the overall compilation flow for a system including CGRA as an accelerator or coprocessor. First, an application is partitioned to extract kernels to be mapped onto CGRAs. Then, the extracted kernels are compiled for CGRA while the rest of the code, i.e., sequential code, goes through the conventional compilation process. CGRA compilation starts from constructing the DFG of a loop. After that, modulo scheduler takes the DFG as an input to generate a valid mapping result for executing the loop on CGRA. Modulo scheduling [24] is a software pipelining technique that exploits the parallelism by overlapping consecutive iterations of the loop. The goal is to find a valid schedule with a minimized initiation interval (II), which is the difference between the start times of successive iterations. Minimizing the II leads to the throughput improvement since one loop iteration takes II cycles ignoring the effects of prologue and epilogue of the loop. Modulo scheduler first initializes the II by taking the maximum out of the resource-constrained lower bound (ResMII) and the recurrence-constrained lower bound (RecMII). It then attempts to generate a valid schedule within the minimal II. If no valid schedule can be found for the given II, the scheduler increments II by one and attempts again until a valid schedule is achieved.

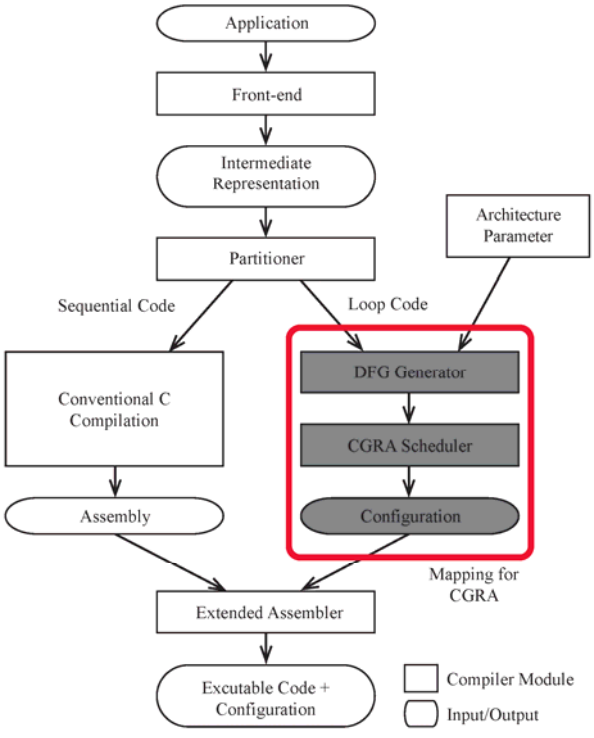
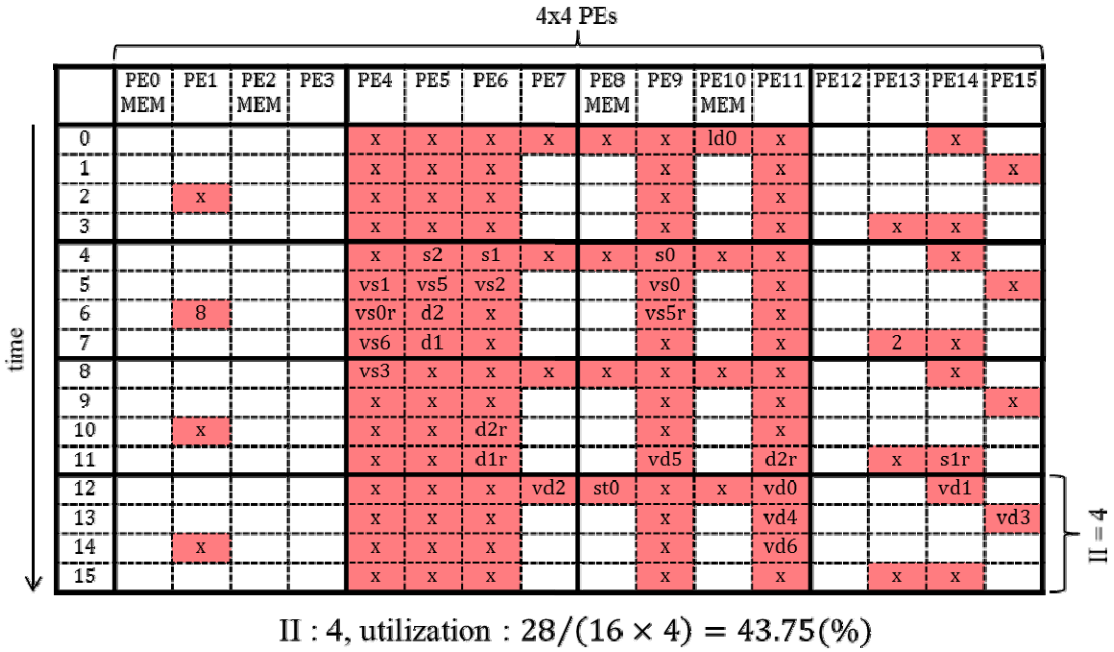


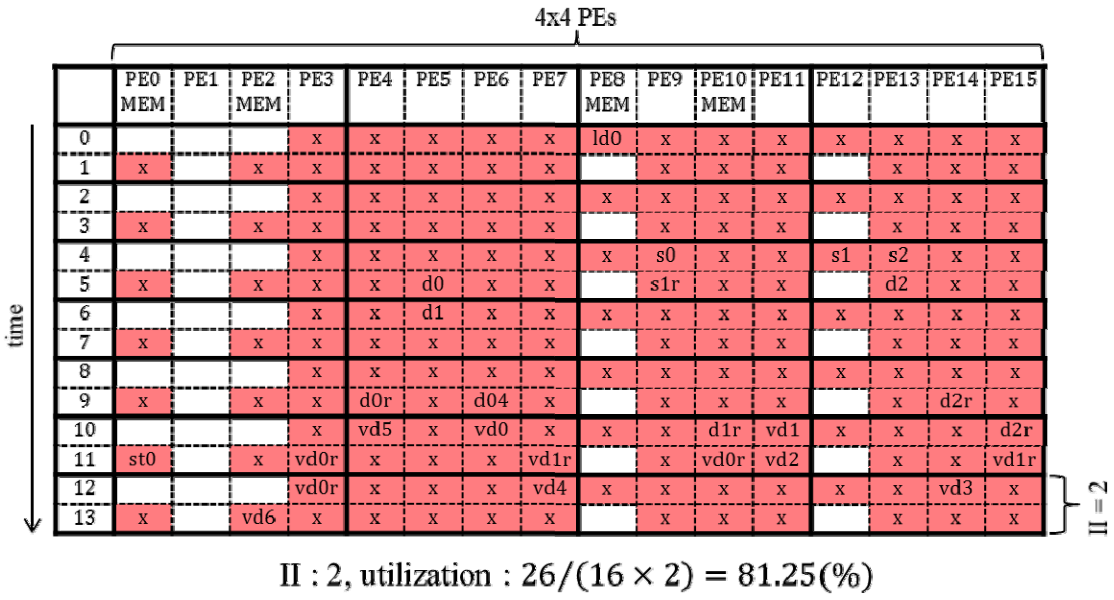
Fig. 13. Compilation flow for a system with CGRA

Figure 9 shows the generated DFGs through this compilation flow for the original code (Figure 9(b)), for the TMR code with the full validation (Figure 9(c)), and for the TMR code with the selective validation (Figure 9(d)). Our TMR with the selective validation introduces just one validation computation as shown in Figure 9(d) while TMR with the full validation introduces two validation computations as shown in Figure 9(c). Thus, we can reduce one set of operations for the validation after the triplicated operation (subtract operation in this example), and this reduction can improve the performance.

Figure 13 illustrates the effectiveness of selective voting technique in terms of II and the utilization with a mapping example. Each DFG is scheduled onto a 4 × 4 CGRA according to compilation flow in Figure 13. In the scheduled results, the ID at each cell in Figure 14 indicates the mapping of an operation from the DFG as shown in Figure 9. For instance, ‘s0’ is scheduled onto PE 9 at the cycle 4 in Figure 14(a). The IDs followed by ‘r’ (e.g., ‘vs0r’) indicate routing operations for the corresponding computation operations. For example, ‘vs0r’ is scheduled onto PE 4 and at the cycle 6 in Figure 14(a) for the routing of ‘vs0’. Slots marked with ‘X’ represent ones occupied due to the modulo constraint. Assume that the latency of a load operation is three cycles and other operations one cycle in our scheduling framework. Figure 14(a) shows the scheduling result for TMR with the full voting consisting of 22 nodes and 35 edges from the DFG in Figure 9(c), and its performance output with II=4. However, our selective voting technique can construct the DFG with less nodes and edges (15 nodes and 21 edges from Figure 9(d)) and thus the II from the scheduled result is equal to 2 (as shown in Figure 14(b)), which can be interpreted about 2 times improvement in performance since the performance is roughly proportional to the II. Interestingly, we can utilize the PEs of CGRA approximately 2 times more efficiently with the selective voting than the full voting. Note that the better utilization can avoid unnecessary waste of CGRA resources and lead to the performance efficiency. Therefore, our selective voting technique can improve II and archive high utilization ratio due to the reduced number of nodes and edges. The number of nodes and edges in the DFG affects several aspects as follows. First, the number of nodes implies the least required number of PEs in CGRA architectures. Second, the increased number of edges in general raises the data dependency among connected PEs. Our selective validation techniques can improve the performance with the reduced II by decreasing the number of nodes and edges in the DFG and by exploiting the unused FUs efficiently by reducing the data dependency.



(a) TMR Implementation with full voting from Figure 9(c)



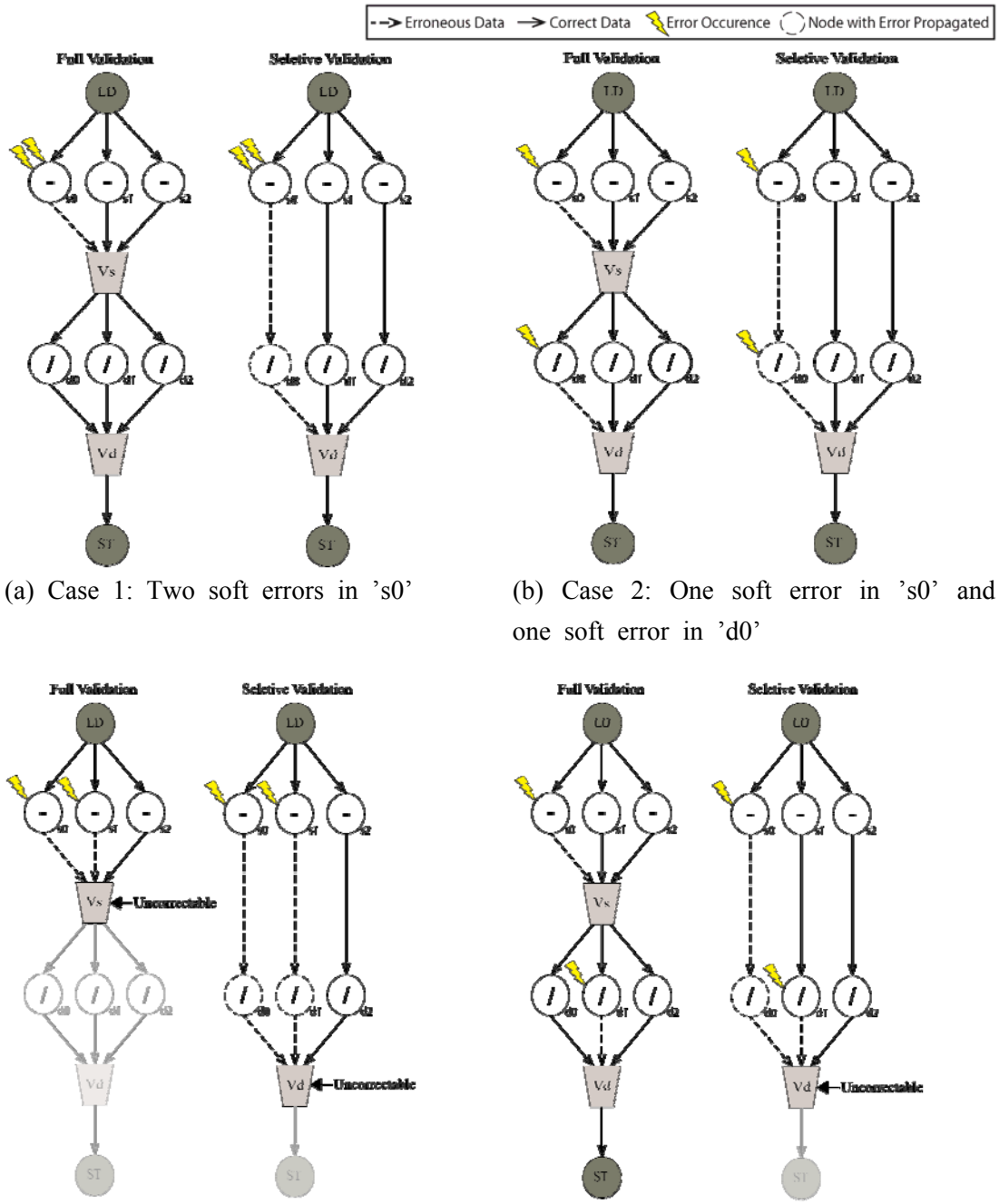
(b) TMR Implementation with selective voting from Figure 9(d)

Fig. 14. Mapping operations for software implemented TMR onto CGRAs

C. Fault Coverage Analysis

Our redundancy techniques with the selective validations on CGRAs can achieve the comparable reliability in terms of the fault coverage as compared to the redundancy techniques with the full validations. Fault coverage can be defined as the ratio of the detected number of faults to the total number of faults. Suppose that the fault coverage of considering both single bit soft errors and multiple bit soft errors is FC = α * FC_{SBSE} + β * FC_{MBSE} where FC_{SBSE} is the fault coverage for single bit soft errors, FC_{MBSE} is the fault coverage for multiple bit soft errors, and and are weight constants for FC_{SBSE} and FC_{MBSE}, respectively. If we consider a single bit error for the whole operation of the kernel, FC_{SBSE} for TMR with the selective validation is equal to that for TMR with the full validation since a soft error induced incorrect value will be eventually corrected at the synchronous point by the validation, which does not cause data corruption or system failure. Thus, when α is set to 1 and β is set to 0, FC for our technique is the same as that for a conventional TMR technique.

On the other hand, if we consider double bit soft errors as multiple ones, which has extremely lower error rate than single bit soft error (100 times less [25]), 4 cases should be taken into account for the fault coverage analysis as described in Figure 15. The first case is that double bit errors occur at the same operation in the datapath at the same cycle as shown in Figure 15(a). These errors should be fixed by both techniques, i.e., our selective validation and the full validation since no erroneous datapaths in nodes s1 and d1, and s2 and d2 will mask the error propagated to d0 from s0 at Vd in our selective validation as shown in Figure 15(a). Thus, the first case results in the same fault coverage. The second case is that double bit errors occur at different operations in the same datapath at different cycles (Figure 15(b)) and then these errors also can be fixed by both techniques since operations at the other datapaths will be executed correctly without errors. Thus, the second case also results in the same fault coverage. The third case is that double bit errors occur at the same operations in two different datapaths at the same cycle (Figure 15(c)) and then these errors cannot be validated by both techniques since the 2-out-of-3 voting may not work to mask these errors. Thus, the third case results in the same missed fault coverage. The last case is that double bit errors occur at different operations in two different datapaths at different cycles (Figure 15(d)) and then these errors will be corrected by the full validation (since each single bit error can be fixed just after each operation has been committed) but these errors may not be masked by the selective validation. Thus, the last case results in the loss of the fault coverage for the selective validation. Thus, when α is set to 0 and β is set to 1, FC for our technique is worse than FC for TMR with the full validation on CGRAs.



(a) Case 1: Two soft errors in 's0'

(b) Case 2: One soft error in 's0' and one soft error in 'd0'

(c) Case 3: One soft error in 's0' and one soft error in 's1'

(d) Case 4: One soft error in 's0' and one soft error in 'd1'

Fig. 15. Fault coverage analysis of software implemented TMRs in case of double soft errors (Shaded nodes and edges indicate no executions in case of uncorrectable validations at (c) and (d))

Assume that double bit soft error rate is considered 100 times less than single bit soft error rate. If we suppose that all multiple bit soft errors are double bit soft errors and the last case (worse fault coverage case for our technique) takes up the whole possibility out of four cases, the FC for the selective validation is less than 1% than that for the full validation in TMR, which is the upper bound of the worse fault coverage for our case even in considering various weight constants between 0 and 1 for α and β. In conclusion, our technique can achieve the same fault coverage for single bit soft errors and the comparable fault coverage for multiple bit soft errors (at most 1% worse with the current ratio of single bit soft errors to double bit soft errors) as compared to the previously proposed TMR techniques with the full validation. Note that our fault coverage analysis excludes the cases where soft errors occur on the PEs for the validation mechanisms together. However, an error at the validation cannot guarantee the reliability for both the full and selective validation techniques.

D. Our Optimization : Minimizing Store Operation

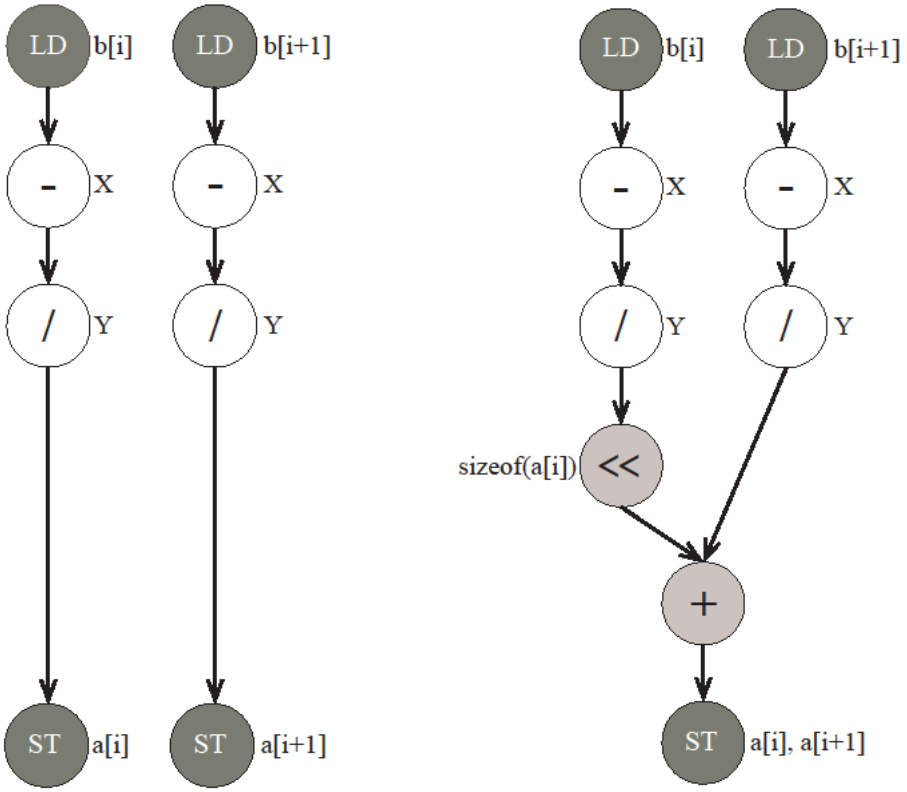
To further improve the performance, our optimization technique merges multiple store operations into one store operation by applying the loop unrolling and modifying the DFG. As illustrated in Figure 16, the original loop unrolling can duplicate the DFG to improve the performance. Assume that the data in the same array are stored in adjacent addresses in the memory. Our idea is that the data in adjacent locations will be stored at one access after merging two store operations into one by applying shift and add operations. For example, a[0] is set to 0x12 and a[1] is set to 0x34 in our example as shown in Figure 16(a). If the unit size of an array in this example is 1 byte while the variables are of two bytes, a[0] will be shifted by 8. And then a[1] (0x0034) will be added to this shifted value of a[0] (0x1200). Finally, the sum of a[0] and a[1] (i.e., 0x1234) will be stored by just one store operation as shown in a form of the DFG in Figure 16(c).

After applying our optimization technique, the number of store operations can be halved. Therefore, the number of validations also can be reduced in a half so it can improve performance. However, there are two limitations in our optimization technique. First, our optimization requires additional PEs for mapping operations such as shift and add operations. However, the number of PEs required by the voting mechanism that is 7 greater than that of these additional PEs. Second, our optimization introduces the dependency between unrolled loops. In the original loop unrolling, each unrolled loop can be executed in parallel. In contrast, our approach requires the sum total between the results of these unrolled loops. Therefore, our optimization techniques must be considered with unrolling factors that are number of copied loop kernel. However, determining the unrolling factor with considering CGRA architectures and property of kernels is beyond our scope. Since software-implemented voting requires much more additional nodes than comparison, our optimization technique has the strength in triplication case, rather than duplication.

Note that our optimization techniques cannot be applicable for the kernel that has recurrent loops. The output of the previous iteration is required as the input of current iteration, so it cannot be stored at the same time. If our CGRA architecture exploits the reuse edge proposed in [26], the output of previous data can be used before store operation. However, it must be required the performance overhead tradeoff between voting and exploiting reuse edge techniques. The optimization techniques for recurrent loops are definitely one of our future works.

```
for ( i = 0;  i < iteration;  i += 2 ) {
    /* X and Y are constants */
    a[i] = ( b[i] - X ) / Y ;
    a[i+1] = ( b[i+1] - X ) / Y ;
}
```

(a) Example of a kernel



(b) Conventional loop unrolling (c) Modified loop unrolling

Fig. 16. Generated DFGs of our optimization technique with the loop unrolling

Ⅴ. Evaluations

A. Experimental Setup

To evaluate the effectiveness of our selective protection and optimization techniques, we have implemented a simulation framework. For the target architecture, we consider a CGRA that is close to the one illustrated in Figure 1. It contains a 4 × 4 PE array consisting of 4 multiplier PEs, 8 normal operations PEs, and 4 load-store PEs. Our CGRA has no shared register file, but each PE has its own register file whose entry size is 8. The local registers are used for scalar variables or routing temporary data. Each PE is connected to its four neighbor PEs, four diagonal ones and 2-hop straight ones. These CGRA configuration is the input to our framework as shown in Figure 17.

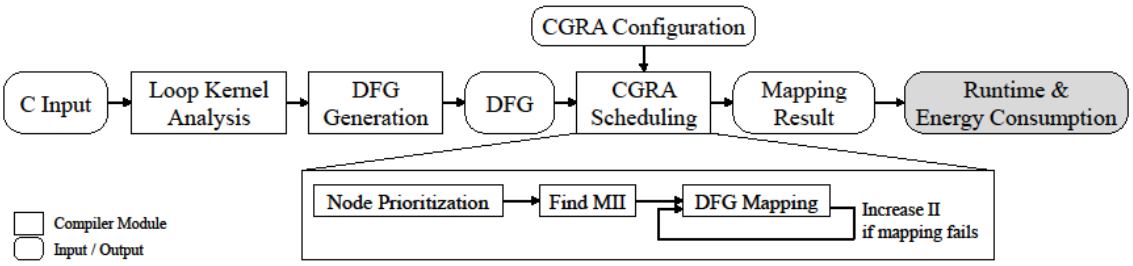


Fig. 17. Our Simulation Framework

We have taken important loops as our benchmark suite from multimedia benchmarks, OpenCV benchmarks [27] and SPEC 2000 benchmarks [28]. DFG generator creates a DFG for each benchmark kernel and this DFG information is an input to our compiler and scheduler with an initial II. Mapping and routing information of benchmarks onto CGRAs are generated using a version of modulo scheduling [26]. Due to the randomness in the cost-based scheduling algorithm (as there is more than one minimum cost candidate), we compile and simulate each benchmark kernel ten times and the result having minimum II among 10 trials is taken as the representative performance for that benchmark. Our experimental framework also returns the runtime in cycles with the minimum II.

The runtime is estimated as the sum of the prologue runtime, the kernel runtime, and the epilogue runtime. The prologue runtime RP and the epilogue runtime RE are the execution times before and after the kernel execution, respectively, and they are equal

to (s − 1) × II where II is the minimum II and s is the number of stages from our simulations. The kernel runtime R_K is calculated as (i − s + 1) × II where i is the number of iterations for the benchmark loop. The number of iterations for the store reduction i′ is calculated as I / 2 because two operations are merged by loop unrolling as shown in Figure 16. The total runtime R is represented as $R_P + R_K + R_E$.

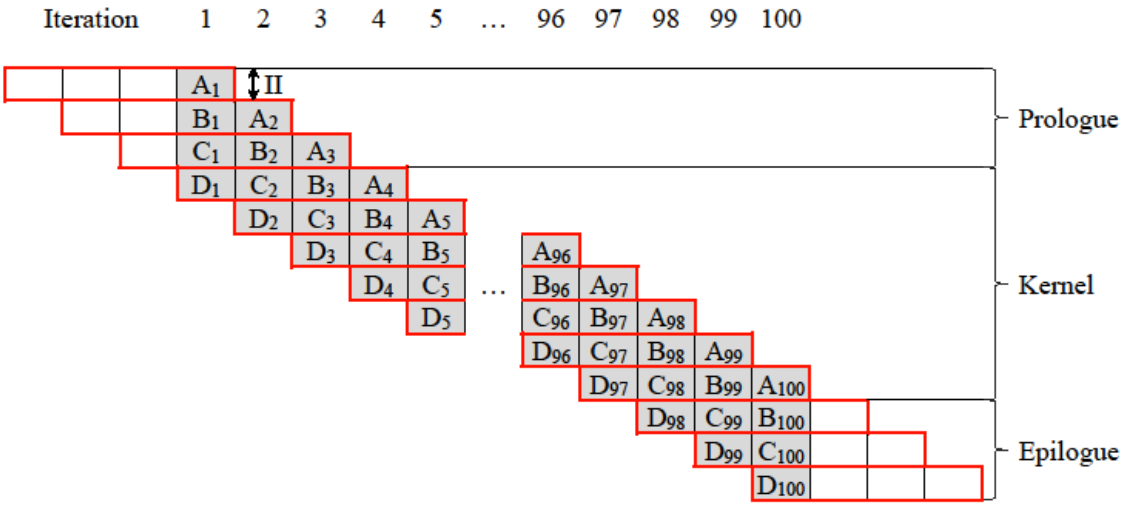
Table 2. CGRA Power Parameters

Module	Variable	Power Dissipation (mW)
Active PE (ALU)	P_{ALU}	2.543
Active PE (Multiplication)	P_{MUL}	3.200
Active PE (Division)	P_{DIV}	3.465
Active PE (Routing only)	P_{ROUT}	0.847
Idle PE	P_{IDLE}	0.254
The rest part of PE array	P_{REST}	25.988
Memory bank access	P_{MEM}	270.030
Configuration cache access	P_{CONF}	34.837

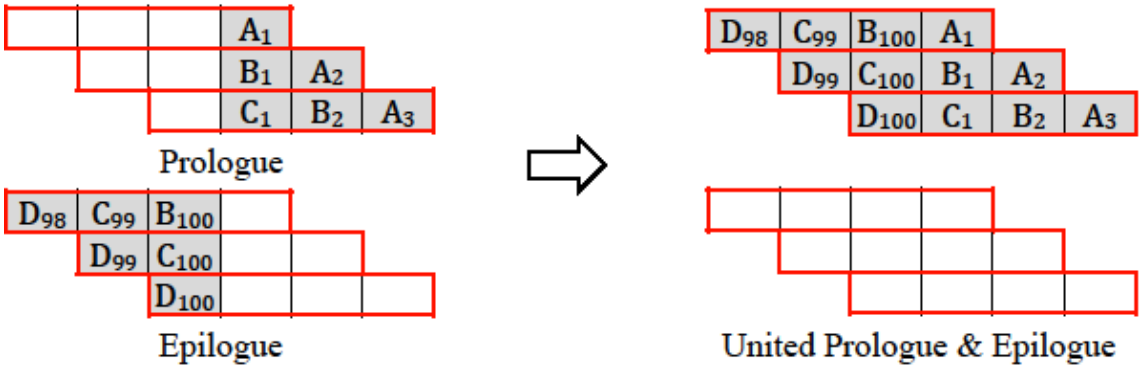
PE : Processing Element

Table 2 shows variables and dissipation values of the power for parameters [30]. For estimating the energy consumption, we need the number of each kind of node because it is the power per each node. Figure 18(a) is an example of loop level paralleled scheduling for CGRA when the number of stage is 4 and the iterations are 100. We can count the number of each kind of node in the kernel part for given II from the mapping results. In the prologue and epilogue parts, we can also count the number of each node in the same way by uniting the prologue and the epilogue as shown in Figure 18(b). The energy consumption is estimated as the sum of energy consumption for the prologue, that for the kernel, and that for the epilogue. Table 2 shows variables and dissipation values of the power for parameters. The sum of energy consumption for the prologue and that for the epilogue, E_{PE} is equal to $(s - 1) \times [\sum_{m \in O} N_m \times P_m + N_{IDLE^{PE}} \times P_{IDLE} + 2 \times II \times (P_{REST} + P_{CONF})]$ where O is a set of CGRA operations which is { ALU; MUL; DIV; ROUT; MEM }, N_m is the number of nodes for m operation, for example, N_{ALU} is the number of nodes for arithmetic and logic operation and $N_{IDLE^{PE}}$ is the number of nodes for idle nodes in the prologue and the epilogue. The kernel energy consumption E_K is equal to $(i - s + 1) \times [\sum_{m \in O} N_m \times$

$P_m + N_{IDLEk} \times P_{IDLE} + II \times (P_{REST} + P_{CONF})]$ where N_{IDLEk} is the number of nodes for idle nodes in the kernel. The number of iterations for the store reduction i′ is calculated as i / 2 for reasons mentioned above. Thus, the total energy consumption E is represented as $E_K + E_{PE}$.



(a) Example of loop level paralleled scheduling for CGRA



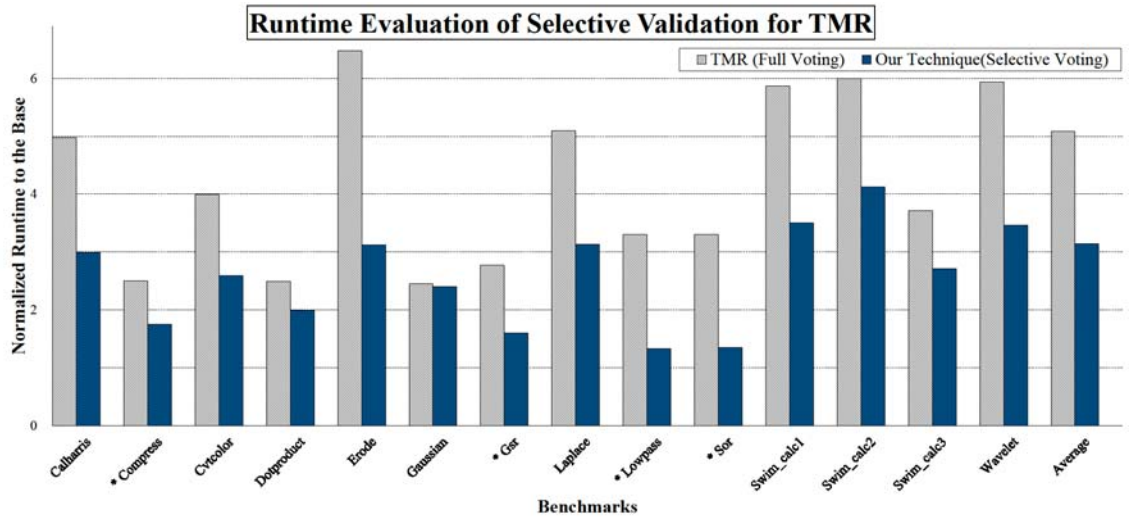
(b) Uniting the prologue and the epilogue parts

Fig. 18. Counting the numbers of nodes for the prologue, the kernel, and the epilogue.

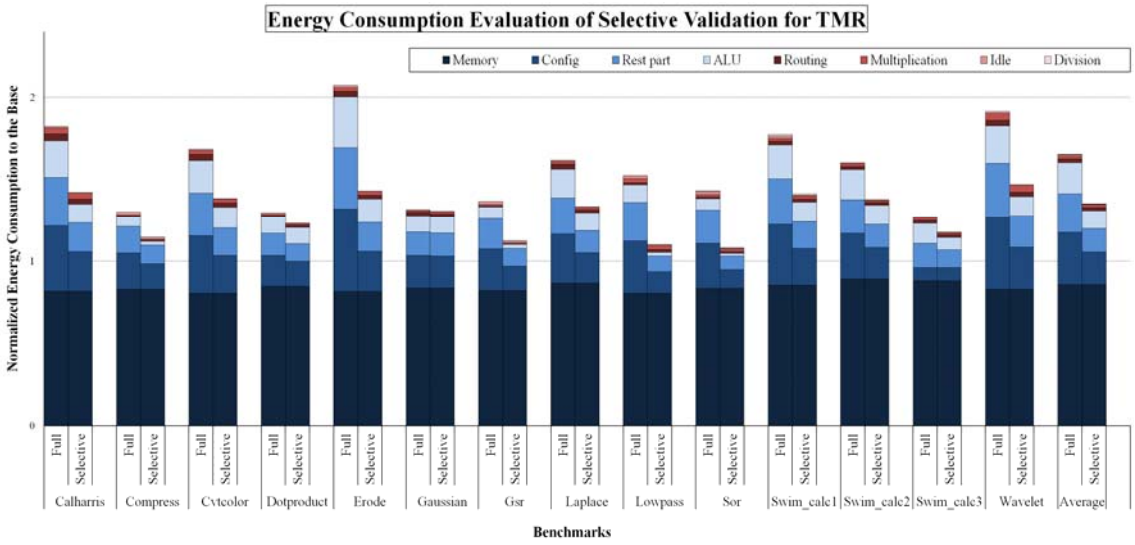
B. Experimental Results

1) Effectiveness of Selective Validations

Our first set of experiments is to evaluate the effectiveness of our selective validations for software-implemented redundancy techniques on CGRAs in terms of the runtime and the energy consumption. Figure 19 clearly show the effectiveness of our selective validation for TMR on CGRAs. Y-axis in Figure 19(a) represents the normalized runtime of TMR with the full voting and that of TMR with the selective voting (our approach) to that of the base. Over the suite of benchmarks, our selective validation for TMR can improve the performance in terms of the runtime by 38.3% on average as compared to that of the full validation for TMR on CGRAs. Y-axis in Figure 19(b) represents the normalized energy consumption of TMR with the full voting and that of TMR with the selective voting to that of the base. Over the suite of benchmarks, our selective validation for TMR can reduce the energy consumption by 18.1% on average as compared to that of the full validation for TMR on CGRAs. The main reason of these improvements of runtime and energy consumption is because our approach selects only synchronous operations, i.e., store operations, as validation points rather than every operation where the previously proposed TMR technique validates. Note that every voting requires additionally seven operations which are extremely expensive with respect to the runtime and the energy consumption. Figure 19(a) and Figure 19(b) show negligible improvements for benchmark Gaussian since it contains relatively small number of normal operations between memory ones. On the other hand, the other benchmarks contain the larger number of normal operations between memory ones where our approach can reduce the number of validations and improve the runtime and the energy consumption. Therefore, additional operations for triplication and voting can be covered by unused PEs for these benchmarks. In particular, Lowpass in TMR with the selective voting significantly improves the runtime (59.6%) than that in TMR with the full voting and Erode in TMR with the selective voting significantly improves the energy consumption (31.1%) than that in TMR with the full voting. Note that our approach triplicates every operation and can manage the comparable fault coverage as the previously proposed or conventional TMR technique does.



(a) Runtime evaluation of our selective voting
(the asterisk indicates benchmarks with recurrent loops)



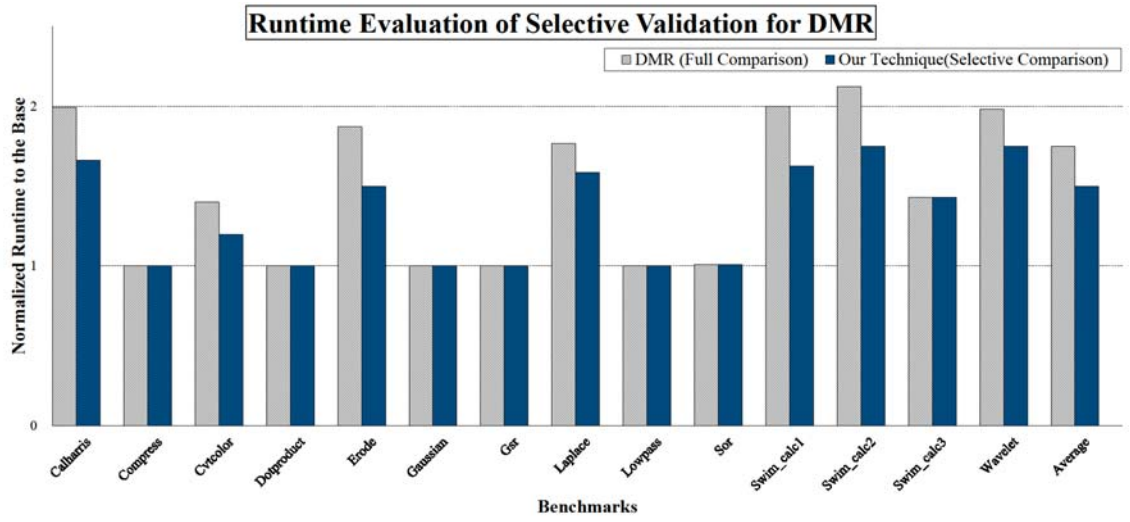
(b) Energy consumption evaluation of our selective voting

Fig. 19. Our selective voting for TMR outperforms the full voting in terms of runtime and energy consumption.

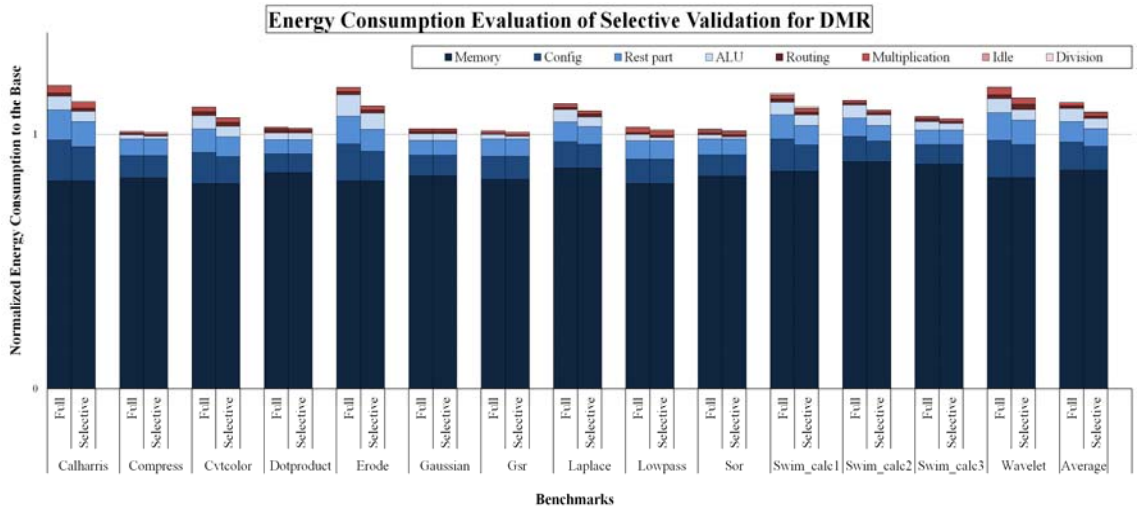
Interestingly, our selective voting techniques are more effective at reducing runtime for the benchmark kernels that include recurrent loops (benchmarks marked with the

asterisk in Figure 19(a)). They can improve the runtime by 47.8% on average in the selective voting as compared to the full voting while the other benchmarks can improve the runtime by 34.0% on average. In the case of applying TMR with the full voting mechanism to these benchmarks, the critical path of recurrent data dependence, crucially affecting the RecMII, lengthens about three times more than the critical path without voting mechanism. Due to the bigger RecMII, MII, the maximum value of RecMII and ResMII, is also set to the value of RecMII that is much higher value than ResMII, so the II increases; i.e., the performance degrades due to the longer data dependence between iterations. In our approach, however, the RecMIIs of these benchmarks slightly increase since the critical path lengthens less than the full voting. Thus, our approach can achieve better performance than the TMR with the full voting in recurrent loop cases.

We also evaluate our software-implemented DMR with the selective comparison and DMR with the full comparison in terms of the runtime and the energy consumption. Figure 20 clearly show that DMR with the selective comparison mechanism improves the runtime and the energy consumption. DMR with the full comparison duplicates and compares all the operations while our DMR with the selective comparison duplicates all the operations but compares only before a store operation is executed. We normalize the runtime of DMR with the full comparison and that of DMR with the selective comparison to that of the base as shown in Figure 20(a). Most benchmarks achieve runtime improvement (14.3% on average over benchmarks) with our selective comparison as compared to DMR with the full comparison. The benchmark Erode in DMR with the selective comparison achieves the maximum runtime improvement by 20.0%. And we normalize the energy consumption of DMR with the full comparison and that of DMR with the selective comparison to that of the base as shown in Figure 20(b). Most benchmarks achieve energy saving slightly (3.6% on average over benchmarks) as compared to DMR with the full comparison. The benchmark Erode in DMR with the selective comparison achieves the maximum energy consumption improvement by 6.4%. In general, DMR techniques with the selective validation achieve the less benefit in terms of the runtime and the energy consumption than TMR techniques mainly because DMR generates the smaller number of duplicated operations than triplicated operations in TMR. DMR also needs additional two operations for implementing the comparison while TMR needs additional seven operations for implementing the voting. Thus, several benchmarks incur the runtime and the energy consumption overheads in the selective comparison for DMR close to those in the full comparison.



(a) Runtime evaluation of our selective comparison



(b) Energy consumption evaluation of our selective comparison

Fig. 20. Our selective comparison for DMR outperforms the full comparison in terms of runtime and energy consumption

In Figure 19(b) and Figure 20(b), we also analyze the energy consumption portions of local memory access, configuration memory access, the rest part of PE array, idle PE, and active PE of ALU, multiplication, division, routing as summarized in Table 2.

There is no improvement of the energy consumption by the local memory access operations between the full validation and the selective validation as shown in Figure 19(b) and Figure 20(b). The reason is because there is no difference in the numbers of local memory access operations between the full validation and the selective one. The improvement of the selective validation as compared to the full validation is mainly influenced by operations of the configuration memory access, the rest part of PE array, and the active ALU PE. The energy savings by operations of the configuration memory access, the rest part of PE array, and the active ALU PE between the full validation and the selective validation are 9.8%, 7.3% and 6.6%, respectively, for TMR as shown in Figure 19(b) and 1.9%, 1.4% and 1.4%, respectively, for DMR as shown in Figure 20(b). The improvement of the energy consumption for the configuration memory access and the rest part of PE array comes from the decrease of II or runtime improvement. The DFG of the selective validation is usually simpler than the DFG of the full validation so II of selective schemes can improve due to the reduced number of nodes and edges. Therefore, the improvement of the energy consumption in the active ALU PE results from the reduced number of ALU nodes by validating selective schemes. Other operations such as the idle PE and the active PE of multiplication, division and routing slightly affect the improvement of the energy consumption between the full validation and the selective validation.

In summary, our selective validation techniques for TMR and DMR are significantly more effective in terms of runtime (by 38.3% and 14.3% on average) and energy consumption (by 18.1% and 3.6% on average) as compared to the complete validation techniques for those redundancy techniques implemented in software on CGRAs.

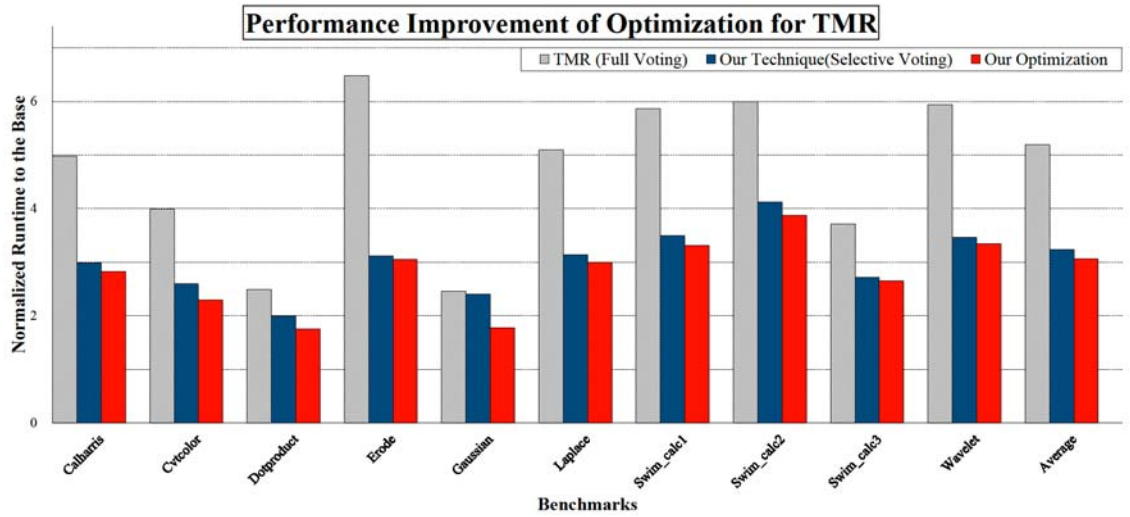
2) Enhanced Effectiveness with Optimizations

Our second set of experiments is to evaluate our optimization technique by reducing the number of store operations where the validation mechanism needs to be applied. Figure 21 and Figure 22 clearly show the effectiveness of our selective validation techniques with store operations reduced in terms of the runtime and the energy consumption. Note that benchmarks with the recurrent loops are excluded in this set of optimization experiments.

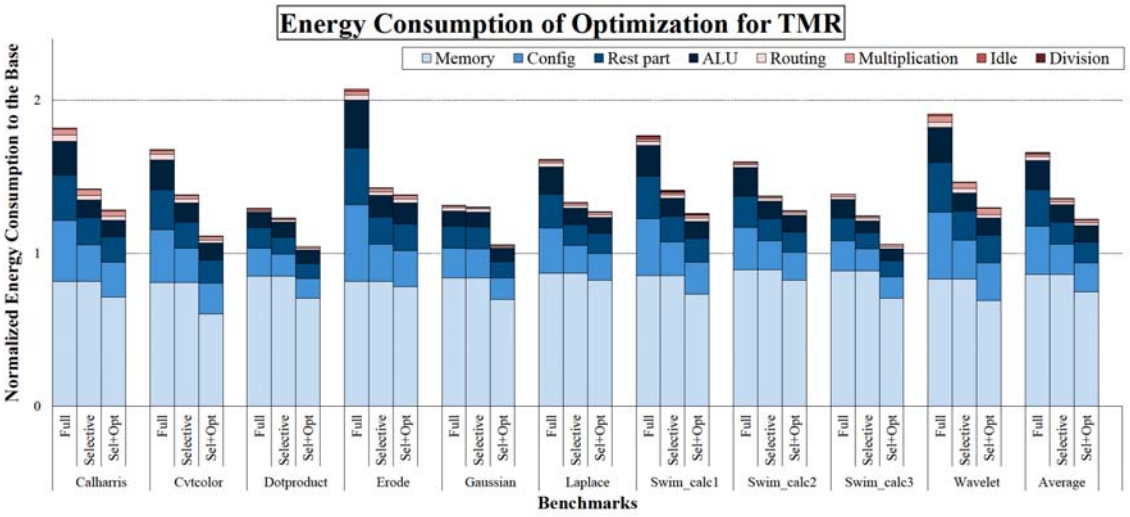
Figure 21(a) shows that our optimization technique can improve the performance with respect to the runtime on average by 10.5% as compared to our own technique without the optimization and by 41.0% as compared to the previously proposed TMR technique with the full validation. Interestingly, our selective validation techniques with the optimization are effective in terms of the runtime for benchmarks Dotproduct (59.9% improvement) and Gaussian (27.4% improvement) while they are less effective in the selective validation techniques without the optimization as shown in Figure 21(a). Figure 21(b) shows that our optimization technique can reduce the energy consumption by 10.1% on average as compared to our own technique without the optimization and by 26.2% as compared to the previously proposed TMR technique with the full validation. This improvement of the energy consumption for the optimization is definitely influenced by the local memory access operations as shown in Figure 21(b) and figure 22(b). The energy savings by the local memory access operations between the full validation and our optimization with store reduction are 6.8% for TMR and 9.9% for DMR. This effectiveness results from the reduced number of store operations obviously, and the power dissipation of the local memory access operation is relatively high as shown in Table 2. The energy savings by operations of the configuration memory access, the rest part of PE array, and the active ALU PE between the full validation and the our optimization with store reduction are 7.8%, 5.8% and 5.2%, respectively, for TMR and 1.7%, 1.3% and 0.8%, respectively, for DMR. The improvement of the energy consumption for other operations such as the configuration memory access, the rest part of PE array and so on comes from the same reason of the improvement by the selective validation as compared to the full validation.

Figure 22(a) shows that our optimization technique for DMR by reducing the number of store operations can achieve the runtime on average by 17.8% as compared to the previously proposed DMR techniques with the full comparisons. Note that these optimization techniques by reducing the number of store operations show the high effectiveness on some benchmarks where there exist several store operations so that we

can have enough margins to decrease the number of store operations, leading to the runtime improvement. On the contrary, benchmark Wavelet shows the runtime degradation since it has just two store operations where the store reduction rather incurs the runtime overhead due to the extra operations for merging these operations. Figure 22(b) shows that our optimization technique for DMR by reducing the number of store operations can achieve the energy consumption on average by 14.0% as compared to the previously proposed DMR techniques with the full comparisons. Interestingly, the energy consumption of our optimization technique for DMR is less than one of the base on average and some benchmarks such as Cvtcolor, Dotproduct, Gaussian, Swim calc1, and Swim calc3 as shown in Figure 22(b). It is reduced to 2.8% on average and 15.2% on benchmark Cvtcolor as compared to the base. In particular the benchmarks Dotproduct and Gaussian, both runtime and energy consumption of our optimization are less than of the base as shown in Figure 22(a) and Figure 22(b). It means that we can detect soft errors by applying the DMR technique under less runtime, less energy consumption, and tolerable reliability than original scheme on Dotproduct and Gaussian.

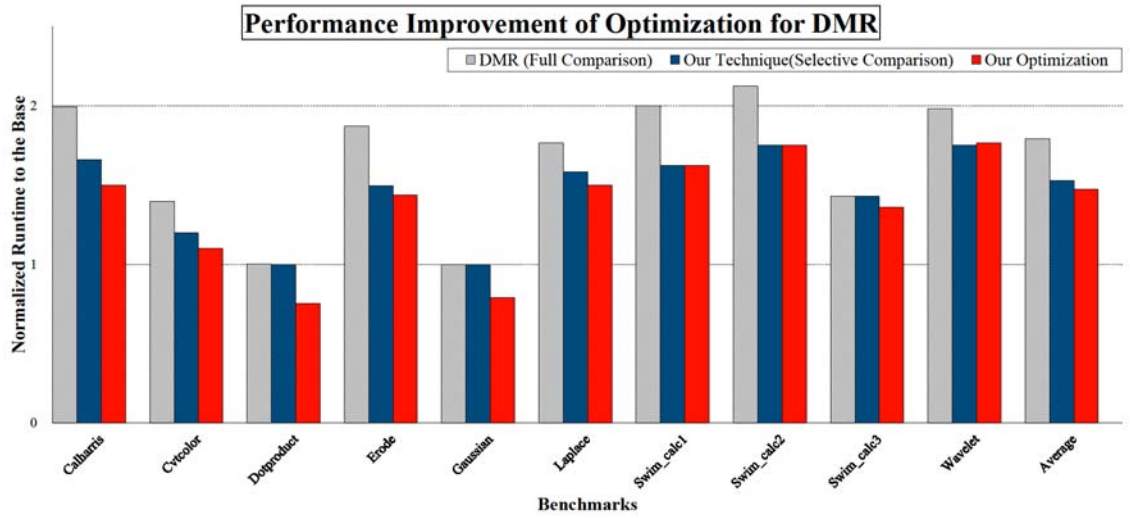


(a) Runtime improvement of our selective voting and optimization

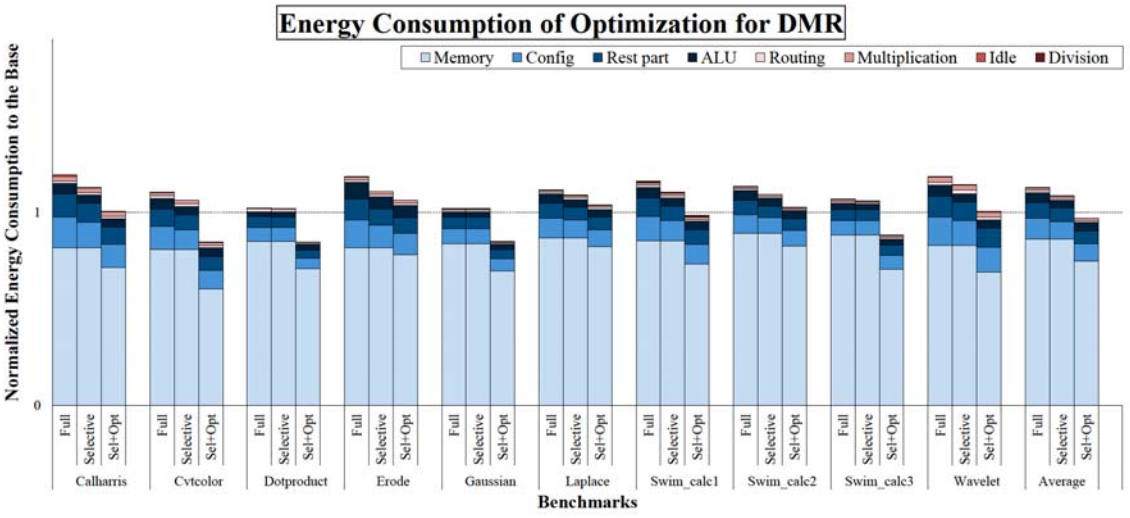


(b) Energy consumption of our selective voting and optimization

Fig. 21. Our optimization techniques for TMR can improve the performance in terms of runtime and the energy consumption



(a) Runtime improvement of our selective comparison and optimization



(b) Energy consumption of our selective comparison and optimization

Fig. 22. Our optimization techniques for DMR can improve the performance in terms of runtime and the energy consumption

In summary, our optimization technique with the selective validation techniques for TMR and DMR can achieve the further improvement in terms of the runtime (by 41.0% and 17.8% on average) and the energy consumption (by 26.2% and 14.0% on average) as compared to the previously proposed redundancy techniques with the complete validation implemented in software on CGRAs.

Our last set of experiments is to show evaluations of the runtime and the energy consumption of the base and all redundancy techniques such as DMR with the full comparison, DMR with the selective comparison, DMR with the selective comparison and the store reduction, TMR with the full voting, TMR with the selective voting, and TMR with the selective voting and the store reduction over the benchmark, Cvtcolor. Clearly, TMR techniques demand higher overheads for the runtime and the energy consumption than the DMR ones as shown in Figure 23 while TMR ones are able to correct errors and DMR ones are not (they just detect them, i.e., they need the recovery mechanisms). Our proposals with the selective validation and the store reduction can achieve the better performance in terms of runtime and energy consumption than conventional DMR and TMR techniques implemented in software on CGRAs. Our optimization technique by reducing the number of store operations can incur the runtime overheads by 10.0% for DMR and 130.0% for TMR and energy consumption overheads by -15.2% for DMR and 11.3% for TMR on average over benchmarks as compared to the base. Note that previously proposed DMR and TMR techniques incur the runtime overheads by 19.9% and 159.5% and the energy consumption overheads by 6.4% and 38.2%, respectively, which are much higher than our selective validation techniques.

Indeed, our selective techniques with the optimization can reduce the runtime overheads by 21.4% and 42.4% for DMR and TMR and also reduce the energy consumption overheads by 23.4% and 33.7% for DMR and TMR with the full validations, respectively. However, our selective validation techniques provide the comparable reliability to conventional redundancy techniques. Note that these experiments can be expanded to guide designers or programmers to explore interesting tradeoff spaces between the runtime, the energy consumption, and the reliability, which is definitely our future work.

In summary, our evaluations of runtime and energy consumption show the efficacy of our selective techniques with the optimization and our various approaches for redundancy techniques can open a new venue for multidimensional tradeoff studies.

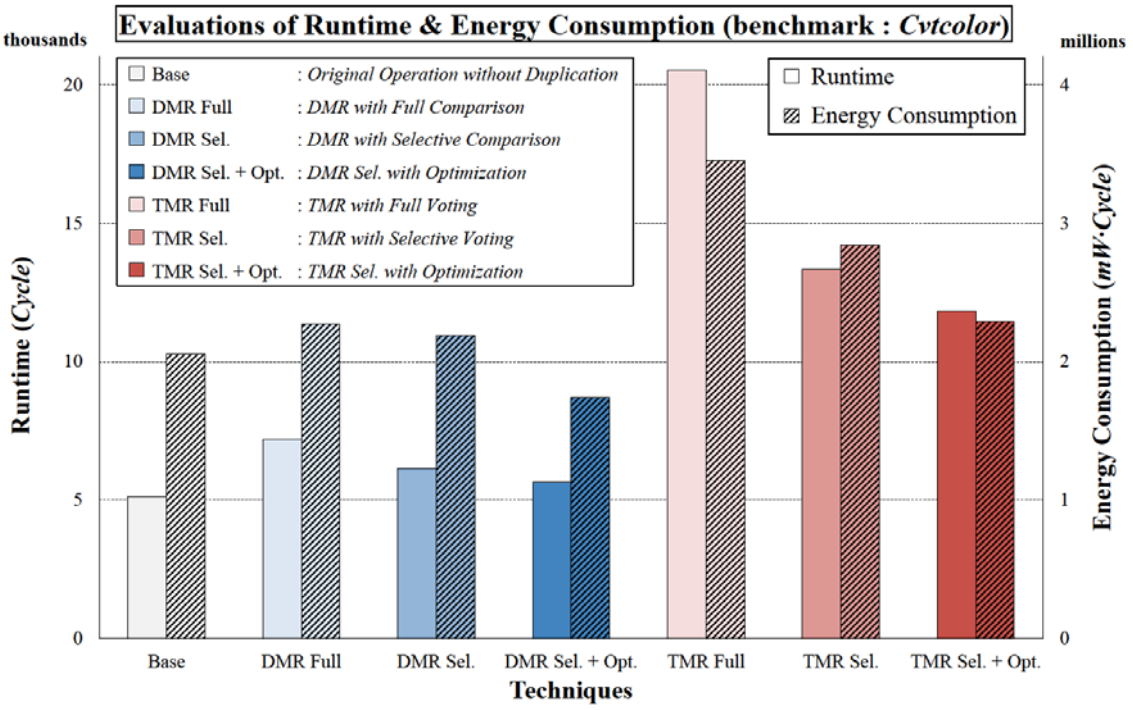


Fig. 23. Performance Evaluations among Various Protection Techniques (Benchmark: Cvtcolor)

Ⅵ. Conclusion

Soft errors induced by radiation are receiving significant concerns since the soft error rate is increasing exponentially with aggressive technology scaling. CGRA with high performance and high flexibility becomes more and more popular even in critical applications such as finance programs, human health system, etc. In order to improve the reliability in CGRA, several fault tolerant techniques have been proposed but they incur area cost and performance degradation significantly. In order to protect the datapath in CGRAs from soft errors without area cost, we propose software based selective validation techniques with the least performance overhead and the comparable fault coverage. We also propose an optimization technique by reducing the number of store operations to maximize the performance improvement. Our optimization technique merges multiple store operations into one store operation by DFG modification to reduce the number of validations. In conclusion, our selective validation techniques with the optimization can improve the runtime by 41.0% and the energy consumption by 26.2% as compared to the previously proposed TMR with the full validation.

Our future works include optimizing the operation of duplicating the original DFGs for applying redundancy technigues such as TMR and DMR with guaranteeing the comparable fault coverage and correct functionality in order to improve performance. We are also interested in investigating different priorities for various operations to apply the selective protection to only important or critical ones in terms of the reliability.

References

[1] Y. Kim, J. Lee, A. Shrivastava, and Y. Paek, “Operation and data mapping for CGRAs with multi-bank memory,” in ACM SIGPLAN Notices, 2010.

[2] H. Singh, G. Lu, R. Maestre, M. Lee, F. Kurdahi, N. Bagherzadeh, E. Filho, and R. Maestre, “MorphoSys: case study of a reconfigurable computing system targeting multimedia applications,” DAC 2000 Proceedings of the Design Automation Conference, 2000.

[3] R. Baumann, “Soft errors in advanced computer systems,” Design and Test of Computers, 2005.

[4] P. Hazucha and C. Svensson, “Impact of CMOS technology scaling on the atmospheric neutron soft error rate,” Nuclear Science, IEEE Transactions on, 2000.

[5] F. Wrobel, J. Palau, M. Calvet, O. Bersillon, and H. Duarte, “Simulation of nucleon-induced nuclear reactions in a simplified SRAM structure: scaling effects on SEU and MBU cross sections,” Nuclear Science, IEEE Transactions on, 2001.

[6] N. Wang, J. Quek, T. Rafacz, and S. Patel, “Characterizing the effects of transient faults on a high-performance processor pipeline,” DSN 2004 Proceedings of the Dependable Systems and Networks Page 61.

[7] D. Lyons, “SUN screen,” Forbes, 2000.

[8] S. Michalak, K. Harris, N. Hengartner, B. Takala, and S. Wender, “Predicting the number of fatal soft errors in Los Alamos National Laboratory’s ASC Q supercomputer,” Device and Materials Reliability, IEEE Transactions on, 2005.

[9] S. Jafri, S. Piestrak, O. Sentieys, and S. Pillement, “Design of a fault tolerant coarse-grained reconfigurable architecture: a case study,” ISQED 2010 Proceedings of the The International Symposium on Quality Electronic Design.

[10] D. Alnajiar, Y. Ko, T. Imagawa, H. Konoura, M. Hiromoto, Y. Mitsuyama, M. Hashimoto, H. Ochi, and T. Onoye, “Coarse-grained dynamically reconfigurable architecture with flexible reliability,” FPL 2009 Proceedings of the Field Programmable Logic and Applications.

[11] T. Schweizer, A. Kuster, S. Eisenhardt, T. Kuhn, and W. Rosenstiel, “Using run-time reconfiguration to implement fault-tolerant coarse grained reconfigurable architectures,” IPDPS 2012 Proceedings of the International Parallel & Distributed Processing Symposium.

[12] T. Schweizer, P. Schlicker, S. Eisenhardt, T. Kuhn, and W. Rosenstiel, “Low-cost TMR for fault-tolerance on coarse-grained reconfigurable architectures,” ReConFig 2011 Proceedings of the Reconfigurable Computing and FPGAs.

[13] C. Engelmann, H. Ong, and S. Scott, “The case for modular redundancy in large-scale high performance computing systems,” IASTED 2009 Proceedings of the

International Association of Science and Technology for Development.
[14] F. Kastensmidt, L. Sterpone, L. Carro, and M. Reorda, “On the optimal design of triple modular redundancy logic for SRAM-based FPGAs,” DATE 2005 Proceedings of the Design, Automation and Test.
[15] K. Singh, A. Agbaria, D. Kang, and M. French, “Tolerating SEU faults in the Raw architecture,” WDES 2006 Proceedings of the International Workshop on Dependable Embedded Systems.
[16] G. Lee and K. Choi, “Thermal-aware fault-tolerant system design with coarse-grained reconfigurable array architecture,” AHS 2010 Proceedings of the Adaptive Hardware and Systems.
[17] S. Pillement, O. Sentieys, and R. David, “DART: a functional-level reconfigurable architecture for high energy efficiency,” EURASIP Journal on Embedded Systems, 2008.
[18] S. Eisenhardt, A. Kuster, T. Schweizer, T. Kuhn, and W. Rosenstiel, “Spatial and temporal data path remapping for fault-tolerant coarsegrained reconfigurable architectures,” DFT 2011 Proceedings of the defect and fault tolerance in VLSI systems.
[19] Y. Kim and R. Mahapatra, “Dynamic context management for low power coarse-grained reconfigurable architecture,” GLVLSI 2009 Processdings of the Great Lakes VLSI.
[20] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J. Lee, W. Lee et al., “The Raw microprocessor: a computational fabric for software circuits and general-purpose programs,” Micro, IEEE, 2002.
[21] G. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. August, “SWIFT: Software implemented fault tolerance,” CGO 2005 Proceedings of the Code Generation and Optimization.
[22] B. Mei, S. Vernalde, D. Verkest, and R. Lauwereins, “Design methodology for a tightly coupled VLIW/Reconfigurable Matrix Architecture: A case study,” DATE 2004 Proceedings of the Design, Automation and Test.
[23] J. Chang, G. A. Reis, and D. I. August, “Automatic instruction-level software-only recovery,” DSN 2006 Proceedings of the Dependable Systems and Networks.
[24] B. R. Rau, “Iterative modulo scheduling: An algorithm for software pipelining loops,” in MICRO 1994.
[25] K. Lee, A. Shrivastava, M. Kim, N. Dutt, and N. Venkatasubramanian, “Mitigating the impact of hardware defects on multimedia applications: a cross-layer approach,” in ACM Multimedia, 2008.
[26] Y. Kim, J. Lee, A. Shrivastava, and Y. Paek, “Memory access optimization in compilation for coarse-grained reconfigurable architectures,” TODAES 2011 Proceedings of the The ACM Transactions on Design Automation of Electronic Systems.
[27] G. Bradski, “The OpenCV library,” Doctor Dobbs Journal, 2000.

[28] J. L. Henning, “SPEC CPU2000: Measuring CPU performance in the new millennium,” Computer, 2000.
[29] http://technology-and-science.lawyers.com/blogs/archives/4461-Are-Cosmic-Rays-a-Factor-in-Toyota-Acceleration-Problems.html
[30] KIM, Y., LEE, J., MAI, T. X., AND PAEK, Y. 2012. Improving performance of nested loops on reconfigurable array processors. ACM Trans. Archit. Code Optim. 8, 4, 32:1 – 32:23.
[31] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. Kim. Robust system design with built-in soft-error resilience. Computer 2005 Proceedings of the Computer.
[32] A. Nieuwland, S. Jasarevic, and G. Jerin. Combinational logic soft error analysis and protection. In On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International, pages 6.
[33] M. Rebaudengo, M. Sonza Reorda, M. Torchiano, and M. Violante. Soft-error detection through software fault-tolerance techniques. In Defect and Fault Tolerance in VLSI Systems, 1999. DFT'99. International Symposium on, pages 210-218. IEEE, 1999.
[34] K. Mohanram and N. Touba. Cost-effective approach for reducing soft error failure rate in logic circuits. In International Test Conference, pages 893-901, 2003.
[35] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. 2004 ACM SIGARCH Computer Architecture News.
[36] J. Blome, S. Gupta, S. Feng, and S. Mahlke. Cost-efficient soft error protection for embedded microprocessors. In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems, pages 421-431. ACM, 2006.
[37] H. Pao-Ann, L. Shang-Wei, H. Chin-Chieh, F. Jih-Ming, L. ChaoSheng, C. Cheng-Chi, C. Kuo-Cheng, L. Chun-Hsien, and L. Pin-Hsien, "Real-time embedded software design for mobile and ubiquitous systems," in International Conference on Embedded and Ubiquitous Computing, EUC 2007, (Lecture Notes in Computer Science vol. 4808), pp. 718-729.
[38] Shivakumar, P., Kistler, M., Keckler, S.W., Burger, D., Alvisi, L.: Modeling the e ffect of technology trends on the soft error rate of combinational logic. In: DSN ’02, Dependable Systems and Networks, Washington, DC, USA, 2002, pp. 389 – 398. IEEE Computer Society, Los Alamitos (2002).
[39] http://techno.okezone.com/read/2009/10/31/324/271023/sun-microsystem-tingkatkan-kinerja-portfolio-storage
[40] http://technology-and-science.lawyers.com/blogs/archives/4461-Are-Cosmic-Rays-a-Factor-in-Toyota-Acceleration-Problems.html
[41] http://www.veteranstoday.com/2012/07/29/fukushimas-melted-reactors-500-days-on/fukushima-daiichi-reactor-3-explosion-images/

[42] https://mocana.com/blog/2012/page/32/
[43] http://www.qualitydigest.com/magazine/2009/mar/article/when-your-life-depends-software.html
[44] http://www.woodward.com/applications-aircraft.aspx
[45] http://drivesteady.com/the-ladys-guide-to-car-insurance

감사의 글

언제나 제게 많은 힘과 용기를 준 아버지와 항상 우리 가족을 위해 기도하시는 어머니, 제 뒤에서 말없이 저를 도와주는 누나에게 먼저 고마움을 전합니다.

석사과정동안 많은 지도와 관심을 가져주신 지도교수 이경우 교수님께 깊이 감사드립니다. 또한, 바쁘신 와중에도 귀중한 시간을 내주시어 논문에 대한 충고를 해 주신 한요섭 교수님, 벅스텔러 교수님께도 감사드립니다. 그밖에도 지난 4년간의 학부시절과 2년간의 석사시절동안 많은 가르침을 주신 컴퓨터과학과 모든 교수님들에게도 감사의 마음을 전합니다.

연구실 동생중에 한명으로 많은 도움을 준 요한이과 학부생 휘수, 또한 석사 2년동안에 좋은 추억을 남겨준 준형이, 영빈이, 주성이 그리고 논문작업을 같이 진행한 서울대 소프트웨어 최적화연구실 이종원 박사과정, 김용주 박사에게도 감사의 마음을 전합니다. 연구실의 막내로써 열심히 한 준현이에게도 앞으로의 미래에 보다 좋은 일들만이 생기기를 바랍니다.

이밖에도 저를 아는 모든 분들께 이제까지의 제 삶에 영향을 준 것에 대해서 깊은 감사의 마음을 전하며 항상 건강하시고 웃음을 잃지 않으시길 바랍니다. 다시 한번 이제까지의 제 삶에 저의 정신적 지주였던 아버지와 어머니, 누나에게 머리 숙여 감사의 마음을 전하며 이 글을 마칩니다.

2013년 7월 12일

강 지 훈