DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

UCL

Fault Tolerant Integer Data Computations: Algorithms and Applications

Ijeoma Jane-Frances Anarado

Communication and Information Systems Research Group

Department of Electronic and Electrical Engineering

University College London

A thesis submitted for the degree of

Doctor of Philosophy

December, 2016

Statement of Originality

I declare that the work presented in this thesis and the thesis itself was composed and originated by myself in the Department of Electronic and Electrical Engineering, University College London. The work of other persons is appropriately acknowledged.

Ijeoma J.F. Anarado

Dedication

To the Almighty, who makes everything perfect in His time.

Acknowledgments

I would like to express my warmest gratitude to everyone who contributed to the success of my PhD program and the completion of this thesis.

To God, the Father Almighty, whose grace kept me going through the long, gloomy, as well as the sunny days of this journey. Many thanks to the members of the Newman House Catholic Chaplaincy and my local parishes of St. John Vianney West Green and Sacred Heart Parish Holloway, for providing an atmosphere of worship, communion, community service and support for international students like me.

To the Federal Government of Nigeria, for funding my PhD studies and to my colleagues at the University of Nigeria, Nsukka, for encouraging me to take this big step.

To my supervisor, Dr. Yiannis Andreopoulos, for being a great mentor and counsellor to me. I cannot thank you enough for nurturing, teaching and most especially, for believing in me so much. I have learned a lot from you through these years and I can only hope to be as supportive to my students as you have been to me. To my 'assistant supervisor' and colleague, Russell, thanks so much for your time, guidance and assistance. In addition, a note of thanks to the supportive academics and staff of the EE department at UCL, for constantly providing learning and networking opportunities for students.

A very big thank you to my family and friends who patiently and lovingly listened to my endless rants, while supporting and encouraging. Your love and prayers kept me going. Special gratitude to my greatest fan and husband, Chidoo, for cheering me up daily from over four thousand miles away these four years.

To all my colleagues in the CISG group and the PRESSID team, thank you for sharing all the "wonderful" PhD tales and reminding me that I was not alone in the valley.

Finally, to my examiners, Prof. Dimitrios Nikolopoulos and Prof. Andreas Demosthenous, as well as the anonymous reviewers of my papers, your constructive criticisms of my work and helpful suggestions have contributed immensely to this final product. Thank you very much.

Chukwu gozie unu niile!!!

Abstract

As computing units move to higher transistor integration densities and computing clusters become highly heterogenous, studies begin to predict that, rather than being exceptions, data corruptions in memory and processor failures are likely to become more prevalent. It has therefore become imperative to improve the reliability of systems in the face of increasing *soft error* probabilities in memory and computing logic units of silicon CMOS integrated chips.

This thesis introduces a new class of algorithms for fault tolerance in compute-intensive linear and sesquilinear ("one-and-half-linear") data computations on integer data inputs within high-performance computing systems. The key difference between the proposed algorithms and existing fault tolerance methods is the elimination of the traditional requirement for additional hardware resources for system reliability.

The first contribution of this thesis is in the detection of hardware-induced errors in integer matrix products. The proposed method of numerical packing for detecting a single error within a quadruple of matrix outputs is described in Chapter 2. The chapter includes analytic calculations of the proposed method's computational complexity and reliability. Experimental results show that the proposed algorithm incurs comparable execution time overhead to existing algorithms for the detection and correction of a limited number of errors within generic matrix multiplication (GEMM) outputs. On the other hand, numerical packing becomes substantially more efficient in the mitigation of multiple errors. The achieved execution time gain of numerical packing is further analyzed with respect to its energy saving equivalent, thus paving the way for a new class of silent data corruption (SDC) mitigation method for integer matrix products that are fast, energy efficient, and highly reliable.

A further advancement of the proposed numerical packing approach for the mitigation of core/processor failures in computing clusters (a.k.a., fail-stop failures) is described in Chapter 3 . The key advantage of this new packing approach is the ability to tolerate processor failures for all classes of sum-of-product computations. Because multimedia applications running on cloud computing platforms are now required to mitigate an increasing number of failures and outages at runtime, we analyze the efficiency of numerical packing within an image retrieval framework deployed over a cluster of AWS EC2 spot (i.e., low-cost albeit terminable) instances. Our results show that more than 70% reduction of cost can

be achieved in comparison to conventional failure-intolerant processing based on AWS EC2 on-demand (i.e., higher-cost albeit guaranteed) instances.

Finally, beyond numerical packing, we present a second approach for reliability in the case of linear and sesquilinear integer data computations by generalizing the recently-proposed concept of numerical entanglement. The proposed approach is capable of recovering from multiple fail-stop failures in a parallel/distributed computing environment. We present theoretical analysis of the computational and bit-width requirements of the proposed method in comparison to existing methods of checksum generation and processing. Our experiments with integer matrix products show that the proposed approach incurs $1.72\% - 37.23\%$ reduction in processing throughput in comparison to failure-intolerant processing while allowing for the mitigation of multiple fail-stop failures without the use of additional computing resources.

Publications:

1. Anarado, I., Anam, M. A., Anastasia, D., Verdicchio, F., & Andreopoulos, Y. “Highly-Reliable Integer Matrix Multiplication via Numerical Packing”. IEEE 19th International On-Line Testing Symposium (IOLTS), 2013 (pp. 19-24).

2. Anarado, I., & Andreopoulos, Y. “Mitigation of Fail-Stop Failures In Integer Matrix Products Via Numerical Packing”. IEEE 21st International On-Line Testing Symposium (IOLTS), 2015 (pp 101 - 107).

3. Anarado, I., & Andreopoulos, Y. “Core Failure Mitigation In Integer Sum-of-Product Computations On Cloud Computing Systems ”. IEEE Transactions on Multimedia , vol.18, no.4, pp.789–801, 2016.

4. Anarado, I., Anam, M. A., Verdicchio, F., & Andreopoulos, Y. “Mitigating Silent Data Corruptions In Integer Matrix Products: Toward Reliable Multimedia Computing On Unreliable Hardware ”. IEEE Transactions on Circuits and Systems for Video Technology, 2016. *To appear*

5. Anam, M. A., Anarado, I., & Andreopoulos, Y. “Generalized Numerical Entanglement For Fail-Stop Failure Mitigation In Linear, Sesquilinear And Bijective Operations On Integer Data Streams ”. IEEE Transactions on Emerging Topics in Computing, 2016. *To appear*

Contents

List of Figures

List of Tables

Abbreviations

ABFT	Algorithm Based Fault Tolerance
ATLAS	Automatically Tuned Linear Algebra Software
AWS	Amazon Web Services
BER	Backward Error Recovery
CMOS	Complimentary Metal-Oxide Semiconductor
DARPA	Defense Advance Research Projects Agency
dGEMM	Double-precision Generic Matrix Multiplication
DMR	Dual Modular Redundancy
DVFS	Dynamic voltage and frequency scaling
EC2	Elastic Compute Cloud
ECC	Error Correcting Code
FER	Forward Error Recovery
GEMM	Generic Matrix Multiplication
gotoBLAS	Basic Linear Algebra Subprograms (designed by Kazushige Goto)
HPC	High Performance Computing
IUD	Independent Uniformly Distributed
KULFI	Kontrollable Utah Low-Level Virtual Machine Fault Injector
LLVM	Low-Level Virtual Machine
MKL	Math Kernel Library

MTTF	Mean Time to Failure
PU	Processing unit
SDC	Silent Data Corruption
SET	Single Event Transient
SEU	Single Event Upset
sGEMM	Single-precision Generic Matrix Multiplication
SIMD	Single Input Multiple Data

Notational Conventions

Symbol	Definition
$\mathbf{A}$,$\mathbf{a}$,$A[m,n]$, $a_{m,n}$	boldface uppercase and lowercase letters indicate matrices and vectors, respectively; the corresponding italicized letters indicate individual elements.
$\mathbf{A}_{\text{top}}$	the submatrix of $\mathbf{A} \in \mathbb{R}^{L \times L}$ comprising all rows and **only** columns with index range between $\left\{0, \ldots, \left\lfloor \frac{L}{2} \right\rfloor - 1\right\}$
$\mathbf{A}_{\text{bot}}$	the submatrix of $\mathbf{A} \in \mathbb{R}^{L \times L}$ comprising all rows and **only** columns with index range between $\left\{\left\lfloor \frac{L}{2} \right\rfloor, \ldots, L-1\right\}$
$\lceil a \rfloor$	rounds a to the nearest integer
$\lfloor a \rfloor$	largest integer that is smaller or equal to a (floor operation)
$a \gg b$	shifts a by b bits to the right discarding the least-significant bits
$a \ll b$	shifts a by b bits to the left discarding the most-significant bits
$\mathcal{S}_b\{a\} = \begin{cases} a \ll b, & b \geq 0 \\ a \gg (-b), & b < 0 \end{cases}$	generalization for left and right bit shifting
$a \leftarrow b$	assigns value b to a
$\widehat{a}$	packed/entangled element (equivalently for vectors and matrices)
$\widetilde{a}$	recovered element after unpacking/entanglement (equivalently for vectors and matrices)
$\Pr\{\text{event}\}$	indicates the probability of occurrence of event "event"
$\text{Re}\{\text{method}\}$	indicates reliability, i.e., the percentage of errors that method "method" can detect under a predetermined model for error occurrences.

Chapter 1

Introduction and Literature Review

The exponential growth in the sophistication and processing power of computer chips, brought about by miniaturization and increase in CMOS integration density, has been the driving force in technological advancements in the past decades. In addition, today, computing clusters and virtualization provide for significant parallelism possibilities in comparison to conventional desktop multicore systems. These systems, together with the corresponding parallel algorithms that run on them, find numerous applications in multimedia processing systems, especially with the increase in the availability of large datasets and the requirement for real-time data processing. However, despite the throughput acceleration offered by multi-core and multi-processor systems, modern computing clusters are now beginning to exhibit unreliable operation even under nominal conditions [10, 11]. This is as a result of the radiation and ther-

mal sensitivity exhibited by CMOS electronics stemming from technology downscaling, aging and other manufacturing non-idealities [12, 13] that lead to the so-called soft errors or silent data corruption (SDC) in memory and logic. An example of the sensitivity of silicon electronics to circuit noise as a result of downscaling is illustrated in Fig. 1.1. The figure shows a basic CMOS inverter circuit, together with its DC characteristics at different supply voltage levels. It is straightforward to observe the increased susceptibility of this circuit to voltage deviations/current pulses (circuit noise) as the supply voltage is scaled to near-threshold levels [14]. More explicitly, Fig. 1.1(c) illustrates a circuit level voltage deviation masked by the inverter when operated at $V_{DD} = 3V$ and $V_{DD} = 5V$. However, when operated at $V_{DD} = 1V$ and the amplitude of the deviation is greater than V_{th1}, the logic state of the inverter output is reversed. The propagation and manifestation of such flips within data input/output or control logic can lead to degraded outputs or even complete system malfunction (e.g. infinite loops).

Thus, future CMOS technologies now require increased levels of resilience to transient arithmetic, memory or logic faults caused by process variations and other soft errors (e.g., caused by particle strikes, circuit overclocking or voltage scaling) [15, 16] .

Beyond externally-induced errors, soft errors may arise due to self-induced changes in a computing system. For instance, one of the major bottlenecks in the increase of CMOS integration density is increased power dissipation [14]. In a bid to reduce the increases in power incurred by the increase in number of transistors per chip, voltage scaling is becoming

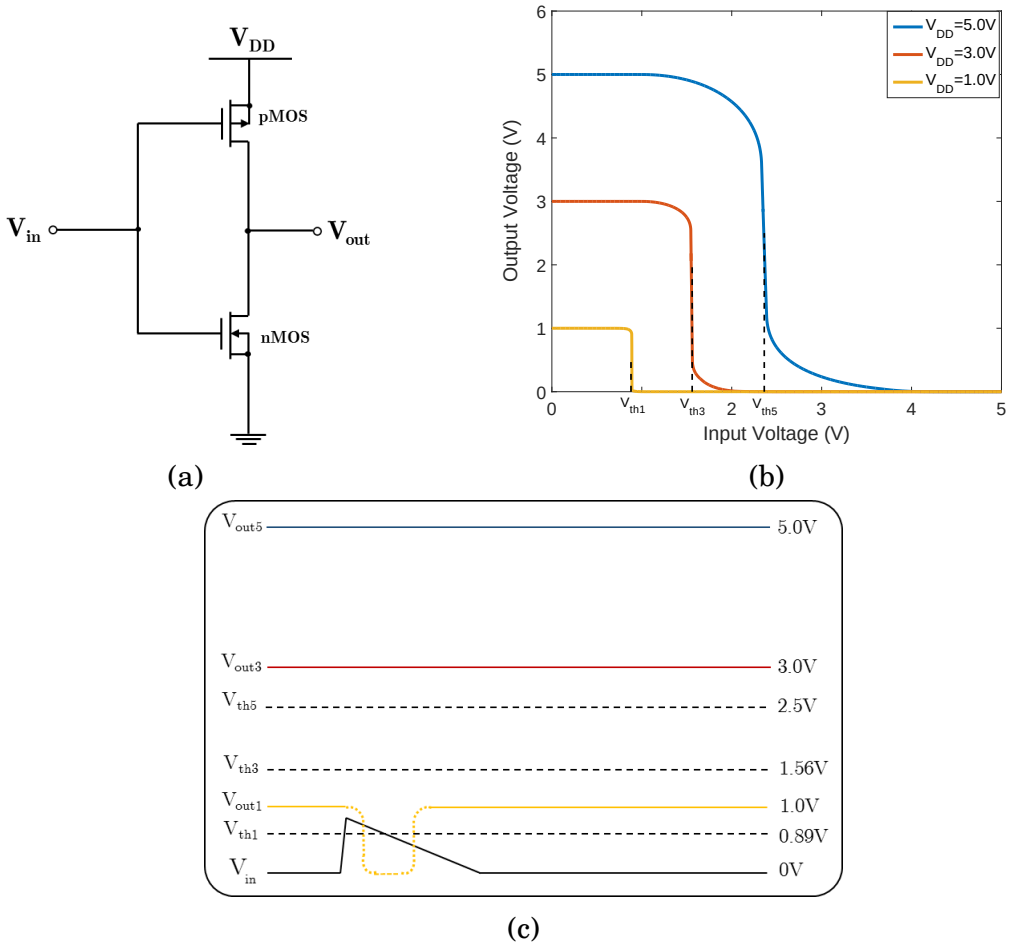


Figure 1.1: (a) Schematic of CMOS inverter (b) Voltage transfer characteristics of a CMOS inverter for different supply voltages [1];(c) Supply voltage deviation and equivalent output voltage response due to circuit noise.

increasingly used by processor designers [14, 17]. However, given that voltage scaling increases the probability of occurrence of soft errors [17], its detriment is that it raises the need for strong error detection and correction mechanisms, especially for soft errors occurring on data cache memories [17, 18] that remain undetected because they do not lead to a system halt or crash.

Furthermore, the proliferation of high-end distributed computing platforms with thousands of processing nodes imply that the overall mean time to failure (MTTF) estimates of such systems continue to decrease, with the highest root causes of failures being hardware and software abnormalities [19]. Schroeder et. al. [20] show that the failure rate of a system is indeed proportional to the number of processors that constitute the system. Presently, a plethora of high performance computing (HPC) failure logs report average typical MTTF of popular HPC systems to be in the order of days or even hours [19–21]. Therefore, to ensure long running scientific and data processing algorithms, fault tolerance is unarguably indispensable both now and for the future especially as we move to the exascale computing era with MTTF estimates in the order of tens of minutes [22]. Similarly, cloud computing clusters today provide for significant parallelism possibilities at the cost of decreased MTTF characteristics in comparison to conventional desktop multicore systems [23, 24]. For example, high-performance clusters can now be deployed using Amazon Web Services Elastic Compute Cloud (AWS EC2) spot instances with substantially-reduced billing cost [25]. However, AWS reserves the right to terminate EC2 spot instances at any moment with little or no prior

notice. In addition, service interruptions may occur at unpredictable intervals, since processor cores in spot instance reservations may not be solely dedicated to the cluster under consideration. More broadly, other types of disruptions, such as vibration, power, or network-induced performance degradations, are also frequently reported in large computing clusters [24, 26]. Such interruptions tend to result in substantial reduction in processing throughput and are therefore extremely detrimental in the performance of high-volume, low-latency, multimedia applications on cloud computing clusters [27, 28].

System reliability has thus become a challenge of increasing importance for prevalent applications in mobile, desktop and high-performance systems, such as: webpage or multimedia retrieval [29, 30], relevance ranking (e.g., identifying webpage significance via Google's PageRank algorithm [31]), object or face recognition in video for machine learning and security applications [32, 33], etc. The algorithmic part within all these systems is based on [34]: power iterations, backpropagation training, transform decompositions, covariance matrix calculations, block Lanczos iterations, etc, and the input data comprise real numbers (image/audio samples, document/webpage features, etc). Within all these algorithms, the compute- and memory-intensive parts comprise linear and sesquilinear operations on real number inputs using integer or floating number representations. Specifically, it is well known that large *sum-of-product computations*, i.e., inner and outer products, generic matrix multiplication (GEMM) [11, 35–40], and multidimensional convolution/cross-correlation (CONV) operations [41, 42] are the building block of most

multimedia and digital signal processing routines. These operations are typically performed using vectorized integer sum-of-product routines, or optimized single/double-precision floating-point libraries [e.g., `sGEMM` and `dGEMM` routines of a mathematics kernel library (MKL)] [35,36,43]. Therefore, ensuring the robustness of these operations to processing unit failures and SDCs at critical processing stages is of paramount importance for large-scale multimedia application deployment in cloud computing clusters exhibiting low MTTF characteristics.

In this thesis, we present a set of algorithms for the mitigation of fail-stop (core/processor failures) and fail-continue (SDC) failures devoid of the traditional requirement for additional computing resources for improving overall system reliability. In the proposed numerical-packing based and numerical-entanglement based algorithms for fault tolerance, bits within the numerical representation of data inputs are harnessed for system reliability purposes. Thus, the proposed algorithms trade off output dynamic range of application data for increased levels of resilience. For example, it will be shown in this thesis that the structure of the generic matrix multiply routine (GEMM), which forms the core compute-intensive unit of several signal processing algorithms allows for a methodical interleaving of several data elements within a single number representation in such a way as to ensure precise extraction of the constituent units after processing. Specifically, through packing pairs of input matrix rows and kernel matrix columns using low-overhead integer bit-shift operations, we are able to pinpoint the local set (usually four matrix outputs) of erroneous matrix outputs, thereby offering low-cost

error correction capability via re-computation of the flagged output elements. We show that the complexity of pre- and post-processing of matrix data for our fault tolerance method is up to two orders of magnitude less that the ensuing GEMM computation when no SDCs are detected, while performing substantially better than existing checksum methods for multiple error cases.

An alternative packing method capable of recovering data under fail-stop failures (or detecting SDCs in the absence of failures) is described in Chapter 3. The proposed method systematically packs/encodes both input and kernel matrix data elements before parallel processing within an $L-$core/processor/node computing cluster, such that a limited number of processing units (PUs[1]), N ($N < L$), is sufficient for producing all L processed results at the output. Signal processing algorithms that support efficient packing and reliable recovery via the proposed method include all sum-of-product computations such as GEMM, multi-dimensional convolution/ cross-correlation (CONV) and Kronecker products. Experiments within shared and distributed memory computing clusters highlight the efficiency of the proposed algorithm when taking into account the cost of data communication accrued for data transfer required for pre- and post-processing of the proposed algorithm.

Finally, in order to simultaneously tolerate a higher number of fail-stop failures within the class of linear and sesquilinear integer processing rou-

[1]In this work, a processing unit (PU) refers to an individual unit of a parallel/distributed computing platform, performing a fraction of the workload and with independent failure characteristics. Therefore, a thread in a single core CPU, or a node with multiple processors within a HPC system is referred to as a PU, provided the system can continue operation even in the event of a PU failure.

tines, we introduce an asymmetric packing method that unlike the previous proposals, allows for the overlap of data elements within the enclosing representation. Our method extends the previous proposal from Anam and Andreopoulos [44] and, because it allows for the superposition of multiple elements, it has been termed as generalized numerical entanglement. By simplifying the exposition and derivations to the case of data stream entanglement, we show for the first time, the possibility of recovering the set of L data streams when up to F streams are lost ($L \geq 2F + 1$) without the use of checksum data groups.

The derivations of this work thus pave the way for a new class of algorithms that provide an application level power-aware fault tolerance mechanism. These derivations are in line with the recommendations from researchers including DARPA [18], for a robust cross-layer fault tolerance model, capable of alleviating the rising energy costs that are synonymous with hardware redundancy models. [18, 45, 46].

1.1 Literature Review

We start our review of literature in the field of fault tolerance by discussing the evolution of soft errors in semiconductor devices as well as its propagation across components of the system stack in order to produce visible erroneous or catastrophic results. In section 1.1.2, the fail-stop failure model is described with emphasis on likely failure causation, while also highlighting the limitations of existing mitigation methods. Since the proposals of this work target fault tolerance on the application

layer of system hierarchy, we discuss existing algorithms in this field, which form the basis of comparative analysis carried out in the later part of this work. We conclude this chapter by discussing the importance of SDC detection and correction within error tolerant applications.

1.1.1 Soft Errors and Silent Data Corruption

Soft errors are random, non-destructive short lived disturbances in semiconductor devices, usually caused by electrical noise, fluctuations in signal voltage, inductive coupling effects, particle strikes, clock skew effects, CMOS process variations and other external factors beyond the system designer's control [47]. For example, when charged particles (typically stemming from radioactive materials in chip packaging), cosmic rays and other high energy particles interact with semiconductor memory elements [48, 49], single or multiple bits of the stored value at the memory location are flipped. When these flips go undetected, subsequent computations with these erroneous data elements are likely to propagate silently to the application layer thereby producing erroneous outputs or degrading system performance (cf. Fig.1 of [50]) and in severe cases, failures. Thus, although a soft error may not be damaging to the device in itself, its proliferation as silent data corruption (a.k.a, fail continue failures) poses great reliability challenges for end users. Previously, it was assumed that soft errors were specifically problematic only for space and aviation-based electronic devices since they are continuously exposed to high energy particles (e.g. neutrons from cosmic rays) that generate electron-hole pairs when they come in contact with semiconductor de-

vices [51–53]. These electron-hole pairs result in the appearance of large number of charges such that when a given critical threshold is exceeded, the operating circuit responds by erroneously flipping the state of a logic or memory at the location of the high energy particle impact [48, 51, 54]. However, the impact of soft errors has been amplified with the emergence of deep sub-micron semiconductor electronics that increase the sensitivity of terrestrial-level devices to radiation and thermal noise by reducing the critical charge required to upset a logic/memory state [51, 53]. More specifically, by dynamically reducing the operating voltage of transistors as obtained in the dynamic voltage and frequency scaling (DVFS) technology, the critical charge threshold of transistors is further reduced thereby increasing the susceptibility of processors to the effect of soft errors [55, 56]. These erroneous responses of a system to non-idealities that do not necessarily cause a system failure but may propagate unnoticed through applications depending on time and location of the flip are known as soft-errors, transient faults, single event transients (SET) or single event upsets (SEU). Specifically, SEUs refer to soft errors arising from glitches in sequential logic cells (e.g memory cells, flip-flops) while SETs are used to refer to abnormalities arising from combinational logic circuits that are propagated to sequential logic elements [57, 58].

Existing soft error mitigation techniques for both sequential and combinational logic units at the lower level of the system abstraction layer include: *(i)* circuit-level methods such as hardware replication based radiation hardening techniques [59–61], voltage and current monitoring techniques [62–64] and selective concurrent error detection designs [58, 65];

(ii) architecture level coding techniques such as the Reed-Solomon code [66], Hamming code [67], variants of error correcting codes (ECC) [17, 68, 69] and arithmetic codes for combinational logic units [70–72]. Though an efficient implementation of some of these techniques for protection of memory cells have been shown to reduce SRAM soft error rates (cf. Figure 7 of [54] and Figure 9 of [12]), increasing combinational logic SER along with the complex corruption patterns (e.g., burst errors and multi-bit upsets) found in aggressively-scaled systems require increasingly- sophisticated fault tolerance methods. These methods are known to incur substantial area, performance and power overheads [12, 54, 73–75]. This is why the Exascale Study Group [18] put forward that mitigation of SDC bursts should also be based on fault-tolerant software/algorithmic designs rather than solely depending on expensive circuit-level techniques [18]. This has led to a growing number of research proposals for developing cross-layer system reliability designs [46, 76–78], which ensure that systems remain reliable under unreliable hardware.

1.1.2 Hard Error Models: The Case of Fail-Stop Systems

Beyond non-persistent/transient soft errors, system errors can be further classified based on the duration of their occurrence into: *(i)*: intermittent errors, which are characterized by the re-occurrence of a fault at a certain frequency and *(ii)*: hard errors that tend to be permanent and can only be mitigated by a reconfiguration (sometimes involving removal and repair)

of the affected component of the system [70, 79]. Hard errors can arise from manufacturing irregularities, circuit aging, power failure, software failure or human error [80]. In this work, we focus on a class of hard error causing failures known as fail-stop failures as opposed to byzantine failures (i.e., permanent failures that remain undetected). That is, fail-stop failures comprise a class of system failures that assumes that the overall system is aware of the failure/removal of a subsystem [81, 82]. Therefore, fail-stop failures within processing units are analogous to erasure channels in communication networks [83]. Within HPC environments, existing parallel computing frameworks like fault-tolerant Message Passing Interface (FT-MPI) [84] and Parallel Virtual Machine (PVM) [85], depend on a scheduled "alive" response from participating nodes within the network, without which the node is flagged as failed. In order to recover lost data on the failed node(s), failure mitigation techniques are employed to either recompute the lost data, or recover the data using error recovery algorithms similar to forward error correction codes in communication networks [83].

Recovery from fail-stop failures in parallel compute-intensive routines is currently achieved via *roll-back* or *roll-forward* methods. Roll-back methods are based on periodic checkpointing and recomputation if failures are detected. The vast majority of roll-back methods comprise backward error recovery (BER), where system states are stored periodically and computations are restarted from the last stored state when a failure happens in the computing environment [23, 82, 86–88] . Several advances in this area including Plank *et. al.*'s diskless checkpointing [89], Oliner *et. al.*'s

cooperative checkpointing [90] and multilevel checkpointing by Moody *et. al.* [91] aim at improving the efficiency of checkpointing intervals and reducing the significant communication overhead that characterize checkpoint and restart systems. However, recent BER studies show that, depending on the desired level of resilience to core failures, substantial resources may be spent on checkpointing, system state storage/recovery, and recomputation. This has been identified as a major challenge for future exascale systems [23,92], which calls for alternative methods of fault tolerance with reduced storage and communication requirements [18]. Above all, the need for increased processing throughput requires failure mitigation techniques devoid of expensive re-computation of failed processes [86]. Forward error recovery (FER) techniques are therefore an alternative to BER that ensure continuous processing even in the face of failures.

From a system perspective, module replication is perhaps the de facto FER method for highly reliable systems as employed for example in mission critical systems [93] and HPC systems with very long continuous operational lifetime [94]. However, the two- or three-fold increase in resources required to maintain the minimum form of redundancy is often unacceptable in practical deployments [95]. Specifically, the power requirements in order to achieve reliability by replication have been a problem of growing concern. This is why there is an increasing number of calls for more energy aware fault tolerant systems that are algorithm specific [18] and also for increased research in the area of cross-layer approaches for reliability [46].

Roll forward methods, similar to error correction codes for robust data communication, have been employed for error detection, correction and failure recovery in data processing tasks [96–99]. The key consideration in the design of FER codes for data processing applications is the requirement to maintain the encoding relationship throughout the data processing cycle [100]. However, because different arithmetic algorithms modify data differently, the major limitation of existing encoding-based fault tolerance techniques is their algorithm-specificity. Nevertheless, the low-cost computational overhead offered by algorithm-based FER methods make them suitable especially for real-time and power-aware systems. In addition, because low-level fault tolerance techniques may not be sufficient for SDC detection and correction for increasing error rates [18], these application level techniques can seamlessly be deployed on top of existing robust circuit-level designs for increased system resilience [18,46]. We describe existing algorithm-based methods for fault tolerance in the section that follows.

1.1.3 Algorithm Based Methods for Fault Tolerance in Data Computations

Pioneering works in this area emerged in the early 80's by Huang and Abraham[2] [7, 101, 102], who coined the term Algorithm Based Fault Tolerance (ABFT) that would then become the building block for advanced

[2]In this work, Algorithm Based Fault Tolerance (ABFT) refers to Huang and Abraham's original proposal [101], while mABFT is used to refer to various modifications and extensions, i.e., including the use of weighted checksums and subblock partitioning.

research in the field. ABFT methods are characterized by the use of "checksum" data elements specifically tailored to the algorithm under consideration for the reliable detection (and possibly correction) of up to a limited number of SDCs [4,6,96,97,101,103,104]. Huang and Abraham's proposal for improving the reliability of matrix product, addition, scalar multiplication, transpose and LU decomposition generates checksum elements via a linear combination of input matrix elements in such a way that the preserved checksum relationship after computation can be used for error detection and correction. For example, in order to reliably compute an $M \times K$-by-$K \times N$ matrix product $\mathbf{R} = \mathbf{A} \times \mathbf{B}$, a column checksum $\mathbf{a}_{\mathrm{c}}$ and a row checksum $\mathbf{b}_{\mathrm{r}}$ are generated by:

$$\begin{aligned} \mathbf{a}_{\mathrm{c}} &= \mathbf{w}_{\mathrm{cs}}^{\mathrm{T}}\mathbf{A} \\ \mathbf{b}_{\mathrm{r}} &= \mathbf{B}\mathbf{w}_{\mathrm{rs}} \end{aligned} \tag{1.1}$$

where $\mathbf{w}_{cs} \in \mathbb{R}^M$ and $\mathbf{w}_{rs} \in \mathbb{R}^N$ are unit column weight vectors. By appending the checksum vectors to their corresponding input matrices and performing the GEMM computation as illustrated in Figure 1.2, [101] show that the checksum relationship is preserved after computation and can thus be used for error detection and correction. Specifically: *(i)* the sum of each row and each column of $\mathbf{R}$ (which is contained within $\mathbf{R}_{\mathrm{f}}$) is validated against the corresponding checksum elements of $\mathbf{r}_{\mathrm{r}}$ and $\mathbf{r}_{\mathrm{c}}$; *(ii)* the sum of the checksum vectors themselves is validated against r_{rc}. If these validations fail for a certain row and column, the index of this row and column indicates the location of the SDC within $\mathbf{R}_{\mathrm{f}}$.

An obvious limitation of the traditional ABFT proposal is the inability

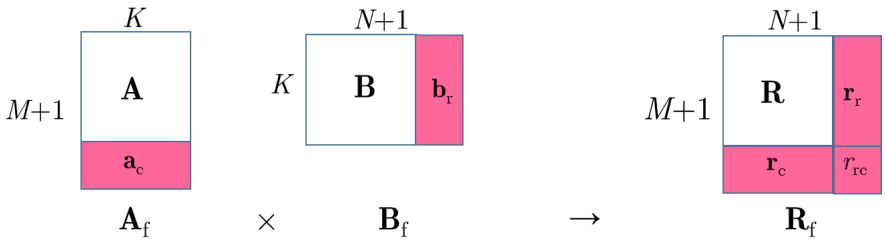


Figure 1.2: ABFT within a single subblock of GEMM via checksum vectors

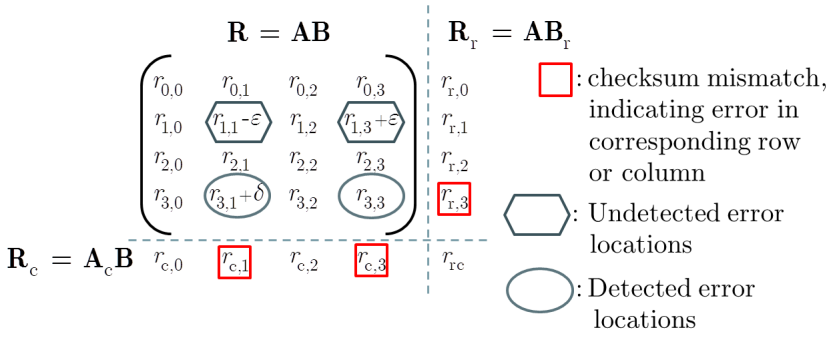


Figure 1.3: Errors in ABFT that require row and column recomputation (rollback ABFT [2]).

to pinpoint the exact locations of SDCs in the event of multiple SDC occurrence. This is often mitigated by row/column recomputation, which is termed "rollback ABFT" [2]. For example, given the $\pm\varepsilon$ and δ error pattern shown in Figure 1.3, ABFT flags locations $r_{3,1}$ and $r_{3,3}$ as erroneous, while two other erroneous locations, $r_{1,1}$ and $r_{1,3}$ go undetected due to the cancellation effect of the given error pattern. Therefore, when multiple SDCs are detected, the only reliable method for recovery via ABFT is to recompute (i.e., "rollback") the entirety of the rows and columns that have been detected as erroneous. If a substantial number of SDCs is detected, this will result in entire GEMM subblock recomputation, i.e., complete execution rollback, incurring significant execution time penalty.

More recently, modified ABFT methods have been proposed that make use of *additional* checksums, constructed with specific weight vectors with improved reliability at the cost of output dynamic range of matrix outputs and computation complexity. Jou, Anfison *et. al.* [7,105] extended ABFT with the introduction of d checksum rows and columns, $d > 1$. All checksum elements beyond the first one are generated via inner products of rows and columns with specially-constructed weight vectors. It is then shown that, under suitable choice of weight vectors, this extension can guarantee the detection of up to d SDCs and correction of up to $\lfloor \frac{d}{2} \rfloor$ SDCs [6, 7, 105] per row and column of a subblock product. This comes at the cost of increasing the size of the checksum matrices by d times. In more detail, the first checksum of each column of $\mathbf{A}$ and row of $\mathbf{B}$ is the one of the original ABFT; we denote this as the *unweighted checksum*, $\mathbf{a}_{c1}$ and $\mathbf{b}_{r1}$, respectively. For an mABFT approach with $d = 2$, the second checksum of each column of $\mathbf{A}$ and row of $\mathbf{B}$ (denoted by $\mathbf{a}_{c2}$ and $\mathbf{b}_{r2}$, respectively) is *the inner product of the column* of $\mathbf{A}$ (resp. *inner product of the row* of $\mathbf{B}$). Jou and Abraham [7] proposed the column and row weight vectors $\mathbf{w}_{\text{cexp}}$ and $\mathbf{w}_{\text{rexp}}$ given by:

$$\begin{aligned} \mathbf{w}_{\text{cexp}}^{\text{T}} &= \begin{bmatrix} 2^0 & 2^1 & \dots & 2^{M-1} \end{bmatrix} \\ \mathbf{w}_{\text{rexp}}^{\text{T}} &= \begin{bmatrix} 2^0 & 2^1 & \dots & 2^{N-1} \end{bmatrix} \end{aligned} . \tag{1.2}$$

However, due to the exponential increase of the dynamic range of $\mathbf{a}_{c2}$ and $\mathbf{b}_{r2}$, the use of $\mathbf{w}_{\text{exp}}$ will cause overflow problems under 32-bit or 64-bit integer representations. Such an mABFT approach is therefore mostly of theoretical interest, rather than of practical relevance to conventional

numerical representations used in programmable processors.

As a remedy to the dynamic range expansion caused by the weight vector of (1.2), Rexford and Jha [6] proposed the further partitioning of input subblocks into smaller blocks. Complementary to this approach, Luk *et. al.* [4, 5] proposed the weights:

$$\begin{aligned} \mathbf{w}_{\text{clinear}}^{\mathbf{T}} &= \begin{bmatrix} 1 & 2 & \dots & M \end{bmatrix} \\ \mathbf{w}_{\text{rlinear}}^{\mathbf{T}} &= \begin{bmatrix} 1 & 2 & \dots & N \end{bmatrix} \end{aligned} \tag{1.3}$$

instead of the exponential weights of 1.2, in order to allow for quadratic increase of dynamic range in $\mathbf{a}_{\text{c2}}$ and $\mathbf{b}_{\text{r2}}$ and quartic increase of dynamic range for the second row-column checksum element, r_{r2c2}. Specifically, the quartic increase in the latter is by factor $\left(\sum_{m=1}^{M} m\right) \cdot \left(\sum_{n=1}^{N} n\right)$, or $\left(\log_2\left(M + M^2\right) + \log_2\left(N + N^2\right) - 1\right)$ bits. Their work shows that, under the use of the linear weights of (1.3) and for $d = 2$, up to two SDCs per row or column of the output $M \times N$ matrix can be reliably detected and up to one SDC can be reliably corrected.

Beyond SDC mitigation for matrix products, algorithm-based techniques have also been developed for fault tolerance in compute-intensive routines as well as a FER method for fail stop failures. These methods either employ the use of checksums as proposed by Huang and Abraham, or exploit the properties of the algorithm for error detection. For example, Hoemmen and Heroux [106] as well as Chen [107] show that algorithmic properties such as convergence time and vector orthogonality can be employed within iterative methods for SDC detection. On the other hand, the use of checksums have been applied to several linear algebra and

signal processing routines including: parallel matrix products [96, 100], matrix factorization [8,108], fast Fourier transforms (FFT) [109,110] and sparse matrix-vector-matrix multiplication [11].

For these checksum-based methods, the overarching concept is the production of checksum rows and columns (or entire checksum matrices) so that the performed computation can be applied to the checksum elements alongside the input matrices or vectors. These additional elements can then be used for FER/SDC detection and correction by solving a system of linear equations if failures are detected. Therefore, most existing algorithm level approaches incur overhead due to the storage and processing of the checksum vectors or matrices. Moreover, the requirement of additional computing nodes for checksum processing in FER decreases the achievable peak performance, as less nodes are dedicated to actual input-data computations. For example, in order to mitigate a single process failure during parallel GEMM computation on an L^2 process grid, Bosilca et. al. [96] and Chen and Dongarra [100], show that $2L - 1$ processing nodes must be reserved for checksum storage and processing. The implication of this setup was that for a single fail-stop failure mitigation, 25% of processing throughput must be sacrificed when running on an 8–by-8 process grid. Improved throughput values are recorded for increasing process grid sizes. Similar results recorded in [111] show that in comparison to a fault-intolerant GEMM computation on a graphics processing unit (GPU), ABFT-protected GEMM incurs $11\% \sim 125\%$ execution time overhead for square matrix sizes ranging from $4096 \sim 32$. Finally, Pilla et al [112] reports a 41% average execution time overhead for detecting radi-

ation induced errors in a GPU performing 1D FFT computations. These results show that most existing checksum based methods only become efficient at large matrix/vector computations as well as for failure mitigation within larger cluster sizes.

Overall, for failure resilience in compute-intensive routines like GEMM and CONV, roll-forward methods are preferable to roll-back methods, as they achieve higher throughput and can immediately mitigate the effect of failures without service interruption [82, 95, 100, 113, 114].

1.1.4 Fault tolerance for Error Tolerant Systems

One of the proposals of this thesis focuses on SDC mitigation for integer matrix products, which are often encountered in multimedia applications where low precision processing may be sufficient for the desired throughput/processing results (a.k.a error tolerant systems) [40]. Thus, the question of the need for detecting and correcting errors in systems that can inherently tolerant errors often arises in the community. Indeed, detecting SDCs in error-tolerant multimedia applications like image/video rendering or pixel-level processing is of limited interest. However, it has been shown in [115–117] that there is need to differentiate between "critical/perceptible" and "uncritical/imperceptible" soft errors in various applications. The former category comprising applications that process compacted data (or data that is subsequently used as inputs for further processing). For example, many video retrieval, video database and video editing applications exhibit critical dependency on the accu-

racy of large matrix products between non-pixel values, e.g., compacted descriptor data or compact projections [9, 118]. In such cases, mitigation of soft errors is imperative, especially when outputs from such sections are reused for subsequent computations and processing, e.g., the generation and presentation of a ranked list of results based on sorting and thresholding of the GEMM outputs. Therefore, despite the error-tolerance of pixel-based video processing applications, the sensitivity of non-pixel based processing in video systems stresses the need for error tolerance in these applications, especially since video retrieval and video database applications are now gaining significant traction.

1.2 Thesis Objective

In this thesis, our aim is to introduce a new class of low-complexity, algorithm and application-specific fault tolerant methods. The proposed designs are expected to be deployable either as a stand-alone reliability mechanism, or as a component within existing fault tolerant system designs within a cross-layer resilience framework [18, 46, 78]. Importantly, by reducing/eliminating all forms of physical or temporal redundancy synonymous with generic fault tolerant designs [89, 95], the proposals of this thesis are expected to incur lower implementation overhead, reduced resource utilization and increased processing throughput. For example, because the GEMM performed within multimedia processing [32] and linear system solutions [34] have different precision and throughput requirements, a generic replication or checksum-based fault tolerant

GEMM may degrade the performance requirements of such applications. Therefore, by focusing on the class of error-tolerant applications where reduced precision processing is acceptable, our goal is to design reliability solutions that are fast, energy efficient and tailored for increased processing throughput.

1.3 Thesis Structure

The remaining parts of this thesis consists of four chapters. In Chapter 2, we introduce the concept of numerical packing as proposed in previous work for throughput acceleration and show how a redesign of the GEMM algorithm introduces reliability at a reduced cost. Furthermore, the theoretical complexity and reliability aspects of numerical packing for SDC detection and correction are investigated in comparison to existing methods of checksum generation and modular redundancy. We also analyze in this chapter, the execution time and energy consumption overhead of SDC mitigation methods via an SDC-injection campaign experiment using an open source low-level virtual machine (LLVM) based fault injection tool, the Kontrollable Utah LLVM Fault Injector (KULFI) [3, 119].

In Chapter 3, we present a numerical packing based single core/processor failure FER algorithm for sum-of-product computations. Given that our algorithm can only be used for integer data computations, we focus on large-scale, low-latency, multimedia applications that require high-throughput integer-to-integer sum-of-product computations [40]. To

quantify the complexity of the proposed approach, we derive its overhead in terms of arithmetic operations in comparison to the equivalent checksum-based method. We also present experimental results on *(i)* an 18-core, shared-memory, AWS EC2 instance[3], and *(ii)* a StarCluster [120] of AWS EC2 spot instances that are terminated and migrated to AWS EC2 on-demand instances for the duration that the spot price exceeds a predetermined threshold. For the former, we demonstrate that the proposed method achieves substantially-higher peak performance against the equivalent FER method based on checksums, both under failure-free and failure-occurring conditions. For the latter, we show that our approach provides for substantial reduction in deployment cost, especially in comparison to the failure-intolerant approach that cannot use spot instances.

In order to generalize our FER methods to recover from multiple failures, we propose a new FER method for linear and sesquilinear operations performed in integer data streams in Chapter 4. We present generalization for the mitigation of K failures when L data streams (also processing core data) are "entangled" together and show that the proposed algorithm can also be used for SDC detection and correction in the absence of failures.

Finally, we conclude this thesis in Chapter 5 while outlining possible future research directions based on the proposals of this work.

[3]This is a `c4.8xlarge` AWS EC2 instance composed of multiple dual-nanocore physical processors (Intel Haswell) with hyperthreading.

Chapter 2

Numerical Packing for Fault-tolerance in GEMM

THE generic matrix multiply (GEMM) routine comprises the compute and memory-intensive part of many information retrieval, relevance ranking and object recognition systems that process integer inputs. Because of the prevalence of GEMM in these prominent applications, ensuring its robustness to transient hardware faults is of paramount importance for highly-reliable systems. Mitigation techniques for SDCs are especially important for *matrix products* performed during descriptor matching, power iterations, backpropagation, transform decompositions, random projections, kernel methods, covariance matrix calculations and block Lanczos iterations within information retrieval, object detection and tracking, machine learning, and classification applications [9,32,39,118,121–125]. This is because: *(i)* matrix-product computations comprise the bulk of processing in such applications; *(ii)* subsequent pro-

cessing stages apply low-complex-yet-noise-sensitive thresholding, sorting and clustering operations [9, 116, 117, 126] and any undetected SDCs in the matrix-product stage can have severe repercussions in the final results. Many such matrix products utilize integer inputs and are typically performed with an integer generic matrix multiply (GEMM) routine, or the single- or double-precision floating-point GEMM (sGEMM or dGEMM) routines of a high-performance mathematics kernel library (MKL) [35, 36]. Thus, ensuring that integer matrix products remain reliable against SDCs is of paramount importance for such multimedia systems.

In Section 2.1, we summarize the operation of high-performance GEMM routine highlighting existing optimization techniques available on modern processors. We describe the proposed algorithm of numerical packing in Section 2.2, including the requirement of a low-cost checksum inner-product computation for increased reliability. Section 2.3 provides discussion on dynamic range, computational complexity and reliability of existing SDC mitigation techniques. We make some concluding remarks in Section 2.6.

2.1 Structure of Generic Matrix Multiplication (GEMM) Routine

Consider the GEMM design depicted in Figure 2.1, which follows the general structure found in optimized linear algebra kernels [36] (e.g Intel

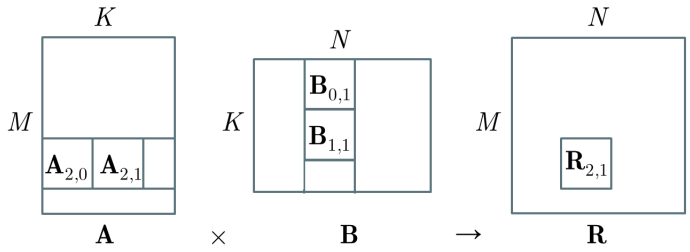


Figure 2.1: Top-level processing of GEMM $\mathbf{R} = \mathbf{A} \times \mathbf{B}$, highlighting the input subblocks used for the computation of the subblock result $\mathbf{R}_{2,1}$

MKL, ATLAS, gotoBLAS). The application calls GEMM for an $M \times K$ by $K \times N$ matrix multiplication, which is further subdivided into $T \times T$ "inner-kernel" matrix products. For our approach, T is specified by ($k \in \mathbb{N}^{\star}$):

$$T = 2k \times \frac{\text{SIMD}_{\text{bits}}}{b_{\text{repr}}} \tag{2.1}$$

with: $\text{SIMD}_{\text{bits}}$ the number of bits of each SIMD register ($\text{SIMD}_{\text{bits}} = 256$ in this work); $b_{\text{repr}} \in \{32, 64\}$ the number of bits for floating-point or integer representations. The inner-kernel result $\mathbf{R}_{2,1}$ of the example shown in Figure 2.1 comprises the sum of multiple subblock multiplications $\mathbf{A}_{2,t}\mathbf{B}_{t,1}$:

$$\mathbf{R}_{2,1} = \sum_{t=0}^{\frac{K}{T}-1} \mathbf{A}_{2,t}\mathbf{B}_{t,1}. \tag{2.2}$$

If the matrices' dimensions are not multiples of T, some "cleanup" code [36] is applied at the borders to complete the inner-kernel results of the overall matrix multiplication. This separation into top-level processing and subblock-level processing is done for efficient cache utilization. Specifically, during the initial data access of GEMM for top-level processing, data in matrix $\mathbf{A}$ and $\mathbf{B}$ is reordered into block major format: for each $T \times T$ pair of subblocks $\mathbf{A}_{i,t}$ and $\mathbf{B}_{t,j}$ multiplied to produce inner-kernel re-

sult $\mathbf{R}_{i,j}$, $0 \leq t < \frac{K}{T}$, $0 \leq i < \frac{M}{T}$, $0 \leq j < \frac{N}{T}$, the input data within $\mathbf{A}_{i,t}$ and $\mathbf{B}_{t,j}$ is reordered in rowwise and columnwise raster manner, respectively. Thus, sequential data accesses are performed during each subblock matrix multiplication and this enables the use of SIMD instructions, thereby leading to significant acceleration. A detailed exposition on matrix ordering for GEMM and available optimizations can be found in [127].

From this description it is evident that the top-level processing simply administers the computation (and, optionally, error detection) at the subblock level and the core operations are performed within each subblock independently, before being aggregated to produce the final results.

2.2 Numerical Packing: A New Numerical Representation Method for Information Redundancy

The concept of numerical packed processing is based on the fact that multiple low precision data inputs can be stacked within a single input, in such a way as to ensure simultaneous and accelerated data processing [42, 128, 129]. Packed processing algorithms for throughput acceleration in convolution [128] show acceleration gains between 27% and 158% in comparison to conventional processing, with acceleration for matrix product [129], block matching [130] and cross correlation [42, 130] following similar trends.

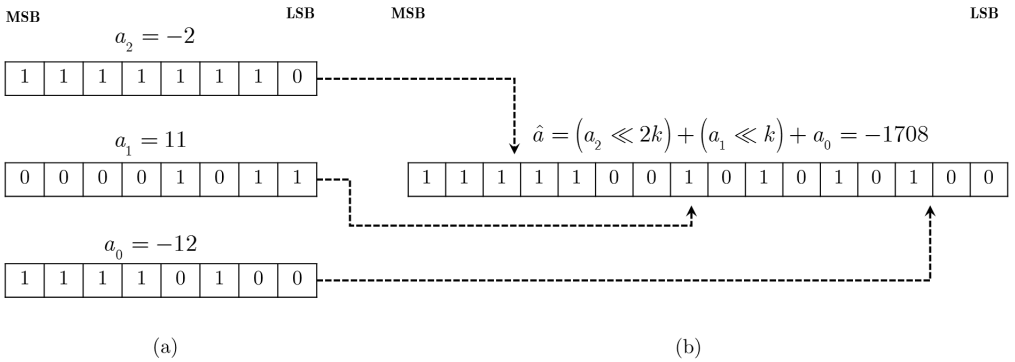


Figure 2.2: Bit layout of signed inputs packed within a 16-bit integer data representation via arithmetic shifting using a packing factor, $k = 5$. MSB/LSB shows the most/least significant bit locations.

Since the proposed method is built upon the concept of numerical packing, we briefly summarize this idea within integer data representation. Consider three integer inputs, a_0, a_1, a_2, that must be processed with an integer kernel, such that the range of output values, r_0, r_1, r_2 is within $[-16, 16]$. Conventional implementation on general purpose processors using, for example, the C programming language, would represent each a_i (and the corresponding output r_i) using 8-bit `signed char` representation. We show in Fig. 2.2(a), the two's complement integer representation of these inputs, a_0, a_1, a_2.

In order to achieve packed processing, we define a packing factor, k given by:

$$k > \log_2\left(\max_{\forall i}|r_i|\right) + 1 \tag{2.3}$$

$$Nk \leq W \tag{2.4}$$

where N is the number of inputs packed within one number representation, and W the data bitwidth used for packed processing.

The packing factor of (2.3) and (2.4) ensure that the total number of bits do not exceed the available bitwidth of the utilized integer representation. Though the packed representation of Fig. 2.2(b) depicts an overlap especially for negative input values, we show in the sections that follow, that all output values are indeed extractable using basic arithmetic operations.

Packed processing is achieved via arithmetic shift and summation operations. Specifically, the packed input $\widehat{a}$ is computed by:

$$\widehat{a} = \sum_{i=0}^{N-1} a_i \ll (i \cdot k) \tag{2.5}$$

Kernel processing can subsequently be carried out using $\widehat{a}$ in place of a_i $(0 \leq i < N)$, provided the constraints of (2.3) and (2.4) are maintained. The inverse operation (unpacking) is performed on the packed output, $\widehat{r}$ using the reverse bitshift operations and conditional subtractions of previously-extracted results that depend on the sign of the extracted information (cf. Section III of [131]).

We now focus on the numerical packing construct for GEMM computation and its utilization for fault tolerance in GEMM.

2.2.1 Numerical Packing for GEMM

To facilitate the exposition, we shall be referring to the illustrations of Figure 2.3(d)–(g) and Figure 2.4, which present the steps of the described methods for the elementary case of a 2×2 matrix produced via a 2×1 by

1×2 vector product. In addition, the conventional (fault-intolerant) approach is illustrated in Figure 2.3(a)–(c). Finally, beyond this elementary case, our analytic exposition will be based on the general case of a $T \times T$ subblock product.

In packing for integer subblock multiplication $\mathbf{R} = \mathbf{AB}$, only one input subblock is packed, i.e., either $\mathbf{A}$ or $\mathbf{B}$ (asymmetric packing). This selection does not affect the performance in the case of GEMM, as both subblocks have been reordered in block-major format. Assuming $\mathbf{A}$ is chosen, the packing process creates block $\widehat{\mathbf{A}}$ with $\frac{T}{2} \times T$ coefficients given by ($\forall m, n : \ 0 \leq m, n < T, \ \widehat{m} = \left\lfloor \frac{m}{2} \right\rfloor$):

$$\widehat{a}_{\widehat{m},n} = \mathcal{S}_k \left\{ a_{2\widehat{m},n} \right\} + a_{2\widehat{m}+1,n} \tag{2.6}$$

where k is the utilized packing coefficient, $k \in \mathbb{N}^{\star}$, and $\mathcal{S}_k \{a\}$ is the left bit shift operator defined in the notational conventions table of pg. xix. The utilized value for k depends on the maximum possible value of the matrix product, as it will be elaborated in the following. An illustration of the packing of (2.6) for the two elements of a 2×1 vector $\mathbf{a}$ is given in Figure 2.3(d), i.e., for $n = \widehat{m} = 0$.

Notice that (2.6) operates along the columns of $\widehat{\mathbf{A}}$ in order to pack rows $2\widehat{m}$ and $2\widehat{m} + 1$ together. This means that, in order to use integer SIMD instructions for accelerated computation of (2.6), we can group $\frac{\text{SIMD}_{\text{bits}}}{b_{\text{repr}}}$ consecutive elements of each row and apply each arithmetic shift and add operation of (2.6) to them using appropriate SIMD instructions. Once (2.6) is completed, processing occurs via $\widehat{\mathbf{R}} = \widehat{\mathbf{A}}\mathbf{B}$ ($\forall m, n : \ 0 \leq m, n < T, \ \widehat{m} =$

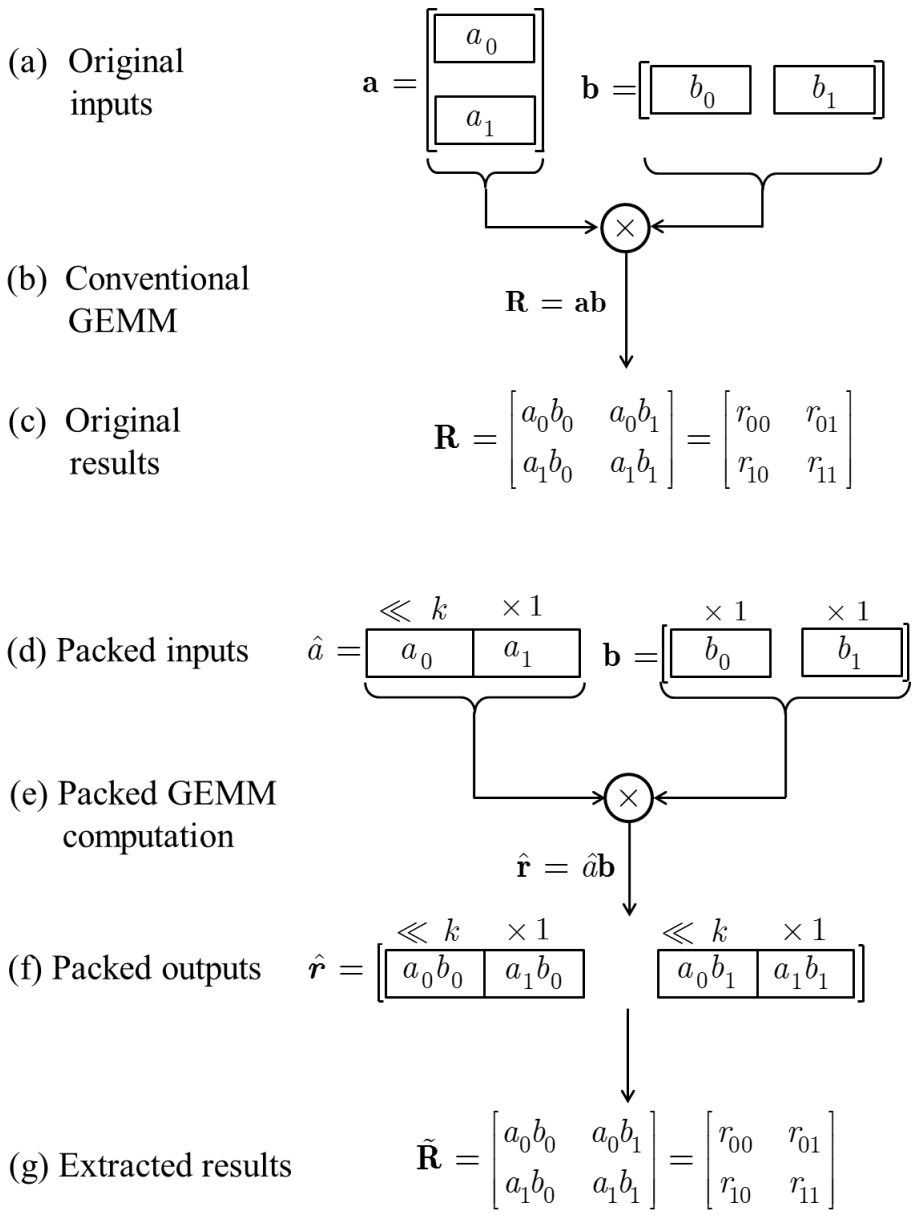


Figure 2.3: (a)–(c) Conventional integer subblock multiplication for the case of a 2×1 by 1×2 vector product. (d)–(g) Packing for the same product with packing coefficient $k = 10$. The partitioning within the rectangles shows the location (shifts by k bits) of inputs/outputs when packed within a single number.

$\left\lfloor \frac{m}{2} \right\rfloor$):

$$\begin{aligned} \widehat{r}_{\widehat{m},n} &= \sum_{j=0}^{T-1} \widehat{a}_{\widehat{m},j} b_{j,n} \\ &= \left(\mathcal{S}_k \left\{ \sum_{j=0}^{T-1} a_{2\widehat{m},j} b_{j,n} \right\} \right) + \sum_{j=0}^{T-1} a_{2\widehat{m}+1,j} b_{j,n} \end{aligned} \tag{2.7}$$

The packed output of (2.7) contains both rows of the results, $\widehat{r}_{2\widehat{m},n}$ and $\widehat{r}_{2\widehat{m}+1,n}$, packed together. An example is shown in Figure 2.3(f) for $n = \widehat{m} = 0$. Importantly, if they do not overlap in the packed representation and the numerical representation used can accommodate both packed outputs, both rows can be computed concurrently via (2.7). For the packed processing of two inputs shown in (2.6) and (2.7), these two conditions are met when the packing coefficient satisfies the below constraint [41, 129, 130]:

- For 32/64-bit integer representation: $k > \log_2 \left(\max_{\forall m,n} |r_{m,n}| \right) + 1$ and $2k \leq W$, with $W \in \{32, 64\}$.

Any high-performance $\frac{T}{2} \times T$ by $T \times T$ subblock code for 32/64-bit GEMM (integer or floating-point) can be used for the computation of (2.7). Following the completion of the processing, unpacking of the results can be performed by the following process [41, 129, 130] ($\forall m, n : \ 0 \leq m, n < T, \ \widehat{m} = \left\lfloor \frac{m}{2} \right\rfloor$).

The first output, $\widetilde{r}_{2\widehat{m},n}$, which is packed at the most significant bits of $\widehat{r}_{\widehat{m},n}$, is extracted by:

$$\widetilde{r}_{2\widehat{m},n} = \mathcal{S}_{-k} \left\{ \widehat{r}_{\widehat{m},n} \right\} \tag{2.8}$$

The extracted output is then removed from $\widehat{r}_{\widehat{m},n}$:

$$\widetilde{r}_{2\widehat{m}+1,n} = \widehat{r}_{\widehat{m},n} - \mathcal{S}_k\left\{\widetilde{r}_{2\widehat{m},n}\right\}, \tag{2.9}$$

The result is then converted into its signed representation. Specifically, if $\widetilde{r}_{2\widehat{m}+1,n} \geq 2^{k-1}$, then $\widetilde{r}_{2\widehat{m}+1,n} \leftarrow (\widetilde{r}_{2\widehat{m}+1,n} - 2^k)$.

Finally if $\widetilde{r}_{2\widehat{m}+1,n} < 0$, then the result extracted from the most-significant bits must be incremented by one, i.e.: $\widetilde{r}_{2\widehat{m},n} \leftarrow (\widetilde{r}_{2\widehat{m},n} + 1)$.

2.2.2 Proposed Fault-tolerant Packing

We utilize the packing concept in order to provide highly-reliable integer matrix products within each GEMM call. Our method uses standard 64-bit integer representations and off-the-shelf GEMM subblock kernels to process 32-bit inputs, as detailed in the following[1].

Packing Process

The proposed approach performs **two packings** of matrix $\mathbf{A}$ into $\widehat{\mathbf{A}}_i$ and $\widehat{\mathbf{A}}_j$ and one packing of $\mathbf{B}$ into $\widehat{\mathbf{B}}$, as shown in the example of Figure 2.4(a). The figure illustrates only the case of an elementary 2×1 by 1×2 vector product with $k = 10$ and, due to the small input vector dimensions, the packed matrices are in fact scalars $\widehat{a}_i$, $\widehat{a}_j$ and $\widehat{b}$. For the general case of a $T \times T$ by $T \times T$ subblock product, the process is identical for all other pairs

[1] In general, double bit-width representation is required by the proposed method in order to accommodate a comparable data range.

of inputs within **A** and **B** and is expressed by ($\forall m, n : \ 0 \leq m, n < T, \ \widehat{m} = \left\lfloor \frac{m}{2} \right\rfloor \ \widehat{n} = \left\lfloor \frac{n}{2} \right\rfloor$):

$$\widehat{a}_{i,\widehat{m},n} = \mathcal{S}_k \left\{ a_{2\widehat{m},n} \right\} - a_{2\widehat{m}+1,n} \tag{2.10}$$

$$\widehat{a}_{j,\widehat{m},n} = \mathcal{S}_k \left\{ a_{2\widehat{m}+1,n} \right\} - a_{2\widehat{m},n} \tag{2.11}$$

$$\widehat{b}_{m,\widehat{n}} = \mathcal{S}_k \left\{ b_{m,2\widehat{n}} \right\} + b_{m,2\widehat{n}+1} \tag{2.12}$$

Each packing "stacks" two inputs together in a single 64-bit integer number (assuming 32-bit inputs) and can be done during initial reading and reordering of each subblock in GEMM [36, 129]. Notice that, in comparison to conventional packing presented in the previous section [and illustrated in Figure 2.3(d)–(g)]:

- (2.10) and (2.11) invert the sign of the elements of **A** that are left shifted by k-bits;
- **A** is packed twice with reverse "ordering", as shown in (2.10) and (2.11);
- **B** is packed as well.

This setup enables packed processing to be used for error detection.

Packed GEMM computations

Two $\left(\frac{T}{2} \times T\right) \times \left(T \times \frac{T}{2}\right)$ subblock multiplications ensue via the use of two standard subblock GEMM calls (using either 64-bit integer or 64-bit floating point representation), producing all required results, as well as a

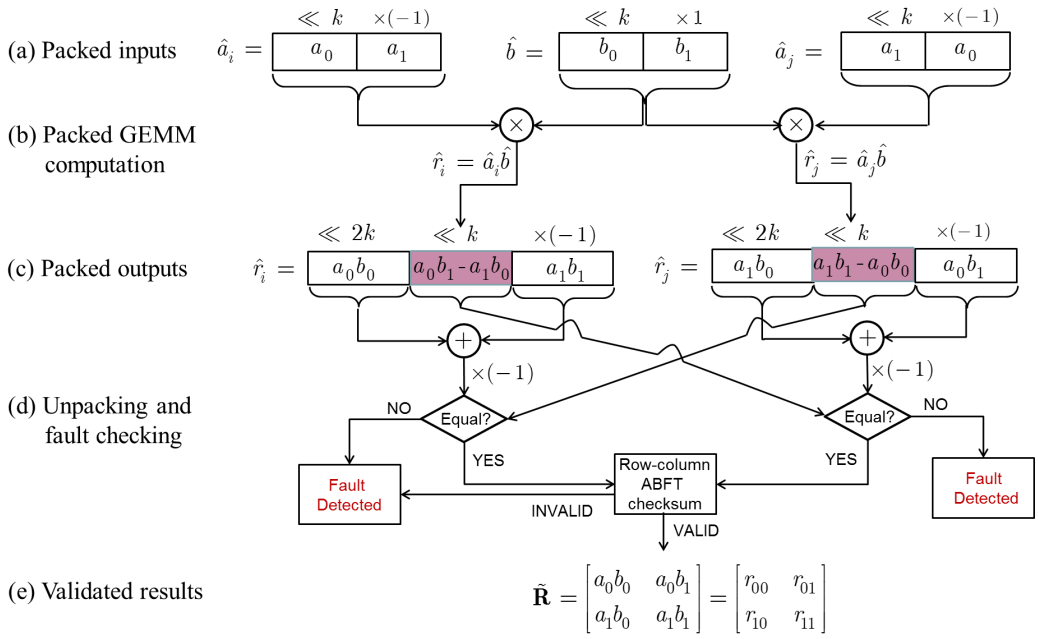


Figure 2.4: Illustration of highly-reliable integer subblock multiplication via numerical packing for the case of a 2×1 by 1×2 vector product with packing coefficient $k = 10$. The partitioning within the rectangles shows the location (shifts by k or $2k$ bits) of inputs/outputs when packed within a single number.

number of "entangled" results within the numerical representation of the packed outputs. An example for the two packed products of elementary 2×1 by 1×2 vectors is shown in Figure 2.4(b)–(c). For the general case of an $T \times T$ by $T \times T$ subblock product, the elements packed within $\widehat{\mathbf{R}}_i$ and $\widehat{\mathbf{R}}_j$

are expressed mathematically by ($\forall m, n:\ 0 \leq m, n < T,\ \widehat{m} = \left\lfloor \frac{m}{2} \right\rfloor,\ \widehat{n} = \left\lfloor \frac{n}{2} \right\rfloor$):

$$\begin{aligned}
\widehat{r}_{i,\widehat{m},\widehat{n}} &= \sum_{t=0}^{T-1} \widehat{a}_{i,\widehat{m},t}\widehat{b}_{t,\widehat{n}} \\
&= \mathcal{S}_{2k}\left\{\sum_{t=0}^{T-1} a_{2\widehat{m},t}b_{t,2\widehat{n}}\right\} - \mathcal{S}_k\left\{\sum_{t=0}^{T-1} a_{2\widehat{m}+1,t}b_{t,2\widehat{n}}\right\} \\
&\quad +\mathcal{S}_k\left\{\sum_{t=0}^{T-1} a_{2\widehat{m},t}b_{t,2\widehat{n}+1}\right\} - \sum_{t=0}^{T-1} a_{2\widehat{m}+1,t}b_{t,2\widehat{n}+1} \\
&= \mathcal{S}_{2k}\left\{\widetilde{r}_{2\widehat{m},2\widehat{n}}\right\} + \mathcal{S}_k\left\{\widetilde{r}_{2\widehat{m},2\widehat{n}+1} - \widetilde{r}_{2\widehat{m}+1,2\widehat{n}}\right\} \\
&\quad - \widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1}
\end{aligned} \tag{2.13}$$

$$\begin{aligned}
\widehat{r}_{j,\widehat{m},\widehat{n}} &= \sum_{t=0}^{T-1} \widehat{a}_{j,\widehat{m},t}\widehat{b}_{t,\widehat{n}} \\
&= \mathcal{S}_{2k}\left\{\sum_{t=0}^{T-1} a_{2\widehat{m}+1,t}b_{t,2\widehat{n}}\right\} - \mathcal{S}_k\left\{\sum_{t=0}^{T-1} a_{2\widehat{m},t}b_{t,2\widehat{n}}\right\} \\
&+ \mathcal{S}_k\left\{\sum_{t=0}^{T-1} a_{2\widehat{m}+1,t}b_{t,2\widehat{n}+1}\right\} - \sum_{t=0}^{T-1} a_{2\widehat{m},t}b_{t,2\widehat{n}+1} \\
&= \mathcal{S}_{2k}\left\{\widetilde{r}_{2\widehat{m}+1,2\widehat{n}}\right\} + \mathcal{S}_k\left\{\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1} - \widetilde{r}_{2\widehat{m},2\widehat{n}}\right\} \\
&\quad - \widetilde{r}_{2\widehat{m},2\widehat{n}+1}
\end{aligned} \tag{2.14}$$

Unpacking of the Results

Subsequently, unpacking occurs within each element of $\widehat{\mathbf{R}}_i$ and $\widehat{\mathbf{R}}_j$ based on the process presented below, which extends the unpacking of the previous section in order to handle the three packed results within each output $\widehat{r}_{i,\widehat{m},\widehat{n}}$ and $\widehat{r}_{j,\widehat{m},\widehat{n}}$, illustrated in Figure 2.4(d).

We extract the components packed within $\widehat{r}_i$ following the unpacking process described in the previous subsection, with $\tilde{\varepsilon}_{(k_0,k_1),(k_2,k_3)}$ indicating the entanglement (i.e., superposition within the representation) of elements

$\widetilde{r}_{k_0,k_1}$ and $\widetilde{r}_{k_2,k_3}$. The first output, $\widetilde{r}_{2\widehat{m},2\widehat{n}}$, which is packed at the most-significant bits of $\widehat{r}_i$, is extracted by:

$$\widetilde{r}_{2\widehat{m},2\widehat{n}} = \mathcal{S}_{-2k}\left\{\widehat{r}_i\right\}, \tag{2.15}$$

The extracted output is then removed from $\widehat{r}_i$ and relevant bitwise operations ensue to determine the sign of the entangled output, $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})}$. We present the details of this sign check by defining two parameters, $t_{i,1}$ and $t_{i,2}$ as outlined below:

$$t_{i,1} = \widehat{r}_i - \left[\mathcal{S}_{2k}\left\{\widetilde{r}_{2\widehat{m},2\widehat{n}}\right\}\right], \tag{2.16}$$

$$t_{i,2} = \mathcal{S}_{-k}\left\{t_{i,1}\right\}, \tag{2.17}$$

If $t_{i,2} \geq 2^{k-1}$, then $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})} = t_{i,2} - 2^k$, else $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})} = t_{i,2}$.
If $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})} < 0$, then : $\widetilde{r}_{2\widehat{m},2\widehat{n}} \leftarrow (\widetilde{r}_{2\widehat{m},2\widehat{n}} + 1)$.

Subsequently, the intermediate result extracted by (2.17) is also removed from the packed representation and a similar check is performed in order to extract the signed representation of $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1}$, i.e.:

$$t_{i,1} \leftarrow \left[t_{i,1} - \mathcal{S}_k\left\{t_{i,2}\right\}\right] \tag{2.18}$$

If $t_{i,1} \geq 2^{k-1}$, then $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1} = 2^k - t_{i,1}$, else $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1} = -t_{i,1}$. Finally, similarly as before, if $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1} > 0$, then the the entangled output, $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})}$ is incremented by one: $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})} \leftarrow \left(\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})} + 1\right)$. We extract the components packed within $\widehat{r}_j$ via the following process, which is

identical to the process described for $\widehat{r}_i$:

$$\widetilde{r}_{2\widehat{m}+1,2\widehat{n}} = \mathcal{S}_{-2k}\left\{\widehat{r}_j\right\}, \tag{2.19}$$

$$t_{j,1} = \widehat{r}_j - \left[\mathcal{S}_{2k}\left\{\widetilde{r}_{2\widehat{m}+1,2\widehat{n}}\right\}\right], \tag{2.20}$$

$$t_{j,2} = \mathcal{S}_{-k}\left\{t_{j,1}\right\}, \tag{2.21}$$

If $t_{j,2} \geq 2^{k-1}$, then $\tilde{\varepsilon}_{(2\widehat{m}+1,2\widehat{n}+1),(2\widehat{m},2\widehat{n})} = t_{j,2} - 2^k$, else $\tilde{\varepsilon}_{(2\widehat{m}+1,2\widehat{n}+1),(2\widehat{m},2\widehat{n})} = t_{j,2}$. If $\tilde{\varepsilon}_{(2\widehat{m}+1,2\widehat{n}+1),(2\widehat{m},2\widehat{n})} < 0$, then $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}} \leftarrow (\widetilde{r}_{2\widehat{m}+1,2\widehat{n}} + 1)$. Subsequently:

$$t_{j,1} \leftarrow \left[t_{j,1} - \mathcal{S}_k\left\{t_{j,2}\right\}\right]. \tag{2.22}$$

If $t_{j,1} \geq 2^{k-1}$, then $\widetilde{r}_{2\widehat{m},2\widehat{n}+1} = 2^k - t_{j,1}$, else $\widetilde{r}_{2\widehat{m},2\widehat{n}+1} = -t_{j,1}$. If $\widetilde{r}_{2\widehat{m},2\widehat{n}+1} > 0$, then $\tilde{\varepsilon}_{(2\widehat{m}+1,2\widehat{n}+1),(2\widehat{m},2\widehat{n})} \leftarrow \left(\tilde{\varepsilon}_{(2\widehat{m}+1,2\widehat{n}+1),(2\widehat{m},2\widehat{n})} + 1\right)$.

If:

- sufficient spacing is provisioned via the packing coefficient k so that no overlapping (or "invasion") of results occurs within the numerical representation [41, 129, 130], and
- all three packed results fit within the utilized numerical representation,

then the results are guaranteed to be recoverable.

Relating to the second requirement, if $3k \leq W$, then the unpacking process determines the correct value of each output. This condition imposes that:

- $k \leq 21$ for $W = 64$ in 64-bit integer GEMM. Given that the sum of two outputs must be accommodated for the packed results shifted by k-bits (i.e., the entangled results) and one bit must be provided to allow for the sign information to be preserved within the packed integer representation, this allows for up to $\pm 2^{19}$ output dynamic range without any approximation. This means that 12 bits of dynamic range are sacrificed in comparison to 32-bit integer GEMM that allows for up to $\pm 2^{31}$ bits, i.e., loss of 37.5% of the bitwidth. This loss is substantially smaller than the corresponding bitwidth loss of ABFT and mABFT, reported in (2.24) and (2.25), respectively. Since the computational complexity and error detectability of the proposed algorithm is independent of the packing factor, we set k to the maximum achievable value, i.e, $k = 20$ and $k = 9$ for 64 and 32-bit integer processing, respectively. Further details on the mathematics of the analytic calculation of the maximum packing factor are found in related work [41, 129, 130].

Error Detection by Post-Entanglement followed by Row-Column ABFT Checksum Validation

The advantage of the proposed packing-based GEMM is that, we not only obtain the results, **but we can also validate them by post-entangling**, i.e., doing $\widetilde{r}_{2\widehat{m},2\widehat{n}+1} - \widetilde{r}_{2\widehat{m}+1,2\widehat{n}}$ and $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1} - \widetilde{r}_{2\widehat{m},2\widehat{n}}$ and comparing these with the entangled results $\tilde{\varepsilon}_{(2\widehat{m},2\widehat{n}+1),(2\widehat{m}+1,2\widehat{n})}$ and $\tilde{\varepsilon}_{(2\widehat{m}+1,2\widehat{n}+1),(2\widehat{m},2\widehat{n})}$, respectively. An example is shown in Figure 2.4(d)–(e) for $\widehat{m} = \widehat{n} = 0$. If differences are detected, the higher level routine (top-level processing) can be notified

and a decision can be made by the application on whether to recompute the erroneous results or not.

One weakness of the post-entanglement check is that, however unlikely, it is still possible that an SDC occurs in a location in the packed results $\widehat{\mathbf{R}}_i$ and $\widehat{\mathbf{R}}_j$ in a way that the operation $\widetilde{r}_{2\widehat{m},2\widehat{n}+1} - \widetilde{r}_{2\widehat{m}+1,2\widehat{n}}$ or $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1} - \widetilde{r}_{2\widehat{m},2\widehat{n}}$ does not detect it. Specifically, this would be the case if *both* the extracted results $\widetilde{r}_{2\widehat{m},2\widehat{n}+1}$ and $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}}$ were affected by an additive noise term δ, with $\delta \in \mathbb{Z}^{\star}$. To detect the occurrence of such pathological cases, we simply utilize the conventional row-column ABFT checksum by calculating a single inner product between the sum of the columns of $\mathbf{A}$ with the sum of the rows of $\mathbf{B}$:

$$r_{\text{rc}} = \sum_{k=0}^{K-1} \left(\sum_{i=0}^{M-1} a_{i,k} \cdot \sum_{j=0}^{N-1} b_{k,j} \right), \tag{2.23}$$

which, similar to ABFT, can be derived during the input subblock reordering. Then, after checking for SDCs based on the post-entanglement check (and recalculating all detected SDCs), we can check if r_{rc} produced by (2.23) agrees with the sum of all the elements of $\mathbf{R}$: $\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} r_{i,j}$. It is straightforward to derive the latter when the output matrix is returned via a raster-scan and summation. This checksum ensures no such SDCs remain undetected.

The overall ABFT-based row-column checksum process requires producing only a single checksum per integer matrix product, i.e., r_{rc} of (2.23), and simply adding the results in the order they are returned to the top-level processing of GEMM. Our experiments will demonstrate that the overall error checking process (packing, unpacking, post-entanglement

check and row-column ABFT checksum) incurs execution time overhead similar to that of ABFT.

2.3 Discussion

2.3.1 Dynamic Range Aspects of ABFT in GEMM

As described in Section 1.1.3, all ABFT methods specifically tailored for GEMM computations [96, 97, 101, 104, 111] append the input subblocks with (redundant) checksum vectors (rows or columns), denoted by $\mathbf{a}_c$, $\mathbf{b}_r$ in Figure 1.2 and highlighted in color. Due to the fact that the checksum inputs $\mathbf{a}_c$ and $\mathbf{b}_r$ are the column-wise or row-wise summation of inputs, the elements of the column and row checksums, $\mathbf{r}_c$ and $\mathbf{r}_r$, will have increased dynamic range by $\log_2 M$ and $\log_2 N$ bits respectively for an $M \times K$ by $K \times N$ integer GEMM. In addition, given that r_{rc} is the sum of the elements of $\mathbf{r}_r$ and $\mathbf{r}_c$, this element will require $\log_2 MN$ additional bits. Therefore, in comparison to the conventional (fault-intolerant) GEMM computation and for $M = N = K = 288$ and 32-bit integer representations, ABFT incurs loss of

$$\left.\frac{\log_2 M}{16}\right|_{M=288} \times 100\% \simeq 51.06\% \tag{2.24}$$

of output dynamic range in order to accommodate all checksum results.

Furthermore, increased reliability offered by mABFT methods also come at the cost of reduced output dynamic range of matrix outputs. For ex-

ample, via the combination of partitioning of $M \times K$ and $K \times N$ input blocks into $P \times P$ blocks[2] [6] with $P = 16$, and the use of two checksum rows/columns per partition (using the linear weights of (1.3)), mABFT is capable of correcting an SDC per partition row/column of the output matrix product. However, this comes at the loss of

$$\left.\frac{\log_2\left(P^2+P\right)-1}{16}\right|_{P=16} \times 100\% \simeq 44.3\% \tag{2.25}$$

of output dynamic range for 32-bit integer representations in comparison to the conventional (fault-intolerant) GEMM computation.

By implication, the maximum absolute value of data inputs, a_i, supported by the conventional 288×288-by-288×288 GEMM computation using 32-bit integer representation is bounded by: $|a_i| \leq 2730$. However, when same computation is to be computed reliably using the ABFT and mABFT constructs, the absolute value of input data must not exceed $|a_i| = 9$ and $|a_i| = 85$ respectively. These allowable absolute values is further reduced to $|a_i| = \{1365, 1, 85\}$ for the conventional, ABFT and mABFT based GEMM computations of 1152×1152 matrices, respectively. Therefore, the practical applicability of the single checksum row and column of ABFT for GEMM computation diminishes as the matrix size increases.

[2]For simplicity of exposition, we assume that M, N and K are multiples of the partition size, P.

2.3.2 Theoretical Analysis of Computational Complexity

The following two propositions summarize the arithmetic operations (addition, multiplication and comparison) for the computation of the proposed method for SDC mitigation in GEMM in reference to ABFT, mABFT and DMR. Note that the operation counts for the proposed approach have been scaled by two to account for the fact that the use of double-bitwidth representations (due to the use of packing) reduces the processing throughput by a factor of two. We focus on the case of square matrices for all analysis in this section as rectangular matrices (i.e $M \neq K$, $M \neq N$) do not possess any complexity property of interest.

Proposition 2.1. *In the absence of SDCs within a fault tolerant* $M \times M$-by-$M \times M$ *GEMM computation, the number of arithmetic operations of the proposed method is:*
(i): $\left(\frac{11M^2}{2M^3+7M^2-1}\right) \cdot 100\%$ *more than that of ABFT;*
(ii): $\left(\frac{34M^3-433M^2+64}{162M^3+719M^2}\right) \cdot 100\%$ *less than that of mABFT;*
(iii): $\left(\frac{2M^3-19M^2+1}{4M^2-M^2}\right) \cdot 100\%$ *less than that of DMR, with all comparisons plotted in Fig. 2.5(a) for* $M \in [32, 1152]$.

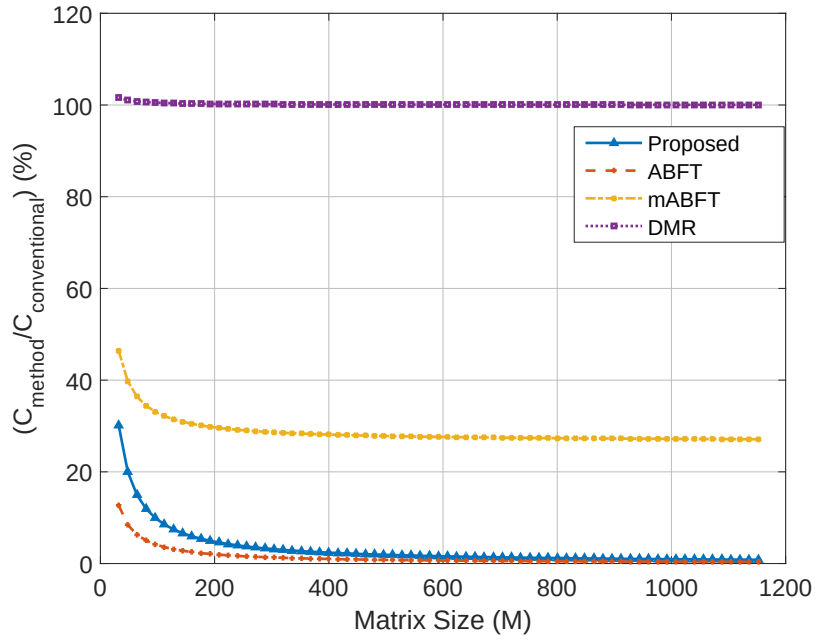
Proof. Given two $M \times M$ matrices $\mathbf{A}$ and $\mathbf{B}$, the proposed method requires $\frac{3M^2}{2}$ addition/subtraction operations (see (2.10) ~ (2.12) ignoring all arithmetic shift operations) to generate the $\frac{M}{2} \times M$ packed matrices $\widehat{\mathbf{A}}_i$, $\widehat{\mathbf{A}}_j$ and $M \times \frac{M}{2}$ matrix $\widehat{\mathbf{B}}$ since each element of the packed inputs is computed with an addition and a bit-shift operation with the packing factor, k.

The GEMM computation of $\widehat{\mathbf{R}}_i = \widehat{\mathbf{A}}_i\widehat{\mathbf{B}}$ and $\widehat{\mathbf{R}}_j = \widehat{\mathbf{A}}_j\widehat{\mathbf{B}}$ subsequently requires $\frac{2M^3-M^2}{2}$ operations as both $\widehat{\mathbf{R}}_i$ and $\widehat{\mathbf{R}}_j$ are $\frac{M}{2} \times \frac{M}{2}$ matrices.

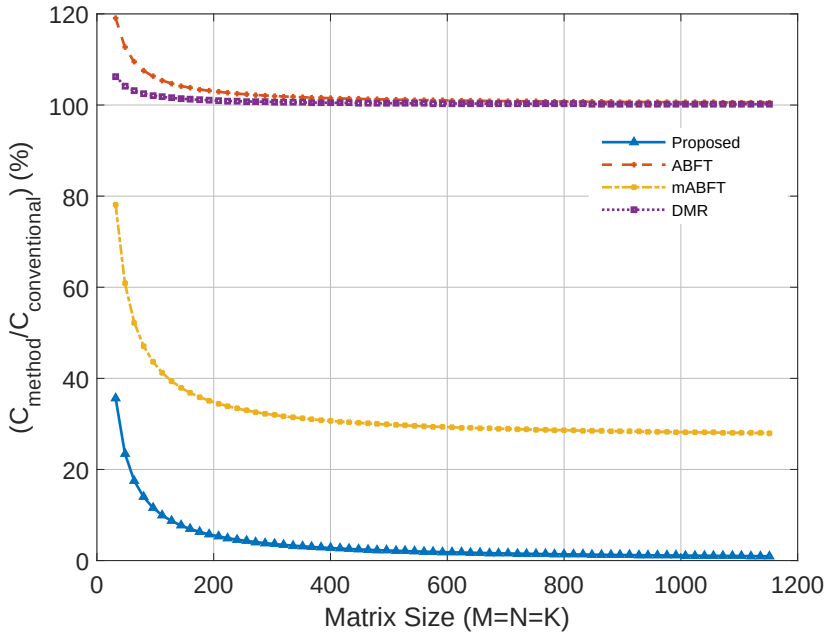
Assuming no SDCs occur within output GEMM results, each packed output $\widehat{r_i}$ or $\widehat{r_j}$ requires 11 arithmetic operations (including comparison operations) for unpacking [as elaborated in equations (2.15) – (2.22)] and an additional subtract and compare operation for error check (cf. Section 2.2.2). Therefore, $\frac{11M^2}{2}$ operations are required for the unpacking of the two $\frac{M}{2} \times \frac{M}{2}$ packed outputs, $\widehat{\mathbf{R}}_i$ and $\widehat{\mathbf{R}}_j$, while M^2 operations are required for error checking. The final Row-Column ABFT validation utilized by the proposed approach for the detection of certain pathological SDC cases would require $(2M^2 - 2M)$ and $M^2 + 2M - 1$ operations for preprocessing and error check respectively. In summary, assuming no SDCs occur, $\frac{1}{2}(2M^3 + 15M^2)$ double bitwidth and $(3M^2 - 1)$ single bitwidth arithmetic operations are required. By doubling the number of operations computed using the double bitwidth number representation, the number of operations is given by:

$$\mathrm{C}_{\{\text{no error in proposed}\}} = 2M^3 + 18M^2 - 1 \tag{2.26}$$

To achieve same protection for GEMM outputs using traditional ABFT, the row-column checksum would require $2M^2 - 2M$ addition operations for it's computation, while the GEMM itself will be computed with $(2M^3 + 3M^2 - 1)$ operations and the ABFT row & column checksum will be validated with $2M^2 + 2M$ addition and comparisons. Therefore, assuming



(a)



(b)

Figure 2.5: Theoretical percentile complexity of numerical packing, ABFT, mABFT and DMR for error detection and correction in comparison to conventional fault intolerant GEMM for: (a) no detected error and (b) One-row error as obtained using KULFI [3] for fault injection.

no SDCs occurred, the operations of traditional ABFT are given by:

$$\mathrm{C}_{\{\text{no error in ABFT}\}} = 2M^3 + 7M^2 - 1 \tag{2.27}$$

For mABFT, given that two checksum rows (and columns) are generated per partition size of 16, each unweighted checksum element would require 15 add operations for it's computation while the corresponding weighted checksum element is computed with 31 arithmetic operations. Therefore, $46M$ operations are required for the checksum computation of each partition, and $\frac{M}{16}$ and $\frac{M}{16}$ partitions exist the input matrices **A** and **B** respectively. In total, pre-processing for mABFT would require $46M \cdot \frac{M}{16} \cdot 2 = \frac{23M^2}{4}$ arithmetic operations for the calculation of the row & column checksums. Performing GEMM between the $\frac{9M}{8} \times M$ and $M \times \frac{9M}{8}$ matrices of mABFT will subsequently require $\frac{81}{64}\left(2M^3 - M^2\right)$ operations and, in order to check for SDCs within each 18×18 partition of the resulting $\frac{9M}{8} \times \frac{9M}{8}$ output matrix, 16 (or 32) operations will be required for each unweighted row/-column checksum error check (or each weighted row/column error check). This implies that mABFT requires $48 \cdot 2 \cdot \left(\frac{81M^2}{64}\right) \cdot \left(\frac{1}{18}\right) = \frac{27M^2}{4}$ operations for error check. Therefore, the overall complexity of mABFT (assuming no SDCs occurred) is:

$$\mathrm{C}_{\{\text{no error in mABFT}\}} = \frac{1}{64}\left(162M^2 + 719M^2\right) \tag{2.28}$$

Finally, dual modular redundancy does not require any pre or post processing for its computation. However, $4M^3 - 2M^2$ and M^2 arithmetic operations are performed for the computation of GEMM and error checking,

respectively. Therefore, assuming no SDCs occurred, the complexity of DMR is:

$$\mathrm{C}_{\{\text{no error in DMR}\}} = 4M^2 - M^2 \tag{2.29}$$

Combining (2.26)–(2.29) leads to the ratios reported in Proposition 1.

By comparing the derived complexities of (2.26)–(2.29) with the conventional fault-intolerant GEMM, we show in Fig. 2.5(a), the expected overhead for incorporating fault tolerance within GEMM computations. Specifically, we show that for the dominant case of no SDCs in practical applications, numerical packing and ABFT is expected to incur less than 1% additional for error checking. Furthermore, the analysis shows that on the average, numerical packing is about 2% less efficient than ABFT, while offering over 20% increased performance in comparison to mABFT and DMR.

Proposition 2.2. *Given x SDCs in output GEMM results, the number of arithmetic operations of the proposed method for the GEMM computation followed by error detection and correction, and averaged between the best and worst case of SDC patterns, is:*

(i): $\frac{M^3+M^2\left(2\sqrt{x}+26.5\right)+M\left(\sqrt{x}-6x\right)-30x-\sqrt{x}}{3M^3+M^2\left(2\sqrt{x}+8.5\right)+M\sqrt{x}-\sqrt{x}-1.5} \cdot 100\%$ *less than that of ABFT;*

(ii): $\frac{34M^3-433M^2-366Mx+1912x+64}{162M^3+719M^2+18Mx-8x}) \cdot 100\%$ *less than that of mABFT;*

(iii): $\frac{2M^3-19M^2-3Mx-31.5x+1}{4M^3-M^2+3xM-1.5x} \cdot 100\%$ *less than that of DMR, with all comparisons plotted in Fig. 2.5(b) for* $M \in [32, 1152]$*, and* $x = M$.

Proof. Given the detection of x SDCs within the output GEMM results, the number of computations required by all algorithms for error correction is dependent on their location. We therefore calculate the operations required for error correction as the average of the "worst-case" and "best-case" SDC distribution in output GEMM results, where "worst-case" refers to an SDC distribution requiring the highest number of operations and "best-case" SDC distribution requires the least number of operations.

For numerical packing, the "best-case" refers to the case of detection of an SDC in an output r_i and its corresponding output r_j , i.e, $i = j$. In such a case, our method of error correction re-computes $\frac{x}{2}$ locations for each of $\widehat{\mathbf{R}}_i$ and $\widehat{\mathbf{R}}_j$, thereby requiring $x\,(2M-1)$ operations for all erroneous location re-computations, while $2 \cdot 11 \cdot \frac{x}{2}$ operations would be required for the unpacking of all x locations in both matrices (as each unpacking requires 11 operations).

On the other hand, the "worst-case" SDC distribution occurs when all x SDCs happen within either of $\widehat{\mathbf{R}}_i$ or $\widehat{\mathbf{R}}_j$. Given that any SDC detected for each r_i or r_j requires the re-computation of both $\widehat{r}_i$ and $\widehat{r}_j$ of the failed location, $2x$ re-computations and unpackings will be required for this scenario, i.e., $x\,(4M+20)$ arithmetic operations.

By averaging between the two cases, the proposed approach requires $\frac{x}{2}\,(6M+21)$ operations to correct x detected SDCs. By doubling this result:

$$\mathrm{C}_{\{x \text{ errors in proposed}\}} = 2M^3 + 18M^2 - 1 + 6Mx + 30x \tag{2.30}$$

Concerning ABFT, assuming (for simplicity of exposition) that $\sqrt{x}$ is integer, the "best-case" SDC distribution for ABFT is having the SDCs located within a $\sqrt{x} \times \sqrt{x}$ square in the matrix. Therefore, provided the SDCs are all detectable, ABFT would flag $\sqrt{x}$ rows and columns as erroneous. When multiple SDCs are detected via ABFT, entire rows and columns that fail the checksum test are recomputed. The "best-case" SDC distribution of ABFT would therefore require a re-computation of $\sqrt{x}$ rows and $\sqrt{x}$ columns involving $2\sqrt{x}\,(2M^2 + M - 1)$ MAC operations.

On the other hand, if x is large (e.g., $x \geq 16$), entire GEMM re-computation is more beneficial in practice in comparison to selective recomputation of multiple rows and columns. This is because the data access irregularity, in conjunction with the high percentage of outputs that are recomputed, ends up incurring higher execution time penalty in comparison to simply recomputing the GEMM. Therefore, $2M^3 + 3M^2 - 1$ MAC operations is required for this case.

By taking the average of these two cases and adding them to the complexity of GEMM and error detection previously derived, we obtain:

$$\begin{aligned} \mathrm{C}_{\{x \text{ errors in ABFT}\}} &= 2M^3 + 7M^2 + 0.5 \cdot \left[2M^3 + 3M^2 - 1\right] \\ &\quad + \left[\sqrt{x}\left(2M^2 + M - 1\right)\right] - 1 \\ &= 3M^3 + M^2\left(2\sqrt{x} + 8.5\right) + M\sqrt{x} - \sqrt{x} - 1.5 \end{aligned} \tag{2.31}$$

The mABFT implementation presented in this work can detect two SDCs and correct one SDC for each row or column within a partition. The "best-case" SDC distribution for this implementation would be having x SDCs

spread in such a way that only one SDC occurs in a row or column of a partition. Since mABFT requires one division operation for error location and one subtraction operation for error correction [5, 103] , $2x$ operations would be required in order to correct the x detected SDCs.

On the other hand, the "worst-case" error location in mABFT occurs when 4 SDCs occur within a 2×2 sub-block in a partition such that two SDCs are detected in each of the two rows and columns corresponding to the sub-block error location. Since the mABFT implementation in this work can not correct more than one SDC per row/column, entire erroneous rows and columns of the affected partitions are re-computed. Thus, $4 \cdot 18\,(2M - 1)$ arithmetic operations are required for the re-computation of 2 erroneous rows and 2 erroneous columns assumed to be spread across $\frac{x}{4}$ different locations. Therefore, the overall complexity of mABFT including the correction of x SDCs is:

$$\mathrm{C}_{\{x \text{ errors in mABFT}\}} = \frac{1}{64}\left(162M^3 + 719M^2\right) + x\,(18M - 8) \tag{2.32}$$

Finally, performing same analysis for DMR, the "best-case" SDC distribution would refer to two SDCs occurring exactly at corresponding positions of the two GEMMs. This would require re-computation of $\frac{x}{2}$ output locations. On the other hand, the "worst-case" SDC here refers to all x SDCs occurring at different locations of one of the GEMM outputs. In such a case, x locations would need to be re-computed for each of the output matrices. Therefore, the overall complexity of DMR (including the correction

of x SDCs) is:

$$\mathrm{C}_{\{x \text{ errors in DMR}\}} = \quad 4M^3 - M^2 + 3xM - 1.5x \tag{2.33}$$

Combining the last four equations leads to the ratios reported in Proposition 2.

In Fig. 2.5(b), we show how these complexities scale with increasing matrix size in comparison to fault-intolerant GEMM computation. We consider the case of $x = M$, which is the single row/column error encountered when for example, memory bit flip(s) manifests as data error within an input matrix element. Because this erroneous value is used for GEMM computation, an entire row/column of matrix outputs is returned erroneous as will be elaborated further in Section 2.4.1. The theoretical exposition of (2.30)–(2.33) and the plot of Fig. 2.5(b) highlights the performance improvement offered by numerical packing in comparison to all existing methods of fault tolerance in GEMM, especially for the case of multiple error detection and correction. On the average, more than 25% performance improvement is offered by numerical packing in comparison to ABFT, mABFT and DMR for SDC mitigation. Overall, the complexity analysis of Propositions 1 and 2 indicates that our proposal is expected to incur small penalty when no SDCs happen, and become highly-beneficial when multiple SDCs occur in GEMM.

2.3.3 Reliability Aspects

In summary, the presented method utilizes the notion of packing to create two complementary and compacted input descriptions for $\mathbf{A}$ (i.e. $\widehat{\mathbf{A}}_i$ and $\widehat{\mathbf{A}}_j$) that are then used within two independent subblock GEMM calls (64-bit integer or floating-point) with a compacted description for $\mathbf{B}$ (i.e. $\widehat{\mathbf{B}}$). This results in two $\frac{M}{2} \times \frac{M}{2}$ independently-computable subblock products that contain all necessary results, albeit in packed and entangled form. These can be unpacked and the entangled parts of one subblock product can be matched with the extracted outputs of the other subblock product by post-entangling the latter. If mismatches are detected, this means a part of a matrix product executed erroneously. If no mismatches are detected (or, otherwise, once all mismatched outputs are recomputed), the results are then cross-checked with the row-column checksum of (2.23). We can thus make the following observations:

- Two quarter-size GEMMs of double bitwidth are used instead of one full-size GEMM for the subblock product. Thus, the storage requirements remain the same, with the only increase stemming from the single additional checksum element of (2.23).

- While the proposed approach (approximately) halves the overall MAC operations against the conventional approach, all operations are performed in 64-bit instead of 32-bit representations. If 64-bit MAC operations have double the cycle count in comparison to 32-bit MAC, the overall execution time for the final results is expected to remain comparable between the proposed approach and the conventional,

fault-intolerant, GEMM. Specifically, while conventional GEMM performs $(M^2 \times M)$ MAC operations for GEMM computation, the proposed approach performs $2\left(\frac{M^2}{2} \times \frac{M}{2}\right)$ operations.

- The process of error checking via the post-entanglement check and the ABFT row-column checksum of (2.23) requires only $M\,(M+1)$ MAC operations.

In order to establish results characterizing the reliability of the proposed approach in comparison to ABFT, partitioned and weighted mABFT and DMR, we define the following error model concerning the results of an individual GEMM operation. For simplicity of exposition, our analysis is focusing on the case of a square subblock matrix product comprising $M \times M$ outputs; it is straightforward to extend it to non-square subblock sizes.

Definition 2.1. *Operation* **R** = **AB***, with* **R** *the* $M \times M$ *integer matrix of the result and* **A** *and* **B** *matrices comprising integer inputs, is considered to be affected by independent uniformly distributed (IUD) errors if all outputs,* $r_{m,n}$ *(*$0 \leq m, n < M$*), are susceptible to IUD bit flips with a certain probability.*

The following propositions derive the reliability of each approach under the IUD error model of Definition (2.1). The focus is mainly on GEMM computation using integer libraries where the effects of IUD errors are easier to quantify.

Proposition 2.3. *Under the IUD error model of Definition* (2.1)*, the percentage of errors detectable by the proposed approach under two 64-bit*

packed subblock products is lower-bounded by

$$\text{Re}\{\text{proposed}\} > \left(1 - \frac{2.77 \cdot 10^{-32}\left(M^2-4\right)}{M^6 - 6M^4 + 11M^2 - 6}\right) \times 100\% \tag{2.34}$$

Proof. The proposed method will not be able to detect errors when *all* of the following conditions are satisfied for two distinct locations, $(\widehat{m}_1, \widehat{n}_1)$ and $(\widehat{m}_2, \widehat{n}_2)$, within the packed output matrices $\widehat{r}_i$ and $\widehat{r}_j$, with $(\widehat{m}_1, \widehat{n}_1) \neq (\widehat{m}_2, \widehat{n}_2)$ and: $0 \leq \widehat{m}_1, \widehat{n}_1 < \lfloor \frac{M}{2} \rfloor$ and $0 \leq \widehat{m}_2, \widehat{n}_2 < \lfloor \frac{M}{2} \rfloor$.

1. Either both $\widetilde{r}_{2\widehat{m}_1,2\widehat{n}_1}$ and $\widetilde{r}_{2\widehat{m}_1+1,2\widehat{n}_1+1}$, or both $\widetilde{r}_{2\widehat{m}_1+1,2\widehat{n}_1}$ and $\widetilde{r}_{2\widehat{m}_1,2\widehat{n}_1+1}$ are corrupted within one packed representation ($\widehat{r}_i$ and $\widehat{r}_j$, respectively) and also another pair, $\widetilde{r}_{2\widehat{m}_2,2\widehat{n}_2}$ and $\widetilde{r}_{2\widehat{m}_2+1,2\widehat{n}_2+1}$ (or $\widetilde{r}_{2\widehat{m}_2+1,2\widehat{n}_2}$ and $\widetilde{r}_{2\widehat{m}_2,2\widehat{n}_2+1}$), are corrupted within same packed or another packed representation ($\widehat{r}_i$ and $\widehat{r}_j$, respectively).

2. Their subtraction still agrees with the entangled result extracted from the other GEMM (i.e. $\tilde{\varepsilon}_{(2\widehat{m}_1,2\widehat{n}_1+1),(2\widehat{m}_1+1,2\widehat{n}_1)}$ and/or $\tilde{\varepsilon}_{(2\widehat{m}_2,2\widehat{n}_2+1),(2\widehat{m}_2+1,2\widehat{n}_2)}$ from $\widehat{r}_i$, respectively, and $\tilde{\varepsilon}_{(2\widehat{m}_1+1,2\widehat{n}_1+1),(2\widehat{m}_1,2\widehat{n}_1)}$ and/or $\tilde{\varepsilon}_{(2\widehat{m}_2+1,2\widehat{n}_2+1),(2\widehat{m}_2,2\widehat{n}_2)}$ from $\widehat{r}_j$, respectively).

3. The errors are complementary, such that sum of all elements of the result matrix after unpacking matches the row-column ABFT checksum value of (2.23).

We can establish the total number of such cases as the number of distinct results produced by ($\forall \varepsilon \in \{\pm 1, \ldots, \pm 2^{k-1}\}$) and shown here for the case when corruption occurs in two different sections of two different packed

representations:

$$\begin{aligned}
\widetilde{r}_{2\widehat{m}_1+1,2\widehat{n}_1+1} &\leftarrow \widetilde{r}_{2\widehat{m}_1+1,2\widehat{n}_1+1} + \varepsilon \\
\widetilde{r}_{2\widehat{m}_1,2\widehat{n}_1} &\leftarrow \widetilde{r}_{2\widehat{m}_1,2\widehat{n}_1} + \varepsilon \\
\widetilde{r}_{2\widehat{m}_2+1,2\widehat{n}_2} &\leftarrow \widetilde{r}_{2\widehat{m}_2+1,2\widehat{n}_2} - \varepsilon \\
\widetilde{r}_{2\widehat{m}_2,2\widehat{n}_2+1} &\leftarrow \widetilde{r}_{2\widehat{m}_2,2\widehat{n}_2+1} - \varepsilon
\end{aligned} \tag{2.35}$$

where ε is the error injected in a complementary manner within $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}+1}$ and $\widetilde{r}_{2\widehat{m},2\widehat{n}}$ (and within $\widetilde{r}_{2\widehat{m},2\widehat{n}+1}$ and $\widetilde{r}_{2\widehat{m}+1,2\widehat{n}}$) and $k = 20$ for integer representation. In all the cases of (2.35), the outputs are erroneous – yet remain undetected.

The error, ε can take up to 2^{k-1} values out of the $\binom{2^{32}}{4}$ possible error values at the four output GEMM locations. In addition, there are $\binom{\frac{M^2}{4}}{2}$ possible ways of choosing the indices $(\widehat{m}_1, \widehat{n}_1), (\widehat{m}_2, \widehat{n}_2)$ in order that errors will be undetected as opposed to a total of $\binom{M^2}{4}$ possible ways of choosing any four locations in output GEMM results. Therefore, the probability of undetectable errors in the proposed method of numerical packing under the error model of Definition (2.1) is:

$$\begin{aligned}
&\Pr\{\text{four undetected faults in proposed}\} = \\
&\frac{2^{k-1}}{\binom{2^{32}}{4}} \times \left(\frac{\text{\# of undetect. comb. of 4 faults}}{\text{\# of possible comb. of 4 faults}} \right) < \\
&2^{-104} \cdot \left(\frac{\binom{\frac{M^2}{4}}{2}}{\binom{M^2}{4}} \right) < \frac{2.77 \cdot 10^{-32} \left(M^2 - 4\right)}{M^6 - 6M^4 + 11M^2 - 6}
\end{aligned} \tag{2.36}$$

with $k = 20$. This means that, under Definition (2.1), the proposed fault-tolerant GEMM is expected to successfully detect the percentage of errors

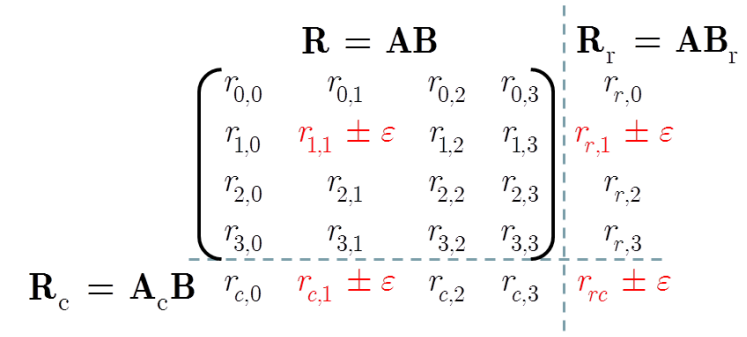
lower-bounded by (2.34).

Proposition (2.3) did not consider the case when more than four undetected errors occur, as the likelihood of such cases is negligible when compared to (2.34). Proposition (2.3) shows that the proposed case achieves at least "48 nines" reliability, i.e. less than 1 in 10^{48} errors in the results would remain undetected (with each error comprising one or more bit flips).

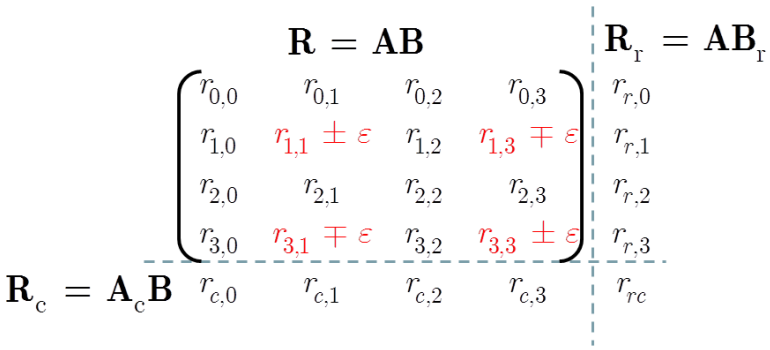
Proposition 2.4. *Under the IUD error model of Definition* (2.1)*, the percentage of errors detectable by a 32-bit* $(M \times M)$ *traditional ABFT-based GEMM is lower-bounded by*

$$Re\{ABFT\} > \left(1 - \frac{2.42 \cdot 10^{-27} M}{(M^2 + 2M - 2) \cdot (M^2 + 2M - 1) \cdot (M + 2)}\right) \times 100\% \quad (2.37)$$

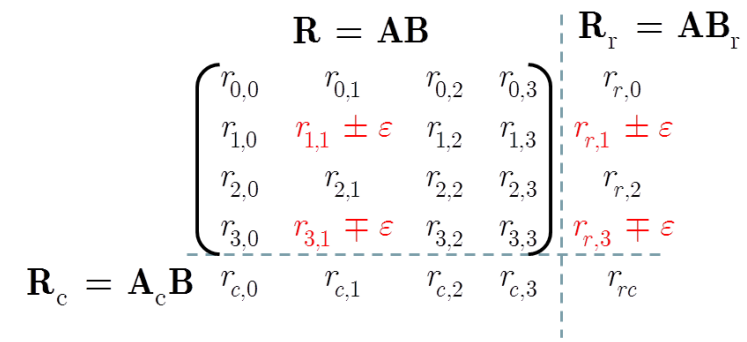
Proof. Traditional ABFT will detect all errors that do not occur in locations with coinciding row or column indices in $\mathbf{R}$. Errors occurring in four locations with coinciding row and column indices [including the column and row checksums of the subblock, i.e., $\mathbf{r}_c$, $\mathbf{r}_r$ in Fig. 1.2] will remain undetectable when IUD bitflips result in the erroneous addition of a value ε in such a way that coinciding rows (also columns) lead to error cancellation during the SDC detection stage of ABFT. Fig. 2.6 shows such error patterns including the expected sign of ε. For example, in Fig. 2.6(a), out of $\binom{2^{32}}{4}$ combinations of SDC values (ε) at the four undetectable locations, ε can only take $2^{32} - 1$ possible values. Thus, the total probability of



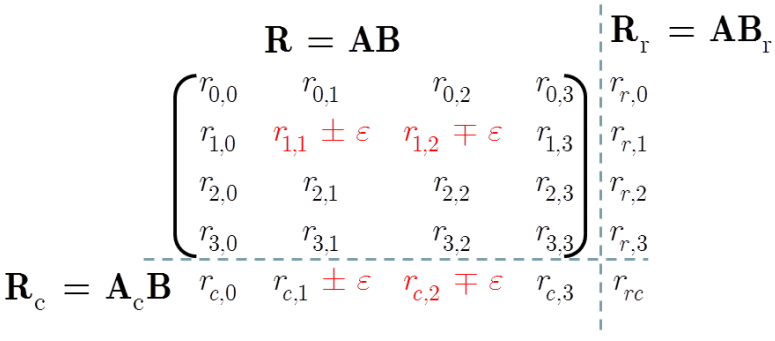
(a)



(b)



(c)



(d)

Figure 2.6: Simplest error distribution patterns undetectable by the traditional ABFT for a 4×4 matrix output, $\mathbf{R} = \mathbf{AB}$

undetectable errors is:

$$
\begin{aligned}
&\Pr\{\text{four coinciding faults in ABFT}\} = \\
&\frac{2^{32}-1}{\binom{2^{32}}{4}} \times \left(\frac{\text{\# of undetect. comb. of 4 faults}}{\text{\# of possible comb. of 4 faults}}\right)_{\text{pattern(a)}} + \\
&\frac{2^{32}-2}{\binom{2^{32}}{4}} \times \left(\frac{\text{\# of undetect. comb. of 4 faults}}{\text{\# of possible comb. of 4 faults}}\right)_{\text{pattern(b)}} + \\
&\frac{2^{32}-2}{\binom{2^{32}}{4}} \times \left(\frac{\text{\# of undetect. comb. of 4 faults}}{\text{\# of possible comb. of 4 faults}}\right)_{\text{pattern(c)}} + \\
&\frac{2^{32}-2}{\binom{2^{32}}{4}} \times \left(\frac{\text{\# of undetect. comb. of 4 faults}}{\text{\# of possible comb. of 4 faults}}\right)_{\text{pattern(d)}} < \\
&2^{-91} \cdot \left(\frac{\binom{M}{1}\cdot\binom{M}{1}}{\binom{(M+1)^2}{4}} + \frac{\binom{M}{2}\cdot\binom{M}{2}}{\binom{(M+1)^2}{4}} + \frac{\binom{M}{2}\cdot\binom{M}{1}}{\binom{(M+1)^2}{4}} + \frac{\binom{M}{1}\cdot\binom{M}{2}}{\binom{(M+1)^2}{4}} \right) < \\
&\qquad\qquad \frac{2.42 \cdot 10^{-27} M}{(M^2+2M-2)\cdot(M^2+2M-1)\cdot(M+2)}
\end{aligned}
\tag{2.38}
$$

In (2.38), the number of undetectable combinations of four errors is derived by taking the sum of the individual probabilities of the error patterns shown in Fig. 2.6. Again, considering considering in Fig. 2.6(a), any one row out of the M rows of the original matrix $\mathbf{R}$, can have $\pm\varepsilon$ at any column (out of the M columns of each row). For the error to be undetectable, the corresponding checksum row, checksum column and the full checksum location must be affected by IUD bitflips with resulting value equal to $\pm\varepsilon$. This means that, under Definition (2.1), a 32-bit ABFT-based GEMM is expected to successfully detect the percentage of errors lower-bounded by (2.38).

Proposition (2.4) did not consider the cases where more than four undetectable errors occur in ABFT, as the likelihood of such occurrences is negligible when compared to (2.37). As an example of the reliability

achieved by ABFT, by using $L = 576$, it is evident from Proposition 2 that ABFT-based GEMM achieves at least "38 nines" reliability, i.e. less than 1 in 10^{38} errors in the results would remain undetected (with each error comprising one or more bit flips).

Proposition 2.5. *Under the IUD error model of Definition* (2.1)*, the percentage of errors detectable by* a 32-bit $(M \times M)$ mABFT-based GEMM partitioned into $P \times P$ matrices (provided L is a multiple of P) with two checksums is lower-bounded by

$$Re\{mABFT\} > \left(1 - \frac{2.84 \cdot 10^{-67} M^2 (P+1)^2}{\prod_{i=0}^{8} [(P+2)^2 - i]}\right) \times 100\% \tag{2.39}$$

Proof. The first checksum of each column and row of the input $P \times P$ blocks of mABFT will detect all errors within each partition as long as they do not occur in locations with coinciding row or column indices such that the errors cancel out as described in the proof of proposition 2.4. Moreover, the second checksum of each column and row, $\mathbf{r}_{c2}$ and $\mathbf{r}_{r2}$, will detect all such errors as long as they do not occur in *both* the weighted checksums and the row-column weighted checksum (r_{r2c2}) in the patterns shown in Fig. 2.7. Therefore,the total probability of undetectable errors when using the combined partitioned and weighted mABFT approaches

$$\mathbf{R} = \mathbf{AB} \qquad \mathbf{R}_r = \mathbf{AB}_r$$

$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} \\ r_{1,0} & r_{1,1} \pm \varepsilon & r_{1,2} & r_{1,3} \\ r_{2,0} & r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,0} & r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix} \quad \begin{matrix} r_{r1,0} & r_{r2,0} \\ r_{r1,1} \pm \varepsilon & r_{r2,0} \pm 2\varepsilon \\ r_{r1,2} & r_{r2,0} \\ r_{r1,3} & r_{r2,0} \end{matrix}$$

$$\mathbf{R}_c = \mathbf{A}_c\mathbf{B} \qquad \begin{matrix} r_{c1,0} & r_{c1,1} \pm \varepsilon & r_{c1,2} & r_{c1,3} \\ r_{c2,0} & r_{c2,1} \pm 2\varepsilon & r_{c2,2} & r_{c2,3} \end{matrix} \quad \begin{matrix} r_{r1c1} \pm \varepsilon & r_{r2c1,0} \pm 2\varepsilon \\ r_{r1c2} \pm 2\varepsilon & r_{r2c2} \pm 4\varepsilon \end{matrix}$$

(a)

$$\mathbf{R} = \mathbf{AB} \qquad \mathbf{R}_r = \mathbf{AB}_r$$

$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} \\ r_{1,0} & r_{1,1} \pm \varepsilon & r_{1,2} \mp \varepsilon & r_{1,3} \\ r_{2,0} & r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,0} & r_{3,1} \mp \varepsilon & r_{3,2} \pm \varepsilon & r_{3,3} \end{pmatrix} \quad \begin{matrix} r_{r1,0} & r_{r2,0} \\ r_{r1,1} & r_{r2,0} \mp \varepsilon \\ r_{r1,2} & r_{r2,0} \\ r_{r1,3} & r_{r2,0} \pm \varepsilon \end{matrix}$$

$$\mathbf{R}_c = \mathbf{A}_c\mathbf{B} \qquad \begin{matrix} r_{c1,0} & r_{c1,1} & r_{c1,2} & r_{c1,3} \\ r_{c2,0} & r_{c2,1} \mp 2\varepsilon & r_{c2,2} \pm 2\varepsilon & r_{c2,3} \end{matrix} \quad \begin{matrix} r_{r1c1} & r_{r2c1,0} \\ r_{r1c2} & r_{r2c2} \pm 2\varepsilon \end{matrix}$$

(b)

$$\mathbf{R} = \mathbf{AB} \qquad \mathbf{R}_r = \mathbf{AB}_r$$

$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} \\ r_{1,0} & r_{1,1} \pm \varepsilon & r_{1,2} & r_{1,3} \\ r_{2,0} & r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,0} & r_{3,1} \mp \varepsilon & r_{3,2} & r_{3,3} \end{pmatrix} \quad \begin{matrix} r_{r1,0} & r_{r2,0} \\ r_{r1,1} \pm \varepsilon & r_{r2,0} \pm 2\varepsilon \\ r_{r1,2} & r_{r2,0} \\ r_{r1,3} \mp \varepsilon & r_{r2,0} \mp 2\varepsilon \end{matrix}$$

$$\mathbf{R}_c = \mathbf{A}_c\mathbf{B} \qquad \begin{matrix} r_{c1,0} & r_{c1,1} & r_{c1,2} & r_{c1,3} \\ r_{c2,0} & r_{c2,1} \mp 2\varepsilon & r_{c2,2} & r_{c2,3} \end{matrix} \quad \begin{matrix} r_{r1c1} & r_{r2c1,0} \\ r_{r1c2} \mp 2\varepsilon & r_{r2c2} \mp 4\varepsilon \end{matrix}$$

(c)

$$\mathbf{R} = \mathbf{AB} \qquad \mathbf{R}_r = \mathbf{AB}_r$$

$$\begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} \\ r_{1,0} & r_{1,1} \pm \varepsilon & r_{1,2} \mp \varepsilon & r_{1,3} \\ r_{2,0} & r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,0} & r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix} \quad \begin{matrix} r_{r1,0} & r_{r2,0} \\ r_{r1,1} & r_{r2,0} \mp \varepsilon \\ r_{r1,2} & r_{r2,0} \\ r_{r1,3} & r_{r2,0} \end{matrix}$$

$$\mathbf{R}_c = \mathbf{A}_c\mathbf{B} \qquad \begin{matrix} r_{c1,0} & r_{c1,1} \pm \varepsilon & r_{c1,2} \mp \varepsilon & r_{c1,3} \\ r_{c2,0} & r_{c2,1} \pm 2\varepsilon & r_{c2,2} \mp 2\varepsilon & r_{c2,3} \end{matrix} \quad \begin{matrix} r_{r1c1} & r_{r2c1,0} \mp \varepsilon \\ r_{r1c2} & r_{r2c2} \mp 2\varepsilon \end{matrix}$$

(d)

Figure 2.7: Simplest error distribution patterns undetectable by the partition and weighted mABFT approach of [4–8] using a partition size, $P = 4$

of [4–8] is:

$$\begin{aligned}
&\Pr\{\text{nine coinciding faults}\} = \#\text{ of partitions}\times \\
&\left[\frac{2^{32}-1}{\binom{2^{32}}{9}}\times\left(\frac{\#\text{ of undetect. comb. of 9 faults per partition}}{\#\text{ of possible comb. of 9 faults per partition}}\right)_{\text{pattern(a)}}+\right. \\
&\frac{2^{32}-2}{\binom{2^{32}}{9}}\times\left(\frac{\#\text{ of undetect. comb. of 9 faults per partition}}{\#\text{ of possible comb. of 9 faults per partition}}\right)_{\text{pattern(b)}}+ \\
&\frac{2^{32}-2}{\binom{2^{32}}{9}}\times\left(\frac{\#\text{ of undetect. comb. of 9 faults per partition}}{\#\text{ of possible comb. of 9 faults per partition}}\right)_{\text{pattern(c)}}+ \\
&\left.\frac{2^{32}-2}{\binom{2^{32}}{9}}\times\left(\frac{\#\text{ of undetect. comb. of 9 faults per partition}}{\#\text{ of possible comb. of 9 faults per partition}}\right)_{\text{pattern(d)}}\right]< \\
&2^{-238}\cdot\left(\frac{\binom{P}{1}\cdot\binom{P}{1}}{\binom{(P+2)^2}{9}}+\frac{\binom{P}{2}\cdot\binom{P}{2}}{\binom{(P+2)^2}{9}}+\frac{\binom{P}{2}\cdot\binom{P}{1}}{\binom{(P+2)^2}{9}}+\frac{\binom{P}{1}\cdot\binom{P}{2}}{\binom{(P+2)^2}{9}}\right)\cdot\frac{M^2}{P^2}< \\
&\frac{2.84\cdot10^{-67}M^2\left(P+1\right)^2}{\prod_{i=0}^{8}\left[\left(P+2\right)^2-i\right]}
\end{aligned} \tag{2.40}$$

This means that, under Definition (2.1), a 32-bit mABFT-based GEMM is expected to successfully detect the percentage of errors lower-bounded by (2.39).

Proposition (2.5) did not consider the cases where more than nine undetectable errors occur per partition in the described ABFT, as the likelihood of such occurrences is negligible when compared to (2.39). As an example of the reliability, by using $M = 576$ and $P = 16$, it is evident from Proposition (2.5) that partitioned ABFT-based GEMM with two checksum vectors achieves at least "79 nines" reliability, i.e. less than 1 in 10^{79} errors in the results would remain undetected (with each error comprising one or more bit flips).

Proposition 2.6. *Under the IUD error model of Definition* (2.1)*, the per-*

centage of errors detectable by a 32-bit $(M \times M)$ *DMR-based GEMM is lower-bounded by*

$$\text{Re}\{\text{DMR}\} > \left(1 - 4.66 \cdot 10^{-10} M^{-2}\right) \times 100\% \tag{2.41}$$

Proof. DMR-based GEMM will detect all errors that do not occur in coinciding locations within the results of the two $M \times M$ identical GEMMs. When two errors δ and ε occur at coinciding locations of the two GEMM outputs, out of $\binom{2^{32}}{2}$ possible combinations of δ and ε, only $2^{32}-1$ conditions will result in $\delta = \varepsilon$. Thus, the probability of errors occurring at coinciding locations in both 32-bit GEMM calls (i.e undetectable errors) is:

$$\begin{aligned}&\Pr\{\text{two coinciding faults in DMR}\} = \\ &\frac{2^{32}-1}{\binom{2^{32}}{2}} \times \frac{\text{\# of undect. pairs of faults}}{\text{\# of possible pairs of faults}} < 2^{-31} \frac{M^2}{M^2 \times M^2} < 4.66 \cdot 10^{-10} M^{-2}\end{aligned} \tag{2.42}$$

This means that, under Definition (2.1), the percentage of errors detectable by an $M \times M$ 32-bit DMR-based GEMM is lower-bounded by (2.42). □

We remark that Proposition (2.6) did not consider the cases where more than two coinciding errors occur in DMR, as the likelihood of such occurrences is negligible when compared to (2.41). As an example of the reliability achieved by DMR, when $M = 576$ it is straightforward to derive from Proposition 1 that DMR-based GEMM achieves more than “15

nines" reliability, i.e. less than 1 in 10^{15} IUD errors in the results would remain undetected (with each error comprising one or more bit flips).

Propositions (2.6) - (2.3) demonstrate that all methods are extremely reliable. DMR detects errors in the results with at least 15 nines reliability, ABFT and the proposed method detect errors in the results with at least 38 nines reliability, while mABFT detects errors in the results with at least 79 nines reliability, but with considerable overhead. However, this additional reliability is not significant in practice: even under the assumption of a petaFLOP computer (i.e. assuming that 10^{15} results are produced per second) and under constant IUD error patterns occurring in all the outputs, it is expected it would take more than 10^{17} seconds for the occurrence of an undetectable error in ABFT and the proposed approach, i.e. more than 3.17 billion years. Instead, the biggest practical advantage of mABFT, DMR and the proposed approach is that, unlike ABFT, *the locations* of the vast majority of all detected errors are explicitly indicated, which allows for minimum effort in the recomputation of all failed results in order to ensure error-free operation.

2.4 Experimental Results

In this section, we present execution time results with an Intel i7- 3632QM 2.2GHz processor (Ivy Bridge architecture with AVX support, Ubuntu 14.04.1 LTS, Clang 3.2 compiler). The BLAS routine of the ATLAS math

library was used for all GEMM calls (sGEMM and dGEMM for 64-bit)[3]. After experimenting with various subblock sizes, we selected six subblock sizes as representative cases, ranging from 32×32 (below which the use of sGEMM or dGEMM is not beneficial) to 1152×1152. We opted for subblock sizes that are multiples of the subblock size settings for sGEMM and dGEMM within most open-source MKLs (e.g., ATLAS and GOTO) and avoid the use of cleanup code for the borders.

The ABFT approach follows Huang and Abraham [101], while we implemented the mABFT following Rexford and Jha [6], with the linear weighted checksum vector [4, 5] of (1.3). We opted for partition block size of 16×16 within each subblock in order to achieve output dynamic range comparable to the proposed approach and at the same time limit the overhead caused by the two additional checksum rows and columns per block[4]. For the ABFT implementation, due to the detection limitations analyzed in Section 2.3.1, entire rows and columns are recomputed when multiple SDCs are detected. In addition, when ten SDCs are detected in different rows and columns, the error detection process stops and entire GEMM subblock recomputation (i.e., complete rollback) is carried out, as this turns out to be less costly than complete detection and multi-row & column recomputation for the chosen subblock sizes.

[3]Since all optimized math kernel libraries today support only 32/64bit floating point number representations, we cast all integer inputs into floating point data-type after preprocessing while casting to integer data type in order to perform error-check and correction if required.

[4]Specifically, we found that, under $P = 8$, the overhead caused by mABFT is more than 150% in comparison to the fault-intolerant original GEMM and $P = 32$ or higher leads to unacceptable loss of dynamic range, as illustrated by (2.25).

2.4.1 SDC Injection Technique

In order to investigate the performance of all presented approaches for the detection and correction of soft errors, we perform an artificial fault injection campaign using the open source LLVM [132] based fault injection tool, KULFI [3]. KULFI emulates faults occurring in CPU state elements, which usually manifest as bit flips in a random computational state chosen at run time by injecting faults at the intermediate representation (IR) code of LLVM. Because the LLVM IR code preserves variable and function names and supports program analysis and transformations, it allows for controlled fault injections to be performed at specific program points, and into specific instructions. Given that such flips are somewhat rare in computing systems today, KULFI injects a single transient fault during every execution under investigation, albeit at a random time instance. By selecting fault injection parameters[5] as outlined in the author's example code [119], and by specifying the GEMM function name as the fault location, we observe and characterize the effect of the injected faults during GEMM computation into three classes:

- Benign errors: errors that have no effect on the GEMM output.
- Single SDC: errors that affect a data element that are not reused during the GEMM computation.
- Single-row SDC: errors that affect a data element that is used to

[5]Namely, the following parameters are used for all our fault injection executions: `iteration=100; byte_position=random; expected_fault_count=10; total_fault_count=100; inject_once flag=1; static_fault/dynamic_fault=1; inject_pointer_error=0; inject_data_error=1; print_fault_site=0`

compute an entire row/column of GEMM.

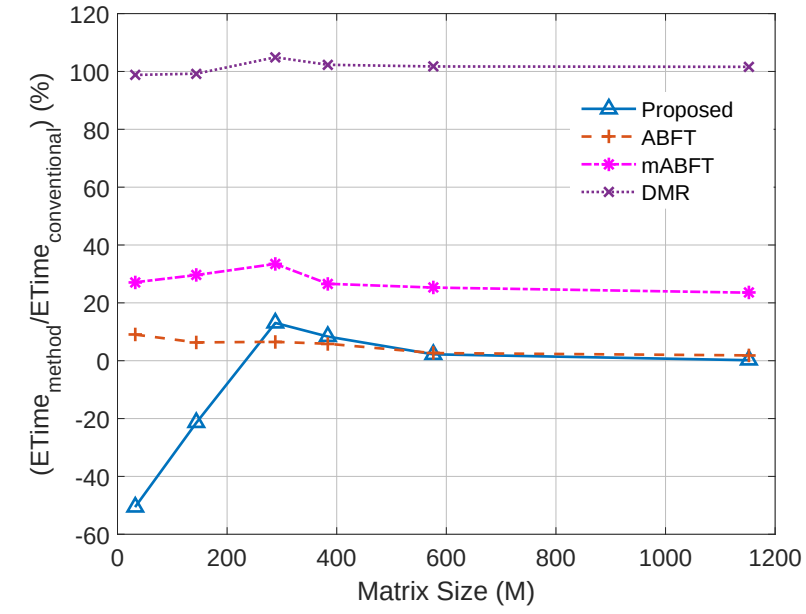
For example, a typical error summary when performing 100 fault injection executions (excluding segmentation and out-of-bound faults) of an $N \times N$-by-$N \times N$ GEMM is given by:

```
Single-row SDCs:16, Single SDC:34, Benign errors:50
```

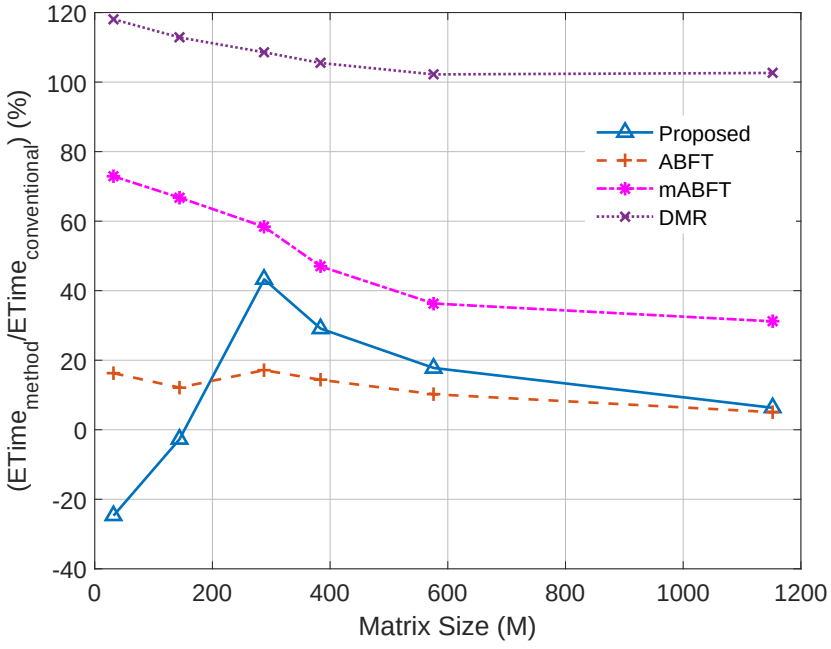
In order to keep track of the number of injected errors that manifest as SDCs, KULFI performs two executions of a given source code: error-free and error-prone executions. We design the error-free execution to implement optimized GEMM calls (using the `cblas_sgemm` and `cblas_dgemm` function of the ATLAS library), while the error-prone GEMM implementation is written in plain C code in such a way as to allow for error injection during GEMM. In this way, GEMM execution time overhead reported in Figure 2.8(a) is measured from the error-free implementation while the performance for pre- and post processing (including error correction) of Figs 2.8(b) and 2.9 are measured from the error-prone execution.

2.4.2 Results under SDCs

The plots of the percentage execution time overhead for all presented algorithms in comparison to the conventional fault intolerant single precision GEMM (sGEMM) computation is presented in Figures 2.8 and 2.9. The results were obtained by averaging the execution time of all methods

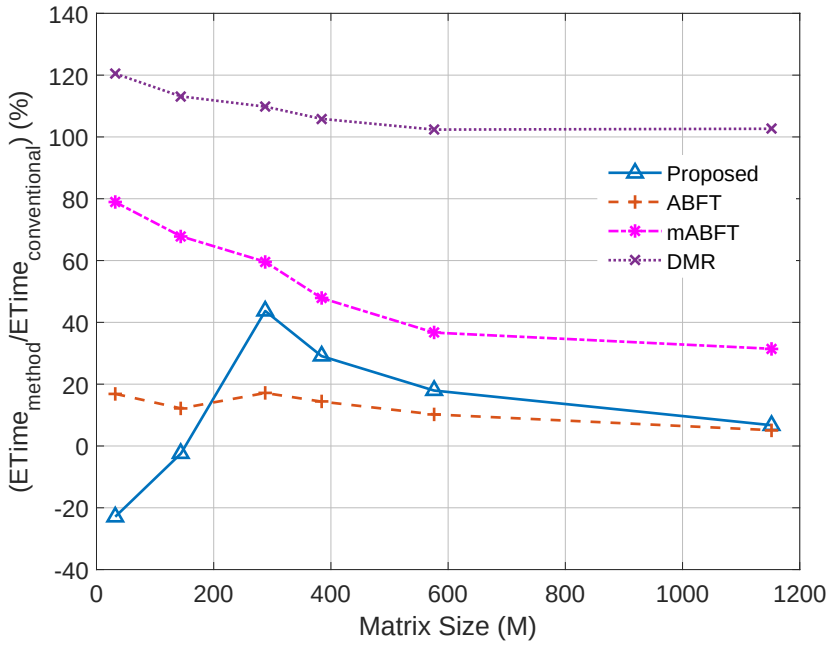


(a)

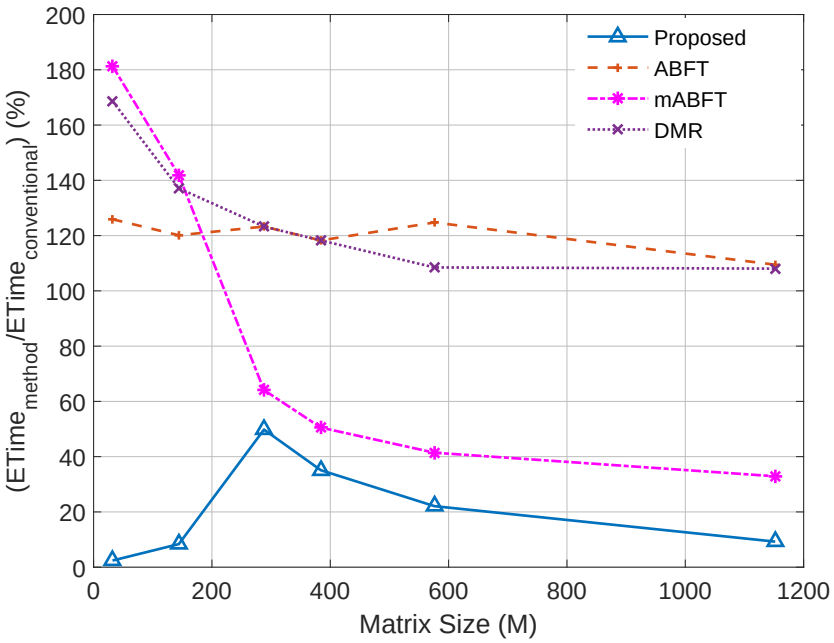


(b)

Figure 2.8: Execution time percentile overhead of numerical packing, ABFT, mABFT and DMR for (a) GEMM computation only; (b) pre-processing, GEMM computation, error check and post-processing in the absence of errors.



(a)



(b)

Figure 2.9: Execution time percentile overhead of the numerical packing, ABFT, mABFT and DMR for error detection and correction for: (a) single error injection and (b) One-row error injection using the KULFI fault injection.

after several independent runs (each using randomly-generated inputs). For all the reported experiments, all methods detected all incurred errors (i.e., the detection rate was 1.0 for all approaches). Therefore, the reported results compare the different methods with respect to execution time. To present a detailed analysis of the execution time performance of all methods, four subcases are shown: *(i)* only the cost of GEMM calls (i.e., not the cost of packing, unpacking, checksum generation or error checking) where the proposed approach is computed using double precision GEMM (dGEMM) and the other approaches utilize the sGEMM routine; *(ii)* everything when no SDCs occur; *(iii)* everything, including error checking and correction when one SDC occurs; *(iv)* everything, including the detection and correction of one row of SDCs injected using the KULFI tool.

Theoretically, the execution time for GEMM computation (without error tolerance) is expected to be equal for the conventional sGEMM and the two quatersize dGEMMs performed by the proposed approach (cf. proof of Section 2.3.2). In practice, due to the internal kernel optimizations of the ATLAS MKL for different matrix inner blocks, the results of Figure 2.8(a) show both positive and negative GEMM execution time overhead for the proposed approach, which subsequently affects the overall overhead for error tolerance for the presented matrix sizes. Overall, the results show that, against fault-intolerant (conventional) GEMM design, the proposed approach incurs execution time overhead between -24.70% and 43.20% when no SDCs occur and 2.41% and 49.94% when mitigating up to N SDCs in an $N \times N$ GEMM output. On average, 12.05% to 21.21%

overhead is incurred by the proposed scheme for tolerating up to N SDCs in GEMM for the presented matrix sizes. Similarly, the plots show that ABFT incurs an average execution time overhead of 12.64% to 120.34% for the same level of fault tolerance. Overall, it is evident from the obtained results that the proposed method incurs comparable overhead to ABFT when no errors occur and this overhead is (approximately) 18.04% and 46.37% less than that of mABFT and DMR-based GEMM.

We note that the GEMM execution time is controlled by the optimization offered by the utilized ATLAS MKL for different matrix sizes and data types. For example, while experimenting with smaller subblock sizes, we observed both positive and negative execution time overhead (i.e., speedup) for the two quarter-size 64-bit GEMMs computed by the proposed method in comparison to the conventional fault-intolerant 32-bit GEMM. Such behavior is well documented in the experimental results reported with such libraries, e.g., see the experiments with small GEMM sizes reported in [8, 36, 97]. In terms of the choice of subblock sizes used for our experiments, the proof of Section 2.3.2 illustrates that low-cost SDC correction techniques in matrix products (beyond the brute force method of modular redundancy) become very valuable as the matrix size increases, since, for small matrix sizes, SDCs can be efficiently mitigated by recomputation. From the results of Figs. 2.8(b) and 2.9 (cf. Table I of [131] for quantitative values), we see that, for the proposed method, mABFT and DMR, the ratio of pre- and post-processing to the actual GEMM computation decreases significantly as the matrix size increases. Thus, we focus our application section on a multimedia retrieval system with requirement for large integer GEMM computations as discussed in

subsequent sections. Finally, in terms of even smaller sizes (like 4×4 or 8×8 blocks), the use of GEMM is not justified there because typical implementations apply direct calculation of the results without requiring a high-performance library, also exploiting the potential symmetries that tend to exist in block transform matrices of that size (e.g., Hadamard or DCTs used in video coding, etc.). Therefore, such small block sizes are out of the scope of matrix products considered in our work.

The presented results in this section are in line with the theoretical predictions of Propositions 2.1 and 2.2 of Section 2.3.2 with the "best case" single SDC case and "worst-case" one-row SDC case shown in Figures 2.5(a) and 2.5(b) respectively. As expected, the percentile overhead of all methods tends to decrease with increased matrix size (with some fluctuation for small subblock sizes due to internal kernel optimizations of the utilized ATLAS library).

For execution time overhead when correcting SDCs, the execution time plot of Figure 2.9(b) and the theoretical prediction of Figure 2.5(b) show that the performance of ABFT could be worse than modular redundancy in multiple SDC scenarios. Specifically, while the proposed algorithm, mABFT and DMR incurs very low overhead for the correction of detected SDCs, ABFT requires (on average) more than 50% additional overhead for error correction. This is because, under multiple detected SDCs in GEMM, ABFT recomputes several rows and columns of the result, or indeed the entire GEMM subblock when ten erroneous rows/colums are detected ("rollback ABFT" [2]). This significant increase in the incurred overhead is also evident in the theoretical analysis of Proposition 2.2. It

is also evident that the additional overhead for tolerating multiple SDCs decreases considerably for all approaches, with the exception of ABFT, as the matrix size increases. For example, numerical packing requires 53.01% additional overhead to tolerate multiple SDCs for the 32×32 matrix, while requiring only 9.07% to tolerate the same proportion of SDCs in the 1152×1152 matrix. This property is shared amongst all exact error-location algorithms (like numerical packing and DMR) and is beneficial as the requirements for low-cost SDC correction techniques are more significant for large matrix sizes, where entire GEMM recomputation would lead to substantial performance degradation.

In terms of error detection, by injecting IUD bit flips in all the outputs of the two GEMM calls of the proposed approach (in the integer case) under an extensive SDC campaign, we verified experimentally that the **locations** of all SDCs were indeed detectable by the proposed approach. On the contrary, as detailed in Section 2.3.1, ABFT can reliably detect and correct only up to a single SDC within each GEMM product. ABFT requires recomputation of entire rows and columns to ensure no SDCs remain uncorrected, as discussed in the example of Figure 1.3. This is circumvented via the use of mABFT, which, under the utilized settings, can reliably detect the locations of up to 32 SDCs per GEMM subblock, albeit at the cost of substantial execution time overhead.

Overall, our theoretical analysis and experimental results demonstrate that our proposal offers very high accuracy and reliability in the detection of the locations of SDCs, while it comes with runtime overhead that is similar to that of ABFT when no SDCs occur.

2.5 Application in Energy-aware Computing Systems under Voltage Scaling for Visual Descriptor Matching in Image and Video Retrieval

Voltage scaling has been proposed as a means to improve battery life in portable devices, albeit at the risk of exposing the execution to transient faults in memory [133, 134] . For example, Narayanan and Xie [133] report that dropping the operating voltage of a 4Mbit-SRAM memory for one hour brings significant energy savings but also increases the number of memory errors from 57 to 658. Our approach can be used as the fault detection and mitigation framework when applying such a scenario within integer matrix products of multimedia applications. It may be argued that, because multimedia applications are inherently error tolerant [40], SDCs would always have a benign effect on the output results. However, recent studies [115–117] show that there exist "critical" sections of multimedia applications where mitigation of soft errors is imperative, especially when outputs from such sections are reused for subsequent computations. In view of this, in the next two subsections we present such an example within the context of state-of-the-art image and video retrieval algorithms and we also showcase the corresponding energy savings obtained when SDC tolerant algorithms are used in a vulnerable voltage-scaled environment.

2.5.1 Effect of SDCs on the Results of an Image or Video Retrieval Task

Consider a large image database preprocessed using the state-of-the-art vector of locally aggregated descriptors (VLAD) method of Jegou *et. al.* [9] to produce a compact descriptor for each database image comprising M integers, with $M \in [288, 1152]$. The database images could be standalone pictures, or consist of frames of several video sequences. In order to match a given query image (or video frame) with the database images, an inner product between the compact descriptor of the query image and the descriptor of each of the images in the database is computed [9]. Given that multiple query images (a.k.a., image "bunch") are matched through the stored database at any given moment (e.g., because of many concurrent users, or due to the use of video that results in multiple feature vectors per query), the matching operations are carried out via GEMM products between the descriptors of query image bunches and the ones of the database images.

In order to see the impact of SDCs in such a framework, we consider matching a query bunch comprising M image descriptors within 1000 segments of M database descriptors each (i.e., 1000 $M \times M$-by-$M \times M$ GEMMs), with the descriptors extracted from the Holidays dataset [9]. The experiment is set to establish the differences occurring in the retrieved images when running under SDC-free conditions versus when running under KULFI's single-row SDC case of Figure 2.9(b). We present an illustration of returned matches in both error free and erroneous cases

Table 2.1: Average number of discrepancies in the VLAD matches under one row of computed similarity measures being affected by SDCs.

Image bunch size M	No. of wrong VLAD matches
288	84.2 (28.61%)
384	111.4 (29.01%)
576	165.2 (28.68%)
1152	343.1 (29.78%)

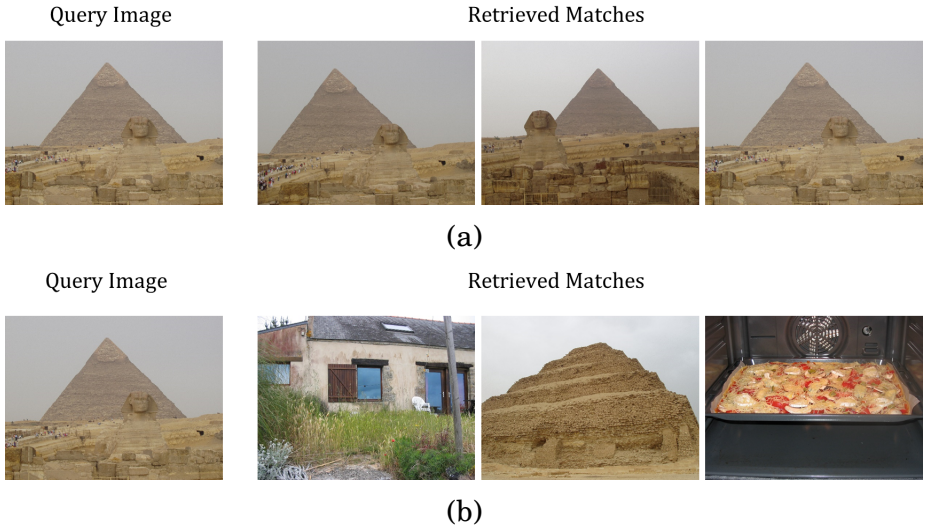


Figure 2.10: Indicative image matches (from the Holidays dataset [9]) in VLAD-based image retrieval when using the query image on the left for *(a)* an error free case; *(b)* a single-row erroneous case.

in Fig. 2.10. The average difference in the obtained results for various bunch sizes is presented in Table 2.1. It is evident that the compact nature of the image descriptors (which makes them sensitive to errors) leads to nearly 30% discrepancy between the error-free and the SDC execution of the descriptors. This results in irrelevant images being retrieved, as shown in Fig. 2.10. Therefore, in such image and video retrieval experiments, mitigation of SDCs is imperative in order to maintain reliable system operation.

2.5.2 Application in Energy-aware Computing Systems under Voltage Scaling Incurring High SDC Rates in Data Cache Memory

We present an example of the efficacy of our proposal as a power saving technique within systems where it is imperative to ensure reliable performance at reduced energy consumption.

Voltage scaling is commonly found today within power-aware computing for multimedia systems [135]. Recently, *voltage scaling* has been proposed as the means for achieving substantial reduction in energy consumption at the cost of transient cell failures in large memory arrays (e.g., data caches) [17]. We thus assume that the fault-intolerant parts of the multimedia processing algorithm (including data input, packing—for the proposed approach—and post-computation error detection and correction) operate at V_{safe} Volt, which ensures no SDCs occur at runtime, albeit at the cost of high power dissipation. However, the fault-tolerant parts (i.e., the GEMM calls) operate under scaled voltage V_{error} Volt (with $V_{\text{error}} < V_{\text{safe}}$), selected such that transient cell failures are occasionally encountered within the utilized data cache, while instruction cache is protected by on-chip variable-strength error-correcting codes (VS-ECC) that leave no errors exposed to the operating system and application layers [17].

2.5.3 Application Context

We performed experiments where ABFT, mABFT, DMR and the proposed approach are used to detect SDCs within the GEMM operations of the image retrieval task described in Section 2.5.1 for database segment sizes of 576 and 1152 images, and each image in the database represented by a descriptor comprising 576 and 1152 integers, respectively.

2.5.4 System Description and SDC Injection

The experiments were performed on an Intel i7-4578U 3GHz processor (Windows 8.1, Intel C++ Compiler 15.0.1). The Intel Extreme Tuning Utility was used for all voltage adjustments in our experiments and the Intel Power Gadget API [136] was used to measure the dynamic CPU power dissipation for the different approaches. The Intel Power Gadget allows for the retrieval of Intel processor features, including the estimated real time processor power and frequency [136] . By incorporating the C++ API version of the gadget within our fault tolerant implementation, power (as well as equivalent energy) measurements were obtained for the desired code sections. Specifically, a call to the `GetPowerData()` function of the Intel Power Gadget 3.0, returns frequency, power, temperature or package power limit measurements of the processor, by accessing the relevant drivers [136]. The power data, selected by setting `iMSR=1` in the call to `GetPowerData()`, returns power measurement in watts (W) and energy measurements in joules (J) and milliwatthour (mWh), for a given code section. We selected $V_{\text{safe}} = 1.37$ V for SDC-free operation and

$V_{\text{error}} = 0.98$ V for scaling. The chosen value for V_{error} is below the recommended voltage for our system, but was still found to be providing for safe operation. Further scaling causes actual failures in our setup (blue screen of death), but does not allow for any control in our experimental conditions. Therefore, in common with related work [126, 137–139], we opted to operate our platform at a safe voltage level and then inject SDCs artificially with probabilities of 10^{-4} and 10^{-5} per data cache access[6]. Thus, on average, 1 SDC occurs every $10,000$ and $100,000$ accesses during each GEMM computation, respectively. This ensures that all methods undergo the same SDC injection campaign. In summary, the chosen experimental setup emulates the case where the fault-tolerant parts of the application (operating at V_{error} V) are exposed to data-cache SDCs that are detected and corrected at runtime.

2.5.5 Energy Consumption Results

Table 2.2 shows the average CPU energy for the different approaches. Due to the increased number of recomputations of ABFT, its performance drops with increased SDC rates. Table 2.2 shows that the proposed approach provides for up to 35% reduction in energy consumption against the fault-intolerant GEMM and up to 96% reduction against the other alternatives. This is because:

1. like DMR, it achieves highly-reliable detection of the locations of

[6]Related work [17, 126, 137–139] has carried out experiments and simulation studies with similar SDC rates to the ones utilized here and we have verified that KULFI produces similar rates.

Table 2.2: Average CPU energy (mJ) and percentile comparison against the fault-intolerant (conventional) sGEMM operating at V_{safe}V for GEMM computations for matching a bunch of $N = 576$ and $N = 1152$ images against a database of images consisting of $M = 576$ and $M = 1152$ images respectively. For each case, each image is represented by an N-element VLAD.

Approach	SDC rate	Measured Energy (mJ)	% decrease in Energy	SDC rate	Measured Energy (mJ)	% decrease in Energy
$M = 576$						
sGEMM at V_{safe}	10^{-5}	133.92	0	10^{-4}	133.92	0
Proposed		127.07	5.11		133.15	0.57
ABFT [101]		238.30	-77.94		235.86	-76.12
mABFT [4–8]		148.20	-10.66		150.35	-12.27
DMR [95]		168.04	-25.47		176.00	-31.42
$M = 1152$						
sGEMM at V_{safe}	10^{-5}	1124.28	0	10^{-4}	1124.28	0
Proposed		730.86	34.99		1030.32	8.36
ABFT [101]		2148.78	-91.13		2154.37	-91.62
mABFT [4–8]		1349.31	-20.02		1381.31	-22.86
DMR [95]		1875.17	-66.79		2168.82	-92.91

Table 2.3: Summary of features of different methods for SDC mitigation in integer matrix products.

Feature \ Method	Proposed Fault-tolerant Numerical Packing	ABFT [101]	mABFT [4–8]	Dual Modular Redundancy [95]
Memory overhead	1 checksum element	1 additional row & col	12.5% more rows & cols	100% increase
Dynamic range loss	37.5% of bitwidth	equation (2.24)	equation (2.25)	0
Error correction capability	Recomputes erroneous pairs of locations	Corrects 1 SDC; recomputes rows & columns for >1 SDC	Corrects multiple SDCs	Recomputes erroneous pairs of locations
Mean execution overhead vs. fault-intolerant GEMM under no SDCs	11.50%	12.52%	52.09%	108.34%
Mean execution overhead vs. fault-intolerant GEMM for "single-row" of SDCs	21.21%	120.34%	85.40%	127.31%
Energy decrease with voltage scaling and SDC rate of 10^{-5}	20.05%	−84.54%	−15.34%	−46.13%

the vast majority of all the erroneous computations, while ABFT requires row and column recomputations given that, beyond the case of a single SDC per GEMM subblock, it cannot pin-point the SDC locations precisely (Figure 1.3);

2. it has substantially-reduced overhead in comparison to DMR and mABFT-based GEMM and its processing cycles are comparable to the fault-intolerant GEMM.

It can thus be considered as the best approach for SDC mitigation under aggressive voltage scaling in integer GEMM operations within energy-aware computing systems.

2.6 Conclusions

We proposed a new method for highly-reliable integer matrix products. Our approach inserts redundancy within the numerical representation of the inputs by exploiting the concept of numerical packing. Analysis of our proposal using high-performance generic matrix multiply (GEMM) routines demonstrated that high reliability comes at the cost of certain bitwidth loss and limited additional computations in comparison to the equivalent fault-intolerant GEMM. Specifically, we showed that when no SDCs are detected within GEMM outputs, numerical packing is expected to incur about 0.95% more computations than ABFT, while remaining 20.37% and 49.20% *more* computationally efficient than mABFT and DMR respectively. On the other hand, when multiple SDCs are detected, the

requirement for "rollback-ABFT" decreases its efficiency with numerical packing being the most efficient of all algorithms. We also show that by employing "rollback-ABFT" for error correction when more than one error is detected in GEMM outputs using the ABFT method, all methods show a very low likelihood of bitflips going undetected. Experimental results on an Intel Ivy Bridge processor, demonstrate that the average execution time overhead of the proposed method against high-performance, fault-intolerant, GEMM realizations under SDC-free conditions is limited to 11.50%. Therefore, the reliability and execution time efficiency of our method is comparable to that of ABFT and, unlike ABFT, our method is capable of detecting the locations of the vast majority of all possible SDCs. This makes our method superior to ABFT, mABFT and DMR under the occurrence of multiple SDCs in memory. A summary of the features, requirements and performance of the proposed method in reference to ABFT, mABFT and DMR is given in Table 2.3. The summary shows that, on average: *(i)* under no SDCs in the GEMM execution, our approach incurs only 2.51% higher overhead than algorithm-based fault-tolerance (ABFT), and is 18.04% and 46.37% more efficient than modified ABFT (mABFT) and dual modular redundancy (DMR), respectively; *(ii)* under a single-row data corruption emanating from the use of a fault injection tool, KULFI, the proposed approach is 17% to 47% more efficient than all other methods, as it can pin-point the locations of all detected SDCs. In terms of energy consumption requirements, based on an emulated SDC-injection campaign incurred in data cache memory under voltage scaling, we show that the proposed approach performs best in comparison to all other approaches. We therefore demonstrate that our

proposal paves the way for a new class of SDC mitigation methods for integer matrix products that are fast, energy efficient, and highly reliable.

Chapter 3

Numerical Packing: A Checksum-less Method for Fail-Stop Failure Mitigation

3.1 Introduction

Distributed computing clusters today provide for significant parallelism possibilities at the cost of decreased mean-time-to-failure (MTTF) estimates in comparison to single-core systems [19,23]. For example, Schroeder and Gibson [19] report over 20-fold increase in the annual average number of failures recorded for a 4096-processor HPC system in comparison to its 128-processor alternative. Therefore, multimedia applications requiring data-intensive and high-throughput processing (e.g., webpage or multimedia retrieval [29,30], relevance ranking [31] and object or face recog-

nition in video [32, 33]) on such clusters are now prone to core failures, occurring at increasing rates. Within all these applications, the compute and memory-intensive parts comprise large sum-of-product computations, i.e., inner and outer products, generic matrix multiplication (GEMM) [11, 35–40], and multidimensional convolution/cross-correlation (CONV) operations [41, 42]. These operations are typically performed using vectorized integer sum-of-product routines, or optimized single/double-precision floating-point libraries [e.g., Intel's `sGEMM` and `ippsConvolve_64f` library routines] [35, 36, 43]. Therefore, ensuring the robustness of these operations to core failures is of paramount importance for large-scale multimedia application deployment in cloud computing clusters exhibiting low MTTF characteristics.

In this chapter, we propose a novel FER mechanism for integer matrix multiplication and convolution operations that creates redundant results *within* the numerical representation of the output results using a variant of the numerical packing algorithm of Section (2.2). Similar to the design of algorithms to allow for SDC detection and correction, FER can be achieved by packing pairs of inputs within one integer number[1] and carrying out the actual GEMM or CONV operations with packed inputs. Because the packed outputs contain redundant sum-of-products, up to a certain number of failures of consecutive cores can be mitigated based

[1]The proposed algorithm is presented for integer representations, with the packed inputs being typecast to floating point in order to utilized optimized mathematics libraries for GEMM and CONV (e.g., `sGEMM`/`dGEMM` of Goto or Intel MKL [35, 36]). Even though the usage of floating-point representations for integer GEMM and CONV routines may seem counter-intuitive, it is in fact commonplace today since all processors have native support for floating point [140] and, in the case of CONV, floating point allows for Fourier-domain implementations.

on the available outputs. Importantly, the proposed approach does not require the processing of additional checksum inputs. Therefore, unlike all existing FER approaches, this is the first time a core failure recovery mechanism is proposed that *does not* require additional processor cores (e.g., for checksum computations or modular redundancy) in comparison to the conventional (fault-intolerant) routine.

Though the proposed algorithm focuses on fail-stop failure recovery, if all output data streams are received from all processing units, our proposal can also be used for the detection of SDCs. This is because all results can be retrieved by using only a subset of the entire computing cluster, thus allowing for cross-validation of obtained results for SDC mitigation.

Given that our algorithm can only be used for integer data computations, we focus on large-scale, low-latency, multimedia applications that require high-throughput integer-to-integer sum-of-product computations [40]. To quantify the complexity of the proposed approach, we derive its overhead in terms of arithmetic operations in comparison to the equivalent checksum-based method. We also present experimental results on *(i)* an 18-core, shared-memory, AWS EC2 instance[2], and *(ii)* a StarCluster [120] of AWS EC2 spot instances that are terminated and migrated to AWS EC2 on-demand instances for the duration that the spot price exceeds a predetermined threshold. For the former, we demonstrate that the proposed method achieves substantially-higher peak performance against the equivalent FER method based on checksums, both under failure-free and failure-occurring conditions. For the latter, we show that our ap-

[2]This is a `c4.8xlarge` AWS EC2 instance composed of multiple dual-nanocore physical processors (Intel Haswell) with hyperthreading.

proach provides for substantial reduction in deployment cost, especially in comparison to the failure-intolerant approach that cannot use spot instances. Finally, the source code for the proposed approach is made available online at https://github.com/NumericalPacking/Core-Failure-Mitigation-Code.

3.2 Proposed Algorithm Design

Let us consider: *(i)* L integer matrices $\mathbf{A}_0, \ldots, \mathbf{A}_{L-1}$ ($L \geq 3$), comprising $M \times N$ dimensions each, that must be processed[3] via an $N \times K$ integer kernel matrix $\mathbf{B}$ [141]; *(ii)* L integer input signal vectors $\mathbf{a}_0, , \ldots, \mathbf{a}_{L-1}$ ($L \geq 3$), comprising N samples each, that must be filtered by a K-sample integer kernel vector $\mathbf{b}$. In both cases, each operation is performed on a different processing unit[4] (PU) in an L-PU platform. We shall present an approach that can recover all L GEMM or CONV results for a single ($F = 1$) PU failure, *without* requiring additional checksum inputs. In order to achieve this, we utilize symmetric packing, where both input and kernel matrices are packed, as elaborated in Section 2.2.2. Symmetric packing allows for the production of redundant outputs, i.e., akin to the middle zone of k bits in the example of Fig. 2.3(f). As shown in this thesis, if constructed

[3]Supported algorithms include a range of linear and sesquilinear compute-intensive operations including GEMM, Kronecker product, and multidimensional CONV. In this thesis we focus on the two illustrative and important cases of matrix products and one-dimensional convolution operations since the remaining operations follow in a straightforward manner from these results.

[4]In this work, a processing unit (PU) refers to an individual unit of a parallel/distributed computing platform, performing a fraction of the workload and with independent failure characteristics. Therefore, a thread in a single core CPU, or a node with multiple processors within a HPC system is referred to as a PU, provided the system can continue operation even in the event of a PU failure.

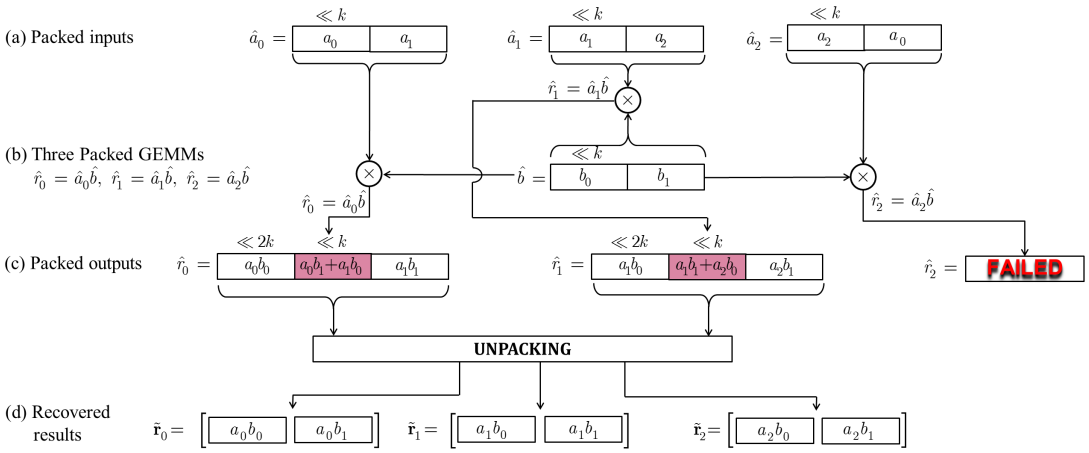


Figure 3.1: Illustration of failure mitigation in integer matrix product for the elementary case of three 1×1-by-1×2 matrix products in an integer representation with $k = 15$. Assuming that the PU that computes $\widehat{r}_2$ failed, the results of the other two PUs ($\widehat{r}_0$ and $\widehat{r}_1$) are used to produce all three outputs $\widetilde{r}_0$, $\widetilde{r}_1$ and $\widetilde{r}_2$ after unpacking. The partitioning within the rectangles shows the location (shifts by k or $2k$ bits) of inputs/outputs when packed within a single number.

appropriately, these redundant outputs allow for the recovery from a PU failure.

3.2.1 Proposed Packing Method for Failure-tolerant GEMM

The first step of the proposed approach packs pairs from the L input matrices, thereby generating L "packed" input matrices $\widehat{\mathbf{A}}_0, \ldots, \widehat{\mathbf{A}}_{L-1}$ given by ($0 \leq l < L$):

$$\widehat{\mathbf{A}}_l \quad = \quad \mathcal{S}_k \left\{ \mathbf{A}_l \right\} + \mathbf{A}_{(l+1) \bmod L} \tag{3.1}$$

where $\mathcal{S}_k\{\mathbf{A}_l\}$ is the elementwise left bit operator defined in page xiv and k the packing factor chosen such that:

$$k > \log_2\left(\max_{\forall l}|\mathbf{R}_l|\right) + 1 \text{ and } 3k \leq W, \tag{3.2}$$

with $W \in \{32, 64\}$ if all processing is carried out in 32-bit or 64-bit integer representations and $W \in \{24, 52\}$ if kernel processing is carried out in single or double-precision floating-point representations [41, 142].

For example, for $L = 3$, we have

$$\begin{aligned} \widehat{\mathbf{A}}_0 &= \mathcal{S}_k\{\mathbf{A}_0\} + \mathbf{A}_1 \\ \widehat{\mathbf{A}}_1 &= \mathcal{S}_k\{\mathbf{A}_1\} + \mathbf{A}_2 \\ \widehat{\mathbf{A}}_2 &= \mathcal{S}_k\{\mathbf{A}_2\} + \mathbf{A}_0. \end{aligned} \tag{3.3}$$

Packing of the kernel matrix $\mathbf{B}$ is also performed. This packing produces the $N \times \frac{K}{2}$ matrix[5] $\widehat{\mathbf{B}}$ by:

$$\widehat{\mathbf{B}} = \mathcal{S}_k\{\mathbf{B}_{\text{top}}\} + \mathbf{B}_{\text{bot}}. \tag{3.4}$$

where $\mathbf{B}_{\text{top}}$ and $\mathbf{B}_{\text{bot}}$ are the sub-matrices of $\mathbf{B}$ defined in the notational conventions table of pg. xix. Given that the packing factor k in (3.1)–(3.4) will increase the dynamic range of all $\widehat{\mathbf{A}}_l$ and $\widehat{\mathbf{B}}$, we utilize 64-bit representations for $\widehat{\mathbf{A}}_l$ and $\widehat{\mathbf{B}}$.

[5]For simplicity of exposition, we assume that K is even.

3.2.2 Packed GEMM Computations

All $M \times N$-by-$N \times \frac{K}{2}$ matrix products can be computed concurrently on L processors via the use of L 64-bit GEMM calls (e.g., OpenMP framework with MKL `dGEMM`),

$$\forall l : \widehat{\mathbf{R}}_l = \widehat{\mathbf{A}}_l \widehat{\mathbf{B}}, \tag{3.5}$$

thereby producing all required results, as well as a number of "duplicate" results within the numerical representation of the packed outputs $\widehat{\mathbf{R}}_l$. Fig. 3.1 illustrates the simple case of three 1×1-by-1×2 matrix products ($L = 3$, $M = N = 1$, $K = 2$) after packing has been carried out via (3.3) and (3.4). The contents of the $M \times \frac{K}{2}$ output matrices $\widehat{\mathbf{R}}_0$, $\widehat{\mathbf{R}}_1$ and $\widehat{\mathbf{R}}_2$ can be expressed mathematically by:

$$\begin{aligned}
\widehat{\mathbf{R}}_0 &= \widehat{\mathbf{A}}_0 \widehat{\mathbf{B}} \\
&= \mathcal{S}_{2k} \left\{ \mathbf{A}_0 \mathbf{B}_{\text{top}} \right\} + \mathcal{S}_k \left\{ \mathbf{A}_0 \mathbf{B}_{\text{bot}} + \mathbf{A}_1 \mathbf{B}_{\text{top}} \right\} + \mathbf{A}_1 \mathbf{B}_{\text{bot}} \\
&= \mathcal{S}_{2k} \left\{ \widetilde{\mathbf{R}}_{0,\text{top}} \right\} + \mathcal{S}_k \left\{ \widetilde{\mathbf{R}}_{0,\text{bot}} + \widetilde{\mathbf{R}}_{1,\text{top}} \right\} + \widetilde{\mathbf{R}}_{1,\text{bot}}
\end{aligned} \tag{3.6}$$

$$\begin{aligned}
\widehat{\mathbf{R}}_1 &= \widehat{\mathbf{A}}_1 \widehat{\mathbf{B}} \\
&= \mathcal{S}_{2k} \left\{ \mathbf{A}_1 \mathbf{B}_{\text{top}} \right\} + \mathcal{S}_k \left\{ \mathbf{A}_1 \mathbf{B}_{\text{bot}} + \mathbf{A}_2 \mathbf{B}_{\text{top}} \right\} + \mathbf{A}_2 \mathbf{B}_{\text{bot}} \\
&= \mathcal{S}_{2k} \left\{ \widetilde{\mathbf{R}}_{1,\text{top}} \right\} + \mathcal{S}_k \left\{ \widetilde{\mathbf{R}}_{1,\text{bot}} + \widetilde{\mathbf{R}}_{2,\text{top}} \right\} + \widetilde{\mathbf{R}}_{2,\text{bot}}
\end{aligned} \tag{3.7}$$

$$\begin{aligned}
\widehat{\mathbf{R}}_2 &= \widehat{\mathbf{A}}_2 \widehat{\mathbf{B}} \\
&= \mathcal{S}_{2k} \left\{ \mathbf{A}_2 \mathbf{B}_{\text{top}} \right\} + \mathcal{S}_k \left\{ \mathbf{A}_2 \mathbf{B}_{\text{bot}} + \mathbf{A}_0 \mathbf{B}_{\text{top}} \right\} + \mathbf{A}_0 \mathbf{B}_{\text{bot}} \\
&= \mathcal{S}_{2k} \left\{ \widetilde{\mathbf{R}}_{2,\text{top}} \right\} + \mathcal{S}_k \left\{ \widetilde{\mathbf{R}}_{2,\text{bot}} + \widetilde{\mathbf{R}}_{0,\text{top}} \right\} + \widetilde{\mathbf{R}}_{0,\text{bot}}.
\end{aligned} \tag{3.8}$$

3.2.3 Unpacking of the Results

Our key insight is that out of the L packed matrices $\widehat{\mathbf{R}}_0, \ldots, \widehat{\mathbf{R}}_{L-1}$, $L-1$ matrices suffice for the recovery of all L outputs. For example, for $L = 3$, any two matrices out of $\widehat{\mathbf{R}}_0$, $\widehat{\mathbf{R}}_1$ and $\widehat{\mathbf{R}}_2$ can produce all outputs, $\widetilde{\mathbf{R}}_0$, $\widetilde{\mathbf{R}}_1$ and $\widetilde{\mathbf{R}}_2$, as illustrated in Fig. 3.1. For instance, assuming $\widehat{\mathbf{R}}_0$ and $\widehat{\mathbf{R}}_1$ are used for the recovery of $\widetilde{\mathbf{R}}_0$, $\widetilde{\mathbf{R}}_1$ and $\widetilde{\mathbf{R}}_2$, the required steps are given by:

$$\widetilde{\mathbf{R}}_{0,\text{top}} = \mathcal{S}_{-2k}\left\{\widehat{\mathbf{R}}_0\right\} \tag{3.9}$$

$$\mathbf{T}_0 = \widehat{\mathbf{R}}_0 - \mathcal{S}_{2k}\left\{\widetilde{\mathbf{R}}_{0,\text{top}}\right\} \tag{3.10}$$

$$\mathbf{T}_1 = \mathcal{S}_{-k}\left\{\mathbf{T}_0\right\}. \tag{3.11}$$

Because of the complement-two arithmetic used in integer representations in commodity hardware, all negative elements (i,j) of $\mathbf{T}_1$ will be found to be larger or equal to 2^{k-1} (maximum positive value within a packed output element) and $\widetilde{R}_{0,\text{top}}[i,j]$ (contained in the most-significant bits of $\widehat{R}_0[i,j]$) will be found to be one less than their correct value. To compensate for these effects of the complement-two arithmetic, we first define the intermediate matrix $\mathbf{E}_0$ to store the signed values of $\mathbf{T}_1$ therein, and adjust the values of $\widetilde{\mathbf{R}}_{0,\text{top}}$ as follows:

- $\forall i,j$: if $T_1[i,j] \geq 2^{k-1}$, then set $E_0[i,j] = T_1[i,j] - 2^k$ (convert to negative number); else set $E_0[i,j] = T_1[i,j]$ (no change);
- $\forall i,j$: if $E_0[i,j] < 0$, then set $\widetilde{R}_{0,\text{top}}[i,j] \leftarrow \left(\widetilde{R}_{0,\text{top}}[i,j] + 1\right)$.

Next, a similar check is performed to extract the signed representation of $\widetilde{\mathbf{R}}_{1,\text{bot}}$ and the values of $\mathbf{E}_0$ are adjusted, i.e.,

$$\mathbf{T}_0 \leftarrow [\mathbf{T}_0 - \mathcal{S}_k \{\mathbf{T}_1\}], \tag{3.12}$$

- $\forall i,j$: if $T_0[i,j] \geq 2^{k-1}$, then set $\widetilde{R}_{1,\text{bot}}[i,j] = T_0[i,j] - 2^k$; else set $\widetilde{R}_{1,\text{bot}}[i,j] = T_0[i,j]$;
- $\forall i,j$: if $\widetilde{R}_{1,\text{bot}}[i,j] < 0$, then set $E_0[i,j] \leftarrow (E_0[i,j] + 1)$.

Similarly, $\widehat{\mathbf{R}}_1$ undergoes the same processing as $\widehat{\mathbf{R}}_0$ in order to extract $\widetilde{\mathbf{R}}_{1,\text{top}}$, the intermediate matrix $\mathbf{E}_1 = \widetilde{\mathbf{R}}_{1,\text{bot}} + \widetilde{\mathbf{R}}_{2,\text{top}}$ and $\widetilde{\mathbf{R}}_{2,\text{bot}}$. Finally, we perform the following operations to complete the extraction of all results:

$$\widetilde{\mathbf{R}}_{0,\text{bot}} = \mathbf{E}_0 - \widetilde{\mathbf{R}}_{1,\text{top}} \tag{3.13}$$

$$\widetilde{\mathbf{R}}_{2,\text{top}} = \mathbf{E}_1 - \widetilde{\mathbf{R}}_{1,\text{bot}}. \tag{3.14}$$

Importantly, the `_mm256_shuffle_epi32` and `_mm256_permute2f128_si256` instructions (which are intrinsically supported in modern SIMD architectures [143]) can further be used to optimize the output matrix reordering using the "top" and "bot" index subsets.

3.2.4 Proposed Approach for One-dimensional Convolution

The same process can be used to mitigate PU failures within L parallel convolution/cross-correlation (CONV) operations of N-sample input sig-

nals $\mathbf{a}_l$ with a K-sample kernel $\mathbf{b}$, producing output vectors $\mathbf{r}_l$ by ($0 \le i < N+K-1,\ \ 0 \le l < L$):

$$\mathbf{r}_l = \mathbf{a}_l \star \mathbf{b} \iff r_l[i] = \sum_{k=0}^{K-1} a_l[i-k]\, b[k] \ . \tag{3.15}$$

Specifically, the proposed algorithm packs the signal vectors, $\mathbf{a}_l$, into the packed inputs, $\widehat{\mathbf{a}}_l$, following the same procedure as for (3.1), while the convolution kernel $\mathbf{b}$ is packed into $\widehat{\mathbf{b}}$ as shown for GEMM in (3.4).

The convolutions ($0 \le l < L$):

$$\forall l: \ \widehat{\mathbf{r}}_l = \widehat{\mathbf{a}}_l \star \widehat{\mathbf{b}} \tag{3.16}$$

can then be carried out with any optimized library (e.g., Intel's IPP `ippsConvolve_64f` routine [43], Matlab's `conv` function, etc.) in order to produce the $\left(N + \frac{K}{2} - 1\right)$-sample signals $\widehat{\mathbf{r}}_0, \ldots, \widehat{\mathbf{r}}_{L-1}$. In addition, unpacking follows the same procedure as for GEMM, except for an additional summation operation. For example, for $L = 3$, given the unpacked outputs $\widetilde{\mathbf{r}}_{0,\text{top}}$, $\widetilde{\mathbf{r}}_{0,\text{bot}}$, $\widetilde{\mathbf{r}}_{1,\text{top}}$, and $\widetilde{\mathbf{r}}_{1,\text{bot}}$, the "top" vectors comprise output signal samples with indices within $\left[0, N + \frac{K}{2} - 2\right]$, while the "bot" vectors comprise signals with indices within $\left[\frac{K}{2}, N + K - 2\right]$. The recovered output signals are obtained by the following concatenation operations ($0 \le l < 3$):

$$\forall l: \ \widetilde{\mathbf{r}}_l = \left[\widetilde{\mathbf{r}}_{l,\text{top}} \quad \mathbf{0}_{\frac{K}{2}}\right] + \left[\mathbf{0}_{\frac{K}{2}} \quad \widetilde{\mathbf{r}}_{l,\text{bot}}\right] \tag{3.17}$$

with $\mathbf{0}_{\frac{K}{2}}$ the $1 \times \frac{K}{2}$ vector of zeros.

3.2.5 Summary of Key Results

In order to recover from the fail-stop failure models described in this work, checksum-based methods produce up to F checksum matrices, $\mathbf{A}_{c1}, \ldots, \mathbf{A}_{cF}$. The number of checksum matrices is dependent on the number of PU failures that should be tolerated in the parallel GEMM execution. The simplest checksum matrix that can be used to recover from a single PU failure in the L GEMM products ($0 \leq l < L$)

$$\mathbf{R}_l = \mathbf{A}_l\mathbf{B} \tag{3.18}$$

is

$$\mathbf{A}_{c1} = \sum_{l=0}^{L-1} \mathbf{A}_l. \tag{3.19}$$

This checksum matrix undergoes the same GEMM operation, $\mathbf{R}_{c1} = \mathbf{A}_{c1}\mathbf{B}$, which requires the usage of an additional PU. The dynamic range of $\mathbf{R}_{c1}$ is increased by L in comparison to GEMM products $\mathbf{A}_l\mathbf{B}$. If the xth PU fails, $0 \leq x < L$, we recover $\mathbf{R}_x$ by $\mathbf{R}_x = \mathbf{R}_{c1} - \sum_{\forall l \neq x} \mathbf{R}_l$. Additional checksums may be added to mitigate more than one failure by using different linear combinations of the $\mathbf{A}_0, \ldots, \mathbf{A}_{L-1}$ matrices, as shown by Stefanidis and Luk [4, 5]. Overall, in order to tolerate F PU failures in a parallel computing environment, F additional PUs are set aside to compute the results of the linear-checksum matrices [100, 113, 114], $\mathbf{R}_{cf} = \mathbf{A}_{cf}\mathbf{B}$, $1 \leq f \leq F$.

On the other hand, the proposed method utilizes packing to create L cyclically-duplicated/double-bitwidth descriptions for the inputs of ma-

trix product and convolution operations. The packed inputs that are then used within L 64-bit GEMM or CONV operations on L PUs. The proposed unpacking process recovers all GEMM or CONV outputs with operations count that depends only on the number of outputs and not on the inner-product dimension of the GEMM or CONV processing. The following four propositions formalize these key points.

Proposition 3.1. *Let the packing of* (3.1) *and* (3.4)*, and the execution of the L GEMM operations of* (3.5) *on L independent PUs. All output GEMM results can be recovered under the failure of any PU.*

Proof. It suffices to show that, given a PU failure with the corresponding failed GEMM denoted by: $\widehat{\mathbf{R}}_f$ $(f \in [0, L-1])$, the proposed algorithm can recover all L GEMM outputs.

Following the packing algorithm of (3.1) and (3.4), any two packed GEMM outputs can be used for the recovery of three output results. Specifically, we recover the set $\widetilde{\mathbf{R}}_f, \widetilde{\mathbf{R}}_{(f+L-1) \bmod L}$ and $\widetilde{\mathbf{R}}_{(f+L-2) \bmod L}$ using only packed outputs, $\widehat{\mathbf{R}}_{(f+L-1) \bmod L}$ and $\widehat{\mathbf{R}}_{(f+L-2) \bmod L}$ expressed by:

$$\begin{aligned} \widehat{\mathbf{R}}_{(f+L-2) \bmod L} &= \mathcal{S}_{2k}\left\{\widetilde{\mathbf{R}}_{(f+L-2) \bmod L,\text{top}}\right\} \\ &+ \mathcal{S}_{k}\left\{\widetilde{\mathbf{R}}_{(f+L-2) \bmod L,\text{bot}}\right\} \\ &+ \mathcal{S}_{k}\left\{\widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{top}}\right\} \\ &+ \widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{bot}}, \end{aligned} \tag{3.20}$$

$$\begin{aligned}\widehat{\mathbf{R}}_{(f+L-1) \bmod L} &= \mathcal{S}_{2k}\left\{\widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{top}}\right\} \\ &+ \mathcal{S}_{k}\left\{\widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{bot}}\right\} \\ &+ \mathcal{S}_{k}\left\{\widetilde{\mathbf{R}}_{f,\text{top}}\right\} + \widetilde{\mathbf{R}}_{f,\text{bot}}.\end{aligned} \tag{3.21}$$

First, the output shifted by $2k$ (equivalent to the "top" subset of columns of $\widetilde{\mathbf{R}}_{(f+L-2) \bmod L}$), is extracted from the packed GEMM of (3.20). The obtained result is then removed from (3.20) in order to obtain the outputs scaled by k and 1 (to be used in subsequent steps) by:

$$\widetilde{\mathbf{R}}_{(f+L-2) \bmod L,\text{top}} = \mathcal{S}_{-2k}\left\{\widehat{\mathbf{R}}_{(f+L-2) \bmod L}\right\} \tag{3.22}$$

$$\mathbf{T}_0 = \widehat{\mathbf{R}}_{(f+L-2) \bmod L} - \mathcal{S}_{2k}\left\{\widetilde{\mathbf{R}}_{(f+L-2) \bmod L,\text{top}}\right\} \tag{3.23}$$

$$\mathbf{T}_1 = \mathcal{S}_{-k}\left\{\mathbf{T}_0\right\}. \tag{3.24}$$

Because of the complement-two arithmetic used in integer representations in commodity hardware, all negative elements (i,j) of $\mathbf{T}_1$ will be found to be larger or equal to 2^{k-1} (maximum positive value within a packed output element) and $\widetilde{R}_{(f+L-2) \bmod L,\text{top}}[i,j]$ (contained in the most-significant bits of $\widehat{R}_{(f+L-2) \bmod L}[i,j]$) will be found to be one less than their correct value. To compensate for these effects of the complement-two arithmetic, we first define the intermediate matrix $\mathbf{E}_0$ to store the signed values of $\mathbf{T}_1$ therein, and adjust the values of $\widetilde{\mathbf{R}}_{(f+L-2) \bmod L,\text{top}}$ as follows:
$\forall i,j$: if $T_1[i,j] \geq 2^{k-1}$, then set $E_0[i,j] = T_1[i,j] - 2^k$ (convert to negative number); else set $E_0[i,j] = T_1[i,j]$ (no change);
$\forall i,j$: if $E_0[i,j] < 0$, then set $\widetilde{R}_{(f+L-2) \bmod L,\text{top}}[i,j] \leftarrow \left(\widetilde{R}_{(f+L-2) \bmod L,\text{top}}[i,j] + 1\right)$.
Next, a similar check is performed to extract the signed representation of

$\widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{bot}}$ and the values of $\mathbf{E}_0$ are adjusted, i.e.:

$$\mathbf{T}_0 \leftarrow \left[\mathbf{T}_0 - \mathcal{S}_k\left\{\mathbf{T}_1\right\}\right], \tag{3.25}$$

$\forall i,j$: if $T_0[i,j] \geq 2^{k-1}$, then set $\widetilde{R}_{(f+L-1) \bmod L,\text{bot}}[i,j] = T_0[i,j] - 2^k$; else set $\widetilde{R}_{(f+L-1) \bmod L,\text{bot}}[i,j] = T_0[i,j]$; $\forall i,j$: if $\widetilde{R}_{(f+L-1) \bmod L,\text{bot}}[i,j] < 0$, then set $E_0[i,j] \leftarrow (E_0[i,j]+1)$.

Similarly, $\widehat{\mathbf{R}}_{(f+L-1) \bmod L}$ undergoes the same processing as $\widehat{\mathbf{R}}_{(f+L-2) \bmod L}$ in order to extract $\widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{top}}$, the intermediate matrix $\mathbf{E}_1 = \widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{bot}} + \widetilde{\mathbf{R}}_{f,\text{top}}$ and $\widetilde{\mathbf{R}}_{f,\text{bot}}$. Finally, we perform the following operations to complete the extraction of all results:

$$\widetilde{\mathbf{R}}_{(f+L-2) \bmod L,\text{bot}} = \mathbf{E}_0 - \widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{top}} \tag{3.26}$$

$$\widetilde{\mathbf{R}}_{f,\text{top}} = \mathbf{E}_1 - \widetilde{\mathbf{R}}_{(f+L-1) \bmod L,\text{bot}}. \tag{3.27}$$

The remaining output GEMMs can be recovered using the described unpacking steps, with a pair of packed outputs producing a triplet of matrix outputs. We have thus shown that the lost $\widetilde{\mathbf{R}}_f$ GEMM output, together with the outputs $\widetilde{\mathbf{R}}_i$ $(0 \leq i < L, i \neq f)$, have been recovered. □

Proposition 3.2. *Let the packing of* (3.1) *and* (3.4)*, and the execution of the L CONV operations of* (3.16) *on L independent PUs. The results of* (3.15) *can be recovered under the failure of any PU.*

Proof. The proof follows the steps of the proof of Proposition 3.1, with the change of matrices to vectors and is omitted for brevity of description. Once all vectors $\widetilde{\mathbf{r}}_{l,\text{top}}$ and $\widetilde{\mathbf{r}}_{l,\text{bot}}$ have been recovered $(0 \leq l < L)$, given

that the "top" vectors comprise output signal samples with indices within $\left[0, N + \frac{K}{2} - 2\right]$ while the "bot" vectors comprise signals with indices within $\left[\frac{K}{2}, N + K - 2\right]$, the concatenation operations of (3.17) are required to obtain the final outputs. □

From the proofs of Propositions 3.1 and 3.2, it is evident that only two out of three PUs are required for each set of unpacking. Therefore, up to a total of $\lfloor \frac{L}{3} \rfloor$ PU failures can be mitigated in the entire parallel/distributed computing system, provided that no more than one failure is encountered within each triplet.
Furthermore, if (3.2) holds, then the unpacking process guarantees that the correct value is obtained for each output under the use of a 64-bit integer representation (assuming 32-bit inputs).

While the proposed approach (approximately) halves the overall multiply-accumulate (MAC) operations against the conventional approach during GEMM, all operations are performed in 64-bit representations (assuming 32-bit inputs). Under the assumption that 64-bit arithmetic operations require twice the cycles of 32-bit arithmetic operations, which (amongst others) holds for AVX2-based realizations, we can quantify the arithmetic operations (additions and MAC operations) required by the proposed method in comparison to the conventional and checksum-based failure mitigation approach. This is quantified in the following proposition. We also focus on the case of up to $F = \lfloor \frac{L}{3} \rfloor$ failures as the same complexity is required for mitigating $F = 1$ PU failures.

Proposition 3.3. *In the absence of failures during L concurrent $N \times N$-by-*

$N \times N$ matrix products, and assuming that a 64-bit operation is equivalent to two 32-bit operations, the number of arithmetic operations required by the proposed approach is:

$$\frac{2NF + 5F - 8L - 1}{2NL + 2NF - 2F} \times 100\% \tag{3.28}$$

less than the conventional checksum-based roll-forward.

Proof. In the proposed approach, the packing of the L input matrices $\mathbf{A}_0, \ldots, \mathbf{A}_{L-1}$ and the $\mathbf{B}$ matrix requires $\frac{N^2}{2}(2L+1)$ addition operations [see (2.6) and (3.4) and ignoring all arithmetic shift operations] and the packed matrix products require $\frac{L}{2}(2N^3 - N^2)$ operations. In terms of recovery, it can be shown via the analysis of Section 3.2.3 that $\frac{7}{2}N^2(L-F)$ operations are required to extract all outputs from the $L-F$ matrices. Given that the proposed method requires double the bitwidth of the conventional failure-intolerant GEMM for its computation, the cycles count of these computations will also be doubled. Therefore, by doubling the sum of all arithmetic operations, the arithmetic operations of the proposed approach are:

$$\mathrm{Cx}\{\text{proposed}\} = N^2(2NL + 8L - 7F + 1). \tag{3.29}$$

For the mitigation of a single PU failure or a group of F failures using the unweighted checksum-based method, $N^2(L-F)$ additions are required for the checksum generation. Subsequently, $L+F$ GEMMs are computed, which correspond to $(2N^3 - N^2)(L+F)$ operations. Post-processing is not required for the checksum-based method when no failures are encoun-

tered or when the failed PUs are the PUs holding the checksum matrices. In such cases, the arithmetic operations performed by the checksum-based method are:

$$\text{Cx}\{\text{no checksum failures}\} = N^2(2NF - 2F + 2NL) \tag{3.30}$$

Combining (3.29) and (3.30) to calculate the percentile reduction in the arithmetic operations stemming from the proposed approach, we reach (3.28). □

For example, for $N = 576$, $L = 6$ and $F = \lfloor \frac{L}{3} \rfloor = 2$, Proposition 3.3 shows that our approach is 24.59% more efficient than the conventional checksum-based approach.

Proposition 3.4. *When F PU failures occur during L concurrent $N \times N$-by-$N \times N$ matrix products, the number of arithmetic operations required by the proposed approach is up to:*

$$\frac{2NF + 4F - 7L - 1}{2NF + 2NL - 3F + L} \times 100\% \tag{3.31}$$

less than the conventional checksum-based roll-forward.

Proof. The proposed approach requires the operations given by (3.29) regardless of whether failures occurred or not. However, under the occurrence of a group of F PU failures, the checksum-based method must solve a system of linear equations to recover the lost data. Assuming the failed PUs correspond to the PUs computing the actual data outputs and not

the checksum outputs, it can be shown that $N^2(L-F)$ operations are required for data recovery. Therefore, the overall number of arithmetic operations of the checksum-based roll-forward method when F fail-stop failures are encountered is:

$$\text{Cx}\{\boldsymbol{F} \text{ checksum failures}\} = N^2(2NF - 3F + 2NL + L) \tag{3.32}$$

Combining (3.29) and (3.32) to calculate the percentile reduction in the arithmetic operations stemming from the proposed approach, we reach (3.31). □

For example, for $N = 576$, $L = 6$ and $F = \lfloor \frac{L}{3} \rfloor = 2$, Proposition 3.4 shows that numerical packing is 24.62% more efficient than the checksum-based method. We note, however, that this is the worst case scenario for the checksum-based method as some of the failed PUs could be PUs computing a checksum result, thus requiring no recovery.

In terms of resilience to multiple non-consecutive failures, similar to the checksum approach of Jou and Abraham [102], the proposed approach would either need additional processing units, i.e., as proposed by Luk, Rexford, *et al.* [4, 6], or a generalized version of the proposed packing mechanism which we present in succeeding chapter.

Table 3.1 presents examples of the arithmetic complexity per output derived from the calculations of Proposition 3.3 when $L = 16$ cores are available for data computation. We focus on the fail-free case as a baseline for comparisons, since, under the occurrence of failures, our approach

Table 3.1: Giga-operations per output [and percentile overhead in comparison to the failure-intolerant (conventional) GEMM computation] for $N \times N$-by-$N \times N$ GEMM products when mitigating $F = 4$ failures in an $L = 16$ core computing platform.

Method	Number of operations per output ($\times 10^9$)		
	$N = 1152$	$N = 4608$	$N = 9216$
Failure-intol.	3.06	195.67	1565.43
Proposed	3.07 (0.33%)	195.82(0.08%)	1566.05(0.04%)
Checksum	4.06 (32.68%)	260.92 (33.35%)	2087.35 (33.34%)

will outperform the checksum-based method with an even higher margin. The results of Table 3.1 show that, although the proposed approach requires a limited number of arithmetic operations for packing and unpacking, the production of L matrices using L PUs makes its operations' count comparable to the conventional, failure-intolerant, GEMM. On the other hand, the checksum-based method produces only $L - F$ matrices via the L utilized PUs, thereby leading to substantially-increased operations-per-output in comparison to both the failure-intolerant GEMM and the proposed approach. On average, Table 3.1 shows that numerical packing incurs about 0.15% additional computations per output for packing and unpacking in comparison to the failure-intolerant computation, while the checksum approach requires 33.12% more computations to produce the same number of outputs. The practical overhead incurred by the proposed and the checksum-based approach is investigated via the experiments of the following section.

3.3 Experimental Results on a Shared-memory 18-core AWS EC2 Instance

We benchmark our proposal for *core failure* mitigation using a compute-optimized `c4.8xlarge` instance of AWS EC2 (Intel Xeon E5-2666v3 2.9GHz cores, on-demand instance type, Windows Server 2012, Intel C++ 15.0 Compiler, Intel MKL for GEMM operations, 36 EC2 virtual cores corresponding to 18 physical cores with shared memory). Each individual throughput or execution-time result in our experiments corresponds to the average of 1000 runs with randomly selected inputs from the utilized datasets.

3.3.1 Execution Time Comparison for Failure Mitigation in Parallel GEMM Computations

Given that the optimal performance of the Intel MKL GEMM library is obtained when parallel computations are done in physical cores [144, 145], we set the system affinity to 18 physical cores rather than the 36 EC2 virtual cores. This configuration can accommodate a shared-memory computing cluster comprising four quadcore processors and we can present results for data recovery when all computations in any of the quadcore CPUs is lost. All packing, unpacking and checksum generation make use of the parallel computing capability of the computing environment via the OpenMP framework, as well as the increased optimization level offered by AVX2 SIMD instructions.

Table 3.2: Average execution time results (in milliseconds) and percentile overhead in comparison to the failure-intolerant (conventional) GEMM computation when mitigating $F = 4$ (one quadcore) failures in $L = 16$ GEMM computations.

Method	$N = 1152$	$N = 4608$	$N = 9216$
Failure-intol.	3.19	164.13	1255.06
Proposed	4.24 (32.92%)	194.98 (18.8%)	1415.66 (12.8%)
Checksum	4.55 (42.63%)	224.27 (36.64%)	1695.44 (35.09%)

Table 3.2 presents the average execution time for all methods when $L = 16$ for the computation of $(0 \leq l < 16)$:

$$\mathbf{R}_l = \mathbf{A}_{l,(N\times N)}\mathbf{B}_{N\times N}. \tag{3.33}$$

By comparing Table 3.1 and Table 3.2, it is evident that the performance ranking of the methods follows the theoretical analysis of Section 3.2.5. While the proposed approach incurs higher overhead than what is theoretically predicted in Table 3.1 (primarily due to variation in execution time between 64-bit and 32-bit memory and arithmetic operations), it is still offering substantial execution-time improvement against the checksum-based failure mitigation approach, particularly as the matrix product dimension increases.

Fig. 3.2 presents the experimental peak performance[6] achieved by each approach including all pre- and post-processing. We note that the peak performance achieved by all methods is slightly reduced because of the

[6]Each of the utilized Intel Xeon E5-2666v3 cores achieves $V = 92.8$ GFlop/s (operating at 2.9 GHz with 32 floating-point operations per cycle under AVX2 instructions). The peak performance achieved by each method is calculated by $\frac{U(2N^3-N^2)}{tVL} \times 100\%$, where U is the number of matrices of output results (i.e., $U = L$ for the conventional and proposed methods and $U = L - F$ for the checksum-based approach) and t is the total execution time (in seconds) for each case.

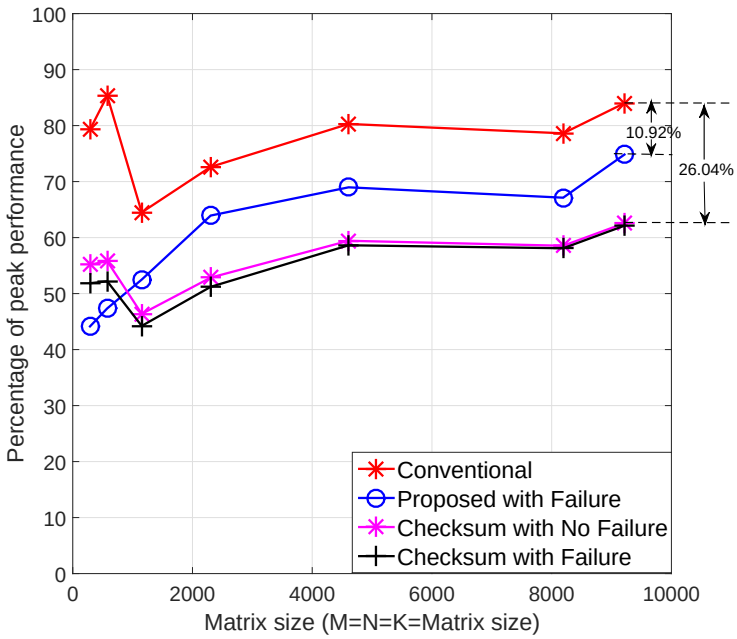


Figure 3.2: Peak performance achieved by each method in the utilized distributed computing environment for $L = 16$.

overheads of multicore systems with shared memory, i.e., it is well known that all practical GEMM realizations will attain less than 90% of peak performance in such a multicore cluster [35, 144, 145]. Furthermore, since the proposed approach and the checksum-based method require additional operations for failure mitigation, reduced peak performance is expected in comparison to the failure-intolerant (conventional) approach. Our results show that the conventional (failure-intolerant) GEMM achieves 64.50% to 85.31% of peak performance, while the proposed method for failure mitigation achieves 43.25% to 74.47% of peak performance. On the other hand, checksum-based failure mitigation achieves only 45.27% to 62.18% of peak performance. Overall, our theoretical analysis and experimental results demonstrate that our proposal offers the same reliabil-

ity to core failures as conventional checksum-based methods, albeit with peak performance that becomes comparable to the conventional failure-intolerant approach as the inner sum-of-product dimension grows.

3.3.2 Execution Time Comparison for Failure Mitigation in Parallel Cross-correlation Computations

We now benchmark our approach and the checksum-based approach for core failure recovery in the multicore execution of the CONV-based music retrieval algorithm of Ellis *et. al.* [146]. The algorithm identifies cover songs within a music database by extracting beat and tempo data from the query song track and performing cross correlation of the beat and tempo feature vectors with the beat and tempo vectors of each of the songs in the database [146] . The dominant computation within this process comprises two-dimensional cross-correlation between a matrix of audio beat and tempo data and a database of such matrices. Using this algorithm, we perform a music retrieval experiment with the music database comprising beat-tempo data from the Million-song dataset subset available at Columbia University's LabROSA repository [147] and the "1517-Artists" dataset of Seyerlehner *et. al.* [148].

The algorithm tracks beats of each of the input music tracks and generates a twelve-dimensional "chroma" representation specifying the pitches of the twelve distinct semitones in the western octave [146]. We used 100 beats to describe every music track. Thus, each music tack is composed of a 12×100 matrix representing its beat-chroma features. We then mod-

Table 3.3: Execution time (in milliseconds) and percentile difference in comparison to the conventional failure-intolerant processing for recovery after a quadcore processor failure in a music retrieval system.

Method Database size	Conventional Failure-intolerant	Proposed Failure-tolerant	Checksum Failure-tolerant
4992	382.26	530.44 (38.8%)	529.68 (38.6%)
7488	490.85	629.41 (28.2%)	692.30 (41.0%)
9984	579.62	726.19 (25.3%)	812.98 (40.3%)
12480	729.13	829.82 (13.8%)	968.92 (32.9%)
17472	926.58	1046.79 (13.0%)	1308.99 (41.3%)

ified the Matlab code of Ellis *et. al.* [146] for parallel execution of the music similarity measurement. Specifically, the Matlab `spmd` function was used to set up the parallel computing environment. Given that Matlab uses only the physical cores of the computing unit for parallel processing (see `feature('numCores')`command in Matlab), we distribute the music database amongst 16 cores for the conventional computation and proposed algorithm, while the checksum-based method uses 12 cores for data storage and the other 4 cores for checksum data. All preprocessing for beat and feature extraction from the audio clips is performed offline, i.e., prior to the actual retrieval task, and it is not carried out by the cloud-computing server of our experiment, which is only used for the actual retrieval task.

Table 3.3 shows the average execution time (in milliseconds) for the search and retrieval of a music track within various database sizes. The results demonstrate that the performance of the proposed algorithm converges to that of the conventional, failure-intolerant, implementation as the database size increases. On the other hand, the checksum-based method is found to incur considerable (and largely consistent) execution

time overhead (33%–41%) across all database sizes.

3.4 Image Retrieval Based on Terminatable AWS EC2 Spot Instances

To illustrate the efficacy of our proposal in *instance failure* mitigation when carrying out integer sum-of-product computations with terminatable instances, we performed several medium-scale image retrieval experiments using the state-of-the-art vector of locally aggregated descriptors (VLAD) method of Jegou *et. al.* [149] deployed on a cluster of AWS EC2 spot instances.

3.4.1 Application Description

The preprocessing done by the VLAD algorithm produces a compact "signature" for each database image and query image based on [149] [150]: *(i)* the local aggregation of visual features into clusters using K-means clustering; *(ii)* compaction of the aggregated feature vectors into a vector of integers based on normalization, projection and quantization. In order to perform a retrieval task, we compute the inner product between the compacted feature vector of a query image and the compacted feature vector of each of the images in the database. The images corresponding to the top-T highest inner-product values are subsequently returned as the T best matches for the given query. Given that multiple query images (a.k.a. image "bunch") are matched through the stored database

at any given moment (e.g., because of many concurrent users, or due to the use of video that results in multiple feature vectors per query), the matching operations are carried out via GEMM products between the feature vectors of query image bunches and the database images. Following the GEMM, only inner-product values above a predetermined threshold are retained and a sorting algorithm is used in order to find the top-T matches [151]. This post-processing stage has negligible computational cost, thereby leaving the GEMM as the compute-intensive operation being carried out in the cloud computing cluster (the preprocessing stage for the VLAD signature extraction is performed offline for the database and on a local core for the query images). In our experiments, we use the VLAD descriptors derived from the INRIA Holidays dataset of [9, 149] comprising 1,491 holiday images, together with subsets selected from an additional 110,700 so-called "distractor" images from the INRIA website `https://lear.inrialpes.fr/~jegou/data.php` . Prior to our test, each database image was preprocessed to derive the 8,192-length VLAD signature vector of integers.

3.4.2 System Description: StarCluster Comprising AWS EC2 Instances

We examine the cost implication of running the VLAD image retrieval algorithm using a five-instance AWS EC2 cluster based on MIT's open source StarCluster toolkit [120] with the MPICH2 plugin. Each instance is a quadcore `m3.xlarge` instance type running Ubuntu 12.04.2. In order

to ensure cluster stability, StarCluster imposes that the master instance is setup as an on-demand instance type, while slave instances can either be spot or on-demand instance types. Unlike the case of the shared-memory cluster of Section 3.3 where the image database is in commonly-accessible memory, in this case the image database is equally spread over the four available slave instances, with the exception of the checksum-based approach, where the database is equally spread across three slave instances, with the fourth instance reserved for storing and computing with checksum data.

Firstly, the slave instances of the failure-intolerant implementation are set to run on on-demand instance types with the set price of $0.266 per hour for the `m3.xlarge` instance type [152], thereby ensuring no service interruption albeit at a high instance cost. On the other hand, given that the proposed method and the checksum-based implementation of the image retrieval experiment can tolerate instance failures, they run on AWS EC2 spot instances with spot bidding price set the same as for the on-demand instances ($0.266 per hour). We monitor the actual expenditure for spot instances by monitoring the evolution of the spot price via the AWS command line interface (CLI). Although AWS EC2 spot instances are known to be about 60% cheaper than the corresponding on-demand instances, the spot price occasionally spikes above a user's spot bidding price (i.e., above $0.266 in our case), which results in the termination of the spot instances [153].

Fig. 3.3 shows the spot price history[7] corresponding to the week of our ex-

[7]We pulled the exact spot prices by running the command: `aws`

Figure 3.3: AWS EC2 m3.xlarge spot instance type pricing history for 06/Dec/2015–13/Dec/2015 [availability-zone: us-east-1e, product-description: Linux/UNIX (AWS VPC)].

periment. We superimpose on the figure a predetermined "safety threshold" of $0.100: when spot prices surpass this threshold, a spot price spike is impending and our system is designed to react (as elaborated in the next subsection) before the spot price reaches our $0.266 bid. For the seven-day spot instance history of Fig. 3.3, the minimum, maximum and average spot prices were found to be $0.034, $0.500 and $0.058 per hour, respectively.

`ec2 describe-spot-price-history --instance-types m3.xlarge --start-time 2015-12-13T07:08:09 --end-time 2015-12-20T07:08:09 --availability-zone us-east-1e --product-description "Linux/UNIX (Amazon VPC)"` on the AWS CLI [154].

3.4.3 Experiment Description and Results

Following this setup and the spot history of Fig. 3.3, every time the spot price rose to the safety threshold, three slave instances were migrated to on-demand instances (one at a time[8]) and the fourth instance was simply terminated. Thus, we run three slave instances at the on-demand price for one hour (since billing is carried out in hourly installments). If after one hour the spot price had dropped below the safety threshold, all four slave instances were brought back to spot instances and the on-demand instances were terminated. In this way, within the week reported in Fig. 3.3, there were 236 timestamps when the spot prices exceeded the safety threshold of $0.100, and they corresponded to 18 hours of on-demand instance usage (i.e., on average, 2 hours 34 minutes per day). Therefore, the total cost of running the failure-tolerant algorithms for the seven-day period (168 hours) is given by:

$$18 \times 3 \times \$0.266 + (168 - 18) \times 4 \times \$0.052 = \$45.564$$

where $0.052 corresponds to the mean of spot prices below the safety threshold for the seven-day period.

On the other hand, the failure-intolerant algorithm run at a flat rate of $0.266-per-instance-per-hour, amounting to $168 \times 4 \times \$0.266 = \$178.752$ for the seven-day time period. Table 3.4 shows the cost savings for the VLAD image retrieval experiment per million images processed, under different

[8] we only need three slave instances in the proposed approach since the results of all four can be derived from three instances if we know no failures are bound to occur (which is the case when switching from spot to on-demand)

Table 3.4: Cost per million image queries (all in US dollar cents) and percentile difference in comparison to the conventional failure-intolerant processing for recovery after a quadcore instance failure in VLAD-based image retrieval.

Method / Database size	Conventional Failure-intolerant (×0.01$)	Proposed Failure-tolerant (×0.01$)	Checksum Failure-tolerant (×0.01$)
4992	0.29	0.09 (-68.88%)	0.11 (-61.25%)
11520	0.30	0.09 (-69.15%)	0.11 (-62.22%)
17472	0.37	0.10 (-71.78%)	0.13 (-64.10%)
82944	0.41	0.11 (-72.57%)	0.14 (-65.95%)
112128	0.59	0.16 (-73.66%)	0.19 (-68.64%)

database sizes.

The results of Table 3.4 show that, as expected, with increased database size: *(i)* the cost of all approaches per million image queries is increasing; *(ii)* the cost reduction percentage of both approaches exhibits is also increasing. Overall, numerical packing allows for 16%–24% reduction of cost in comparison to the checksum-based method and offers, on average, 71.21% cost reduction (i.e., almost 3.5 times less cost) in comparison to the failure-intolerant realization.

3.5 Conclusions

We propose a novel method for fail-stop failure mitigation in sum-of-product computations performed in multicore cloud computing platforms, with particular emphasis on integer matrix products and integer convolution/cross-correlation. Our approach inserts redundancy within the numerical representation of the inputs themselves by exploiting the concept of nu-

merical packing. Therefore, our method does not perform any redundant GEMM/CONV computations (akin to checksums) in order to mitigate processing unit failures. We show theoretically and experimentally that this results in significantly-lower overhead in comparison to the equivalent checksum-based failure mitigation method. Importantly, our approach achieves peak performance results that approach the conventional failure-intolerant computation as the matrix or signal dimensions increase, since the overhead of the required pre- and post-processing diminishes to zero. A deployment of the proposed approach over Amazon Web Services Elastic Compute Cloud (AWS EC2) spot instances that provide for cost savings (but require the mitigation of instance terminations) shows that the proposed method incurs nearly 3.5 times less cost than failure-intolerant processing.

Chapter 4

Numerical Entanglement for Multiple Core Failure Mitigation.

In the previous chapter, we presented numerical packing for the mitigation of single processing unit (PU) failure, based on an information redundancy technique that differs from the established method of checksum data generation, storage and processing. As discussed in Section 3.2, the limitation of the proposed method of Chapter 3 is the inability to recover lost data when multiple PU failures occur within a set of independent PUs designed for the mitigation of a single fail-stop failure. This stems from the limited number of data elements that can be packed within a given number representation, in order to support the dynamic range requirements of practical applications. Numerical entanglement as proposed by Anam and Andreopoulos [44], overcomes this packing prob-

lem by introducing the concept of number overlap (entanglement) within a packed representation. Numerical entanglement maintains the encoding relationship through linear and sesquilinear (LS) operations while providing offline data recovery in the event of processing unit (PU) failures or SDC mitigation in the absence of failures.

Numerical entanglement for the mitigation of fail-stop failures and/or SDCs within LS operations on L integer data streams ($L \geq 3$) linearly superimposes the input streams to form L *numerically entangled* integer data streams. These entangled streams are then stored in-place of the original inputs such that an arbitrary number of LS operations can be performed using these entangled data streams. The output results can be extracted from any $(L - F)$ entangled output streams by additions and arithmetic shifts, thereby mitigating F fail-stop failures ($F \leq \left\lfloor \frac{L-1}{2} \right\rfloor$), or detecting up to F SDCs per L-tuple of outputs at corresponding in-stream locations. Therefore, unlike other methods, the number of operations required for the entanglement, extraction and recovery of the results is linearly related to the number of the inputs and does not depend on the complexity of the performed LS operations.

In the following section, we present an overview of numerical entanglement for single fail-stop failure mitigation as proposed in [44] as well as highlighting its similarities and differences to numerical packing. We then continue in subsequent sections to propose enhancements to the traditional numerical entanglement algorithm for multiple failure mitigation and for SDC detection/correction. Our proposal is validated within an on-demand Amazon EC2 instance (Haswell architecture with AVX2

support) via integer matrix product operations. Our analysis and experiments for fail-stop failure mitigation and SDC detection reveal that the proposed approach incurs 0.65% to 37.23% reduction in processing throughput in comparison to the equivalent error-intolerant processing. This overhead is found to be up to two orders of magnitude smaller than that of the equivalent checksum-based method, with increased gains offered as the complexity of the performed LS operations is increasing. Therefore, our proposal can be used in distributed systems, unreliable multicore clusters and safety-critical applications, where robustness against failures and SDCs is a necessity.

4.1 Overview of Numerical Entanglement

Given L integer inputs $\mathbf{A}_l$, $(L \geq 3;\ \mathbf{A}_l \in \mathbb{R}^{M \times N};\ \forall l:\ 0 \leq l < L)$, our aim is to independently perform any linear or sesquilinear (LS) operation[1] on all L inputs using an $N \times K$ integer processing kernel $\mathbf{B}$, with only $(L-1)$ data groups sufficient for producing all L outputs, $\mathbf{R}_l$. For example, consider a $1 \times N$-by-$N \times N$ vector-matrix-multiplication (VMM) that must be performed on L input data streams, $\mathbf{a}_l$ $(0 \leq l < L)$ using a kernel matrix $\mathbf{B}$ to yield:

$$\mathbf{r}_l = \mathbf{a}_l \cdot \mathbf{B} \tag{4.1}$$

with each computation performed on a different PU in an $L-$ PU computing cluster. We describe below the traditional numerical entanglement

[1]Example of such LS operations include element-by-element addition/subtraction/-multiplication, inner/outer product and circular convolution or cross-correlation.

construct for recovering all data outputs when a single PU ($F = 1$) fail to return results after a deadline or in the event of a failure. To achieve this, input data streams $\mathbf{a}_l$ are pre-processed/encoded to form a set of entangled data streams, $\widehat{\mathbf{a}}_l$, which are stored in place of the plain inputs. The concurrent VMM of (4.1) is subsequently performed using the entangled inputs and the plain kernel matrix $\mathbf{B}$, to produce a set of entangled outputs, $\widehat{\mathbf{r}}_l$. We describe in the subsections that follow, the encoding (i.e. entanglement) algorithm that allows for the requirement of only $L - 1$ of the entangled data outputs for decoding all L outputs, $\widetilde{\mathbf{r}}_l$.

4.1.1 Input Data Entanglement for linear processing

The entangled input data streams $\widehat{\mathbf{a}}_0, \ldots, \widehat{\mathbf{a}}_{L-1}$ are generated by packing pairs from the L input streams while allowing for stream overlap via arithmetic shift and addition operations by ($0 \leq l < L$):

$$\widehat{\mathbf{a}}_l \quad = \quad \mathcal{S}_k \left\{ \mathbf{a}_l \right\} + \mathbf{a}_{l \bmod L} \tag{4.2}$$

where k is an object within an entanglement parameter pair, $\{k, s\}$ and $\mathcal{S}_k \left\{ \mathbf{a}_l \right\}$ is the element-wise left bit shift operator defined in the notational conventions table of pg. xix. The entanglement parameter pair, $\{k, s\}$ controls the output dynamic range of output data while preventing integer overflow. Importantly, numerical entanglement guarantees correct output data recovery provided that:

- maximum bit representation of data outputs does not exceed

$(L-2)k+s$ bits and,

- the entanglement parameters $\{k, s\}$ are bounded by:

$$(L-1)\cdot k + s \leq w \tag{4.3}$$

with w the bit-width of the utilized integer representation.

The entanglement parameters $\{k, s\}$ are chosen in such a way that (4.3) is maximized with $s \leq k$. Thus for 32–bit integer representation with $w = 32$ and for a parallel VMM computation with $L = 3$ and $L = 30$, the optimal entanglement parameters are $\{k, s\} = \{11, 10\}$ and $\{1, 1\}$ respectively. Specifically, setting $\{k, s\} = \{11, 10\}$ for $L = 3$ allows for up to 21-bits of output dynamic range for data outputs. However, a maximum of 17-bits can only be supported if the entanglement parameters were set to $\{k, s\} = \{15, 2\}$.

Following the entanglement of (4.2), the VMM computation of (4.1) can be performed by $(0 \leq l < L)$:

$$\widehat{\mathbf{r}}_l = \widehat{\mathbf{a}}_l \cdot \mathbf{B} \tag{4.4}$$

A pictorial illustration of (4.4) is presented in Figure 4.1.

Conversely, the equivalent construct for failure mitigation using the symmetric numerical packing of Section 3.2 is illustrated in Figure 4.2. The major difference between the two algorithms is that, while the latter specifies a constant output dynamic range for all sets of $\{L, F\}$, the former provides a systematic method of increasing the output dynamic range of data elements as L increases for a given value of F. On the other hand,

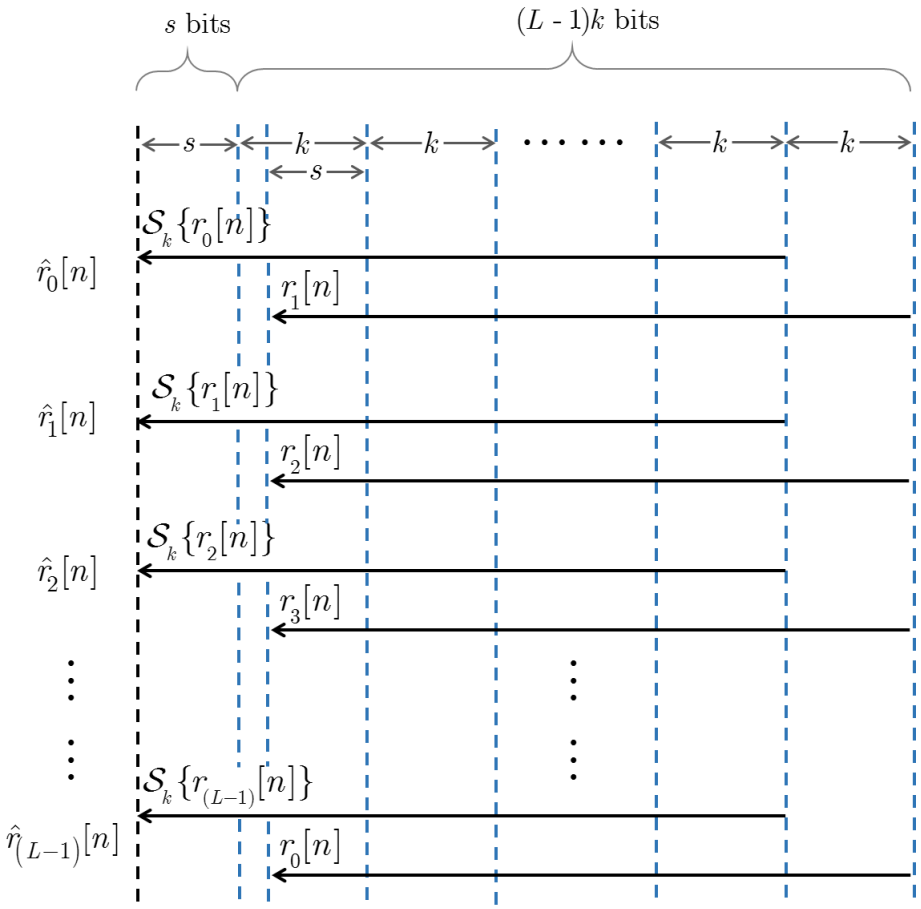


Figure 4.1: Illustration of L numerically entangled outputs after integer LS processing for the mitigation of $F = 1$ failures. The solid arrows indicate maximum bit-width for each output $r_{l,}[n]$ $0 \leq l < L$, $0 \leq n < N$.

because the encoding of numerical packing requires the production of a kernel matrix (or vector) that is half the size of the plain kernel, substantial execution time reduction can be achieved especially for compute-intensive algorithms. The major differences between numerical packing, numerical entanglement and other algorithm-based forward error recovery methods are summarized in Table 4.1.

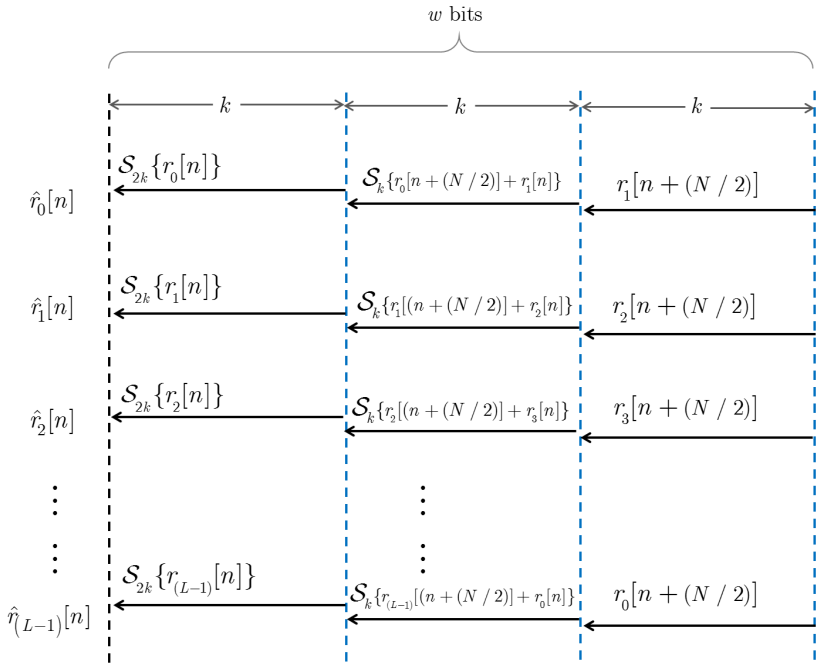


Figure 4.2: Illustration of L numerically packed outputs after integer LS processing for the mitigation of $F = 1$ failures. The solid arrows indicate the maximum attainable dynamic range of each output $r_l\,[n]$ $0 \leq l < L$.

4.1.2 Disentanglement/Failure Recovery

Following the computation of (4.1), let us assume that all but one computing PU return their processed outputs after a pre-established computation deadline. For such a case, we describe the recovery steps for all L processed data outputs, $\widetilde{\mathbf{r}}_l$ $0 \leq l < L$ for the VMM computation of (4.1). The following steps can be verified pictorially by Fig. 4.1.

Given the index of the PU with missing outputs, x, the first step of disentanglement requires the generation of a temporary vector, $\widehat{\mathbf{r}}_{\text{temp}}$, comprising the scaled sum of the available data streams such that:

$$\widehat{\mathbf{r}}_{\text{temp}} = \sum_{i=0}^{(L-2)} (-1)^i \cdot \mathcal{S}_{i\cdot k}\left\{\widehat{\mathbf{r}}_{(x+L-1-i)\,mod\,L}\right\} \tag{4.5}$$

A useful characteristic of the vector of temporary data elements of (4.5) is the disentanglement of the outputs such that each representation holds two distinguishable values in "packed" form i.e., without overlap as shown in Figs. 4.4(b) and 4.5(b). Therefore, after the computation of (4.5), $\widehat{\mathbf{r}}_{\text{temp}}$ is composed of $\mathbf{r}_x$ and $\mathbf{r}_{(x+1)\,mod\,L}$ in the lower and upper bit regions respectively. We can then recover $\widetilde{\mathbf{r}}_x$ and $\widetilde{\mathbf{r}}_{(x+1)\,mod\,L}$ located at the $(L-1)\,k$ least significant and $(L-2)\,k+s$ most significant bits of (4.5) respectively by:

$$\widetilde{\mathbf{r}}_x = \mathcal{S}_{-(2w-(L-1)\cdot k)}\left\{\mathcal{S}_{(2w-(L-1)\cdot k)}\left\{\widehat{\mathbf{r}}_{\text{temp}}\right\}\right\} \tag{4.6}$$

$$\widetilde{\mathbf{r}}_{(x+1)\,mod\,L} = \mathcal{S}_{-((L-1)\cdot k)}\left\{(-1)^L\cdot(\widehat{\mathbf{r}}_{\text{temp}}-\widetilde{\mathbf{r}}_x)\right\} \tag{4.7}$$

The remaining $L-2$ outputs can subsequently be recovered by $(1\leq i<L-1)$:

$$\widetilde{\mathbf{r}}_{(x+(i+1))\,mod\,L} = \widehat{\mathbf{r}}_{(x+i)\,mod\,L} - \mathcal{S}_k\left\{\widetilde{\mathbf{r}}_{(x+i)\,mod\,L}\right\} \tag{4.8}$$

Indeed the recovery of (4.5)–(4.8) shows that the output vector, $\widehat{\mathbf{r}}_x$ was not required to derive the entire set of L output vectors, $\widetilde{\mathbf{r}}_l$.

Note that to ensure accurate recovery of outputs using the disentanglement steps described in this section, (4.5) would require a double bit-width representation for its computation. However, the use of w-bits is also possible with slight modifications to the disentanglement steps.

We show in Listing 4.1 a sample C++ implementation (optimized using Intel's SSE intrinsic) for recovering all L data streams when no results are returned from the last computing core. Further optimization can be

achieved by employing parallel programming algorithms for both the pre- and post processing stages of numerical entanglement.

In the sections that follow, we now focus on the description of the proposed multi-stage/multi-tier entanglement design for the mitigation of multiple fail-stop failures. We also show that the proposed algorithm is capable of detecting and correcting SDCs without data re-computation.

4.2 Generalized Numerical Entanglement for Multiple Failure Recovery.

As described in Section 4.1, the single-tier numerical entanglement of Figure 4.1 can only mitigate a single fail-stop failure. Therefore, a systematic design of an $F-$tier numerical entanglement algorithm is necessary for the recovery of output data when F out of L parallel processing units fail to return processed outputs. In addition, we show that, beyond the recovery of F fail-stop failures, the proposed F-tier entanglement algorithm is capable of detecting F SDCs or detecting and correcting $F-1$ SDCs in output data elements when no failures occur. We begin by describing the simplest multi-tier algorithm for $\{L, F\} = \{5, 2\}$, i.e. the case of two fail stop failures. As before, one-dimensional data inputs are used for all derivations in this section for brevity of exposition as it is straightforward to generalize the proposal for LS matrix computations.

```
int k,s,num_shifts,w;
for (int n=0;n<N;n+=8)
{
        __m256i d_temp64_low= _mm256_setzero_si256();
        __m256i d_temp64_high= _mm256_setzero_si256();
        __m256i r_tilde64_low,r_tilde64_high;
        __m256i r_hat64_low,r_hat64_high, temp_r_tilde;
        __m128i temp_r_tilde32_high, temp_r_tilde32_low;
        __m128i temp_r_hat64_low,temp_r_hat64_high;

        num_shifts=1;
        for (int i=L-2; i>=0;i-=2)
        {
                /*load eight 32-bits outputs of core (i) and split
                into high and low parts in order to utilize 64-bit
                integer AVX intrinsiscs*/

                temp_r_hat64_low=_mm256_extracti128_si256(r_hat32[i][n],0);
                temp_r_hat64_high=_mm256_extracti128_si256(r_hat32[i][n+4],1);
                r_hat64_low =_mm256_cvtepi32_epi64 (temp_r_hat64_low);
                r_hat64_low=_mm256_slli_epi64(r_hat64_low, num_shifts*k);
                //add to d_temp
                d_temp_low = _mm256_add_epi64(d_temp_low,r_hat64_low);
                r_hat64_high= _mm256_cvtepi32_epi64 (temp_r_hat64_high);
                r_hat64_high=_mm256_slli_epi64(r_hat64_high, num_shifts*k);
                d_temp_high = _mm256_add_epi64(d_temp_high, r_hat64_high);

                /*load eight 32-bits outputs of core (i-1) and split
                into high and low parts in order to utilize 64-bit
                integer AVX intrinsiscs*/

                temp_r_hat64_low=_mm256_extracti128_si256(r_hat32[i-1][n],0);
                temp_r_hat64_high=_mm256_extracti128_si256(r_hat32[i-1][n+4],1);
                r_hat64_low =_mm256_cvtepi32_epi64 (temp_r_hat64_low);
                r_hat64_low=_mm256_slli_epi64(r_hat64_low, num_shifts*k);
                //subtract from d_temp
                d_temp_low = _mm256_sub_epi64(d_temp_low,r_hat64_low);
                r_hat64_high= _mm256_cvtepi32_epi64 (temp_r_hat64_high);
```

```
            r_hat64_high=_mm256_slli_epi64(r_hat64_high, num_shifts*k);
            d_temp_high = _mm256_sub_epi64(d_temp_high, r_hat64_high);
            num_shifts+=2;
        }
        d_temp_low = _mm256_slli_epi64(d_temp_low, 2*(w-k));
        r_tilde64_low= _mm256_srai_epi64(d_temp_low,2*(w-k));
        d_temp_high = _mm256_slli_epi64(d_temp_high, 2*(w-k));
        r_tilde64_high= _mm256_srai_epi64(d_temp_high,2*(w-k));

        //Convert 64-bit recovered outputs, r_tilde to 32-bit outputs
        temp_r_tilde32_low= _mm256_cvtepi64_epi32(r_tilde64_low);
        temp_r_tilde32_high= _mm256_cvtepi64_epi32(r_tilde64_high);

        //actual output for core(L-2)
        r_tilde32[L-2][n] = _mm256_set_m128i(temp_r_tilde32_high,temp_r_tilde32_low
            );

        //compute output for core(L-1) using r_tilde32 of core(L-2)
        r_tilde64_low=_mm256_sub_epi64(r_tilde64_low,d_temp);
        r_tilde64_low= _mm256_srai_epi64(r_tilde64_low,2*k);
        r_tilde64_high=_mm256_sub_epi64(r_tilde64_high,d_temp2);
        r_tilde64_high= _mm256_srai_epi64(r_tilde64_high,2*k);

        //Convert 64-bit recovered outputs, r_tilde to 32-bit outputs
        temp_r_tilde32_low= _mm256_cvtepi64_epi32(r_tilde64_low);
        temp_r_tilde32_high= _mm256_cvtepi64_epi32(r_tilde64_high);
        //actual output for core(L-1)
        r_tilde32[L-1][n] = _mm256_set_m128i(temp_r_tilde32_high,temp_r_tilde32_low
            );

        //compute output for remaining L-2 cores
        for (int j=0; j<L-2;j++)
        {
            temp_r_tilde = _mm256_slli_epi32(r_tilde32[(j+L-1)%L][n], k);
            r_tilde32[j][n] = _mm256_sub_epi32(r_hat32[j][n],temp_r_tilde);
        }
}
```

Listing 4.1: Recovery of L data streams using $L-1$ entangled streams

Table 4.1: Summary of features of different methods for K-failure mitigation within L streams under a w-bit representation.

Method / Feature	Checksum-based FER [96,97] [101,113,155]	Dual Modular Redundancy [95]	Numerical Packing [156]	Numerical Entanglement [44,157]
In-place storage	No	No	Yes	Yes
% of redundant LS computations	$\frac{F}{L} \times 100\%$	100%	0%	0%
Reduction of output bitwidth supported	$F = 1$: $\lceil \log_2 L \rceil$ $F > 1$: see Table 4.2	0 bits	$F = 1$: 12-bits $F > 1$: NA	$F = 1$: $\lceil \frac{w}{L} \rceil$ $F > 1$: see Table 4.2
Failure Mitigation	$F \leq \lfloor \frac{L-1}{2} \rfloor$ failures in $L + F$ streams, $L \geq 3$	$F = 1$ failure in $L = 2$ streams	$F = 1$ failure in $L \geq 3$ streams	$F \leq \lfloor \frac{L-1}{2} \rfloor$ failures in L streams, $L \geq 3$
Practical execution overhead	More than $\frac{F}{L} \times 100\%$	More than 100%	$\approx \left(\frac{7}{2N-1}\right) \times 100\%$ for GEMM. (decreases with increased operand length)	0.03% to 37.23% for GEMM. (decreases with increased operand length.)

4.2.1 Generalized Numerical Entanglement in Groups of Five Inputs $(L = 5, F = 2)$

In this subsection, we illustrate the basic case of mitigation of two fail-stop failures in five input streams, detection of two SDCs, or the detection and correction of one SDC within the quintuple of outputs at any position n $(0 \leq n < N)$, with N the length of the output vectors.

Entanglement

In the two-tier entanglement with $L = 5$, the entangled data elements derived from the plain inputs, $\mathbf{a}_0, \ldots, \mathbf{a}_4$, is given by $(\forall l : 0 \leq l < 5)$:

$$\widehat{\mathbf{a}}_l^{(1)} = \mathcal{S}_{k_1} \{\mathbf{a}_l\} + \mathbf{a}_{l \bmod L} \tag{4.9}$$

and

$$\widehat{\mathbf{a}}_l^{(2)} = \mathcal{S}_{k_2}\left\{\widehat{\mathbf{a}}_l^{(1)}\right\} + \widehat{\mathbf{a}}_{l \bmod L}^{(1)} \tag{4.10}$$

with

$$3k_1 + s_1 \leq k_2 + s_2 \tag{4.11}$$

$$2k_2 + s_2 \leq w. \tag{4.12}$$

The maximum bit-width supported for each output element is upper bounded by $2k_1 + s_1$ bits.

The values for k_2 and s_2 are chosen such that $2k_2 + s_2$ is maximum within the constraint of (4.12) and $s_2 \leq k_2$. Similarly, the values for k_1 and s_1 are chosen such that $3k_1 + s_1$ is maximum within the constraint of (4.11) and $s_1 \leq k_1$. Via the application of LS operations on the entangled input data streams, $\widehat{\mathbf{a}}_l^{(2)}$ using a kernel vector $\mathbf{b}$, we obtain the entangled outputs $\widehat{\mathbf{r}}_l^{(2)}$with each output stream comprising of N_{out} data elements. A conceptual illustration of the entangled outputs after an LS operation is given in Fig. 4.3. From the illustration presented, it is straight forward to observe that the proposed two-tier entanglement algorithm achieves fault tolerance by sacrificing $(k_1 + k_2)$-bits out of the available w-bits utilized for conventional computation. As a practical instantiation of (4.11) and (4.12), for $w = 32$, the optimal entanglement parameters for processing signed integer data is given by: $k_1 = 6$, $k_2 = 11$, $s_1 = 3$ and $s_2 = 10$.

We now describe the disentanglement and recovery process in case of any two failures. The reader can also consult Fig. 4.3 in order to verify the results of all the presented steps.

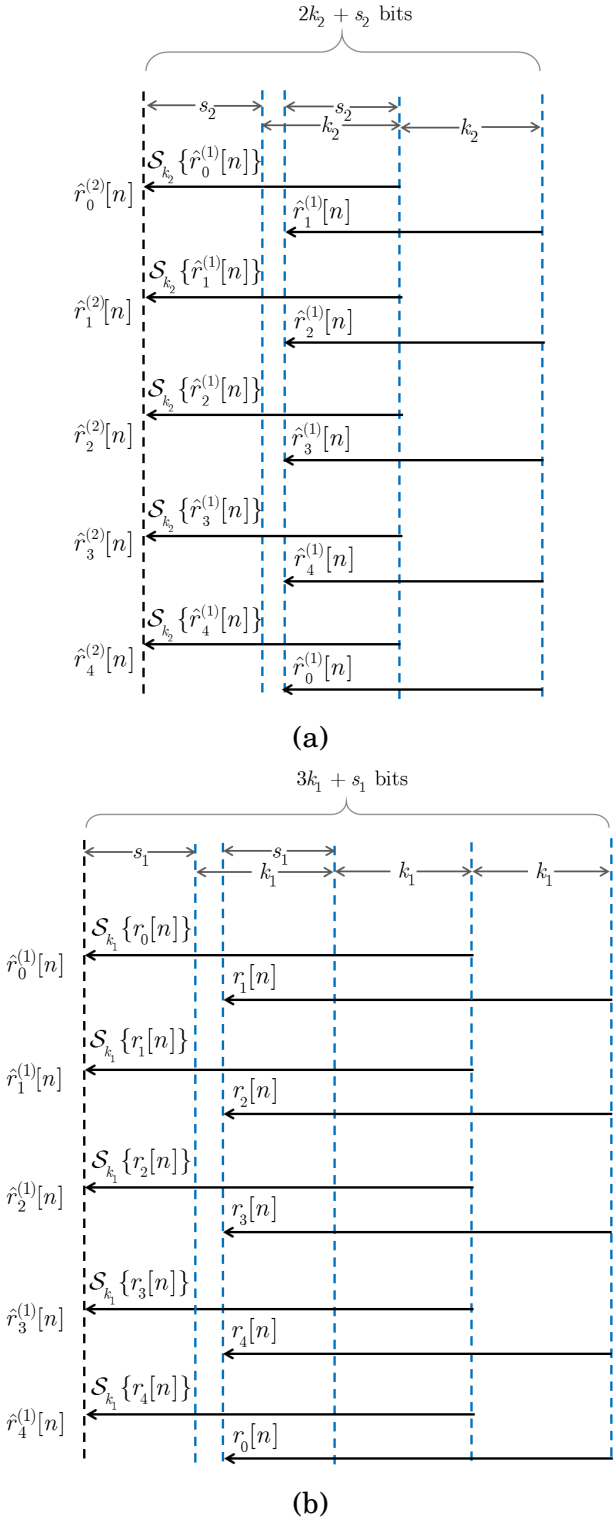


Figure 4.3: Illustration of entanglement of five outputs after integer linear processing: (a) second tier of superimposed outputs; (b) first tier of superimposed outputs, showing the original output values $r_0[n], \ldots, r_4[n]$ that are entangled within, $0 \leq n < N_{\text{out}}$.

Disentanglement

The proposed method mitigates $F \in \{1, 2\}$ failures. Equivalently, if all L entangled output streams are obtained, it can detect up to two SDCs and can also detect and correct any single SDC within any quintuple of outputs $\left[\widetilde{r}_0^{(2)}[n] \quad \ldots \quad \widetilde{r}_4^{(2)}[n]\right]^{\mathrm{T}}$.

For the case of loss of a single stream $\widehat{\mathbf{r}}_x^{(2)}$, $0 \leq x < L$, due to a single fail-stop failure, we recover the outputs of the first tier by:

$$\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}}^{(1)} &= \widehat{\mathbf{r}}_{(x+2)\text{mod}5}^{(2)} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}_{(x+1)\text{mod}5}^{(2)}\right\} \\
\widehat{\mathbf{r}}_{(x+3)\text{mod}5}^{(1)} &= \mathcal{S}_{-2(w-k_2)}\left\{\mathcal{S}_{2(w-k_2)}\left\{\widehat{\mathbf{r}}_{\text{temp}}^{(1)}\right\}\right\} \\
\widehat{\mathbf{r}}_{(x+1)\text{mod}5}^{(1)} &= \mathcal{S}_{-2k_2}\left\{-\left(\widehat{\mathbf{r}}_{\text{temp}}^{(1)} - \widehat{\mathbf{r}}_{(x+3)\text{mod}5}^{(1)}\right)\right\} \\
\widehat{\mathbf{r}}_{(x+2)\text{mod}5}^{(1)} &= \widehat{\mathbf{r}}_{(x+1)\text{mod}5}^{(2)} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}_{(x+1)\text{mod}5}^{(1)}\right\} \\
\widehat{\mathbf{r}}_{(x+4)\text{mod}5}^{(1)} &= \widehat{\mathbf{r}}_{(x+3)\text{mod}5}^{(2)} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}_{(x+3)\text{mod}5}^{(1)}\right\} \\
\widehat{\mathbf{r}}_{x}^{(1)} &= \widehat{\mathbf{r}}_{(x+4)\text{mod}5}^{(2)} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}_{(x+4)\text{mod}5}^{(1)}\right\}
\end{aligned} \tag{4.13}$$

On the next stage, we recover the final outputs by:

$$\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}} &= \widehat{\mathbf{r}}_4^{(1)} - \mathcal{S}_{k_1}\left\{\widehat{\mathbf{r}}_3^{(1)}\right\} + \mathcal{S}_{2k_1}\left\{\widehat{\mathbf{r}}_2^{(1)}\right\} \\
\widetilde{\mathbf{r}}_0 &= \mathcal{S}_{-(2w-3k_1)}\left\{\mathcal{S}_{(2w-3k_1)}\left\{\widehat{\mathbf{r}}_{\text{temp}}\right\}\right\} \\
\widetilde{\mathbf{r}}_1 &= \widehat{\mathbf{r}}_0^{(1)} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_0\right\} \\
\widetilde{\mathbf{r}}_2 &= \widehat{\mathbf{r}}_1^{(1)} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_1\right\} \\
\widetilde{\mathbf{r}}_3 &= \widehat{\mathbf{r}}_2^{(1)} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_2\right\} \\
\widetilde{\mathbf{r}}_4 &= \widehat{\mathbf{r}}_3^{(1)} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_3\right\}
\end{aligned} \tag{4.14}$$

Explanation of (4.13) and (4.14)—see also Fig. 4.3: The first part of (4.13) creates a composite number comprising $\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5}$ in the l_2+k_2 most-significant bits and $\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5}$ in the $2l_2$ least-significant bits (therefore, $\widehat{\mathbf{r}}^{(1)}_{\text{temp}}$ requires $3l_2+k_2$ bits). In the second part, $\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5}$ is extracted by: *(i)* discarding the $(2w-2l_2)$ most-significant bits; *(ii)* arithmetically shifting the output down to the correct range. The third part of (4.13) uses $\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5}$ to recover $\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5}$ and the last three parts of (4.13) use: *(i)* $\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5}$ to recover $\widehat{\mathbf{r}}^{(1)}_{(x+2)\bmod 5}$, *(ii)* $\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5}$ to recover $\widehat{\mathbf{r}}^{(1)}_{(x+4)\bmod 5}$ and, finally, *(iii)* $\widehat{\mathbf{r}}^{(1)}_{(x+4)\bmod 5}$ to recover $\widehat{\mathbf{r}}^{(1)}_{x}$. Finally, having recovered all $\widehat{\mathbf{r}}^{(1)}_0,\ldots,\widehat{\mathbf{r}}^{(1)}_4$, via (4.14), we recover the final outputs $\widetilde{\mathbf{r}}_0,\ldots,\widetilde{\mathbf{r}}_4$.

For the case of two fail-stop failures, $\widehat{\mathbf{r}}^{(2)}_{x_1}$ and $\widehat{\mathbf{r}}^{(2)}_{x_2}$ with $0\le x_1,x_2<L$ and $x_2>x_1$, we first define the stream index x based on the distance $x_{\text{diff}}=x_2-x_1$ between the two streams. Specifically, we set:

$$x\equiv\begin{cases}x_2, & \text{if } x_{\text{diff}}<3\\ x_1, & \text{if } x_{\text{diff}}\ge 3\end{cases} \tag{4.15}$$

If $x_{\text{diff}}=1$, we can disentangle the tier-one outputs by:

$$\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp}} &= \widehat{\mathbf{r}}^{(2)}_{(x+2)\bmod 5}-\mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(2)}_{(x+1)\bmod 5}\right\}\\
\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5} &= \mathcal{S}_{-2(w-k_2)}\left\{\mathcal{S}_{2(w-k_2)}\left\{\widehat{\mathbf{r}}^{(1)}_{\text{temp}}\right\}\right\}\\
\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5} &= \mathcal{S}_{-2k_2}\left\{-\left(\widehat{\mathbf{r}}^{(1)}_{\text{temp}}-\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5}\right)\right\}\\
\widehat{\mathbf{r}}^{(1)}_{(x+2)\bmod 5} &= \widehat{\mathbf{r}}^{(2)}_{(x+1)\bmod 5}-\mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5}\right\}\\
\widehat{\mathbf{r}}^{(1)}_{(x+4)\bmod 5} &= \widehat{\mathbf{r}}^{(2)}_{(x+3)\bmod 5}-\mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5}\right\}
\end{aligned} \tag{4.16}$$

and, on the next stage,

$$\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}} &= \widehat{\mathbf{r}}^{(1)}_{(x+3)\text{mod}5} - \mathcal{S}_{k_1}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+2)\text{mod}5}\right\} \\
&+ \quad \mathcal{S}_{2k_1}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+1)\text{mod}5}\right\} \\
\widetilde{\mathbf{r}}_{(x+4)\text{mod}5} &= \mathcal{S}_{-(2w-3k_1)}\left\{\mathcal{S}_{(2w-3k_1)}\left\{\widehat{\mathbf{r}}_{\text{temp}}\right\}\right\} \\
\widetilde{\mathbf{r}}_{(x+1)\text{mod}5} &= \mathcal{S}_{-3k_1}\left\{\widehat{\mathbf{r}}_{\text{temp}} - \widetilde{\mathbf{r}}_{(x+4)\text{mod}5}\right\} \\
\widetilde{\mathbf{r}}_{x} &= \widehat{\mathbf{r}}^{(1)}_{(x+4)\text{mod}5} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{(x+4)\text{mod}5}\right\} \\
\widetilde{\mathbf{r}}_{(x+2)\text{mod}5} &= \widehat{\mathbf{r}}^{(1)}_{(x+1)\text{mod}5} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{(x+1)\text{mod}5}\right\} \\
\widetilde{\mathbf{r}}_{(x+3)\text{mod}5} &= \widehat{\mathbf{r}}^{(1)}_{(x+2)\text{mod}5} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{(x+2)\text{mod}5}\right\}
\end{aligned} \tag{4.17}$$

If $x_{\text{diff}} = 2$, we can disentangle the outputs by:

$$\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp}} &= \widehat{\mathbf{r}}^{(2)}_{(x+2)\text{mod}5} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(2)}_{(x+1)\text{mod}5}\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+3)\text{mod}5} &= \mathcal{S}_{-2(w-k_2)}\left\{\mathcal{S}_{2(w-k_2)}\left\{\widehat{\mathbf{r}}^{(1)}_{\text{temp}}\right\}\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+1)\text{mod}5} &= \mathcal{S}_{-2k_2}\left\{-\left(\widehat{\mathbf{r}}^{(1)}_{\text{temp}} - \widehat{\mathbf{r}}^{(1)}_{(x+3)\text{mod}5}\right)\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+2)\text{mod}5} &= \widehat{\mathbf{r}}^{(2)}_{(x+1)\text{mod}5} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+1)\text{mod}5}\right\}
\end{aligned} \tag{4.18}$$

and on the next stage

$$
\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}} &= \widehat{\mathbf{r}}^{(1)}_{(x+3)\bmod 5} - \mathcal{S}_{k_1}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+2)\bmod 5}\right\} \\
&+ \mathcal{S}_{2k_1}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5}\right\} \\
\widetilde{\mathbf{r}}_{(x+4)\bmod 5} &= \mathcal{S}_{-(2w-3k_1)}\left\{\mathcal{S}_{(2w-3k_1)}\left\{\widehat{\mathbf{r}}_{\text{temp}}\right\}\right\} \\
\widetilde{\mathbf{r}}_{(x+1)\bmod 5} &= \mathcal{S}_{-3k_1}\left\{\widehat{\mathbf{r}}_{\text{temp}} - \widetilde{\mathbf{r}}_{(x+4)\bmod 5}\right\} \\
\widetilde{\mathbf{r}}_{(x+2)\bmod 5} &= \widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod 5} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{(x+1)\bmod 5}\right\} \\
\widetilde{\mathbf{r}}_{(x+3)\bmod 5} &= \widehat{\mathbf{r}}^{(1)}_{(x+2)\bmod 5} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{(x+2)\bmod 5}\right\} \\
\widetilde{\mathbf{r}}_{x} &= \left(\widehat{\mathbf{r}}^{(2)}_{(x+4)\bmod 5} - \mathcal{S}_{(k_1+k_2)}\left\{\widetilde{\mathbf{r}}_{(x+4)\bmod 5}\right\}\right. \\
&- \left.\widetilde{\mathbf{r}}_{(x+1)\bmod 5}\right) \times \left(2^{k_1} + 2^{k_2}\right)^{-1}
\end{aligned}
\tag{4.19}
$$

The last part of (4.19) can be implemented without element-wise multiplication with the inverse factor of $(2^{k_1} + 2^{k_2})$ if one additional temporary variable, one addition and two arithmetic shifts are used.

Finally, if $x_{\text{diff}} \in \{3, 4\}$, disentanglement and recovery of all outputs $\widetilde{\mathbf{r}}_0, \ldots, \widetilde{\mathbf{r}}_4$ follows the steps for $x_{\text{diff}} \in \{2, 1\}$ respectively , albeit with $x \equiv x_1$ as per (4.15). We conclude the presentation for the case of $(L, F) = (5, 2)$ with some remarks relating to various implementation aspects of our approach.

Remark 1 (operations within w bits): To facilitate our exposition, the first three parts of (4.13), the first two parts of (4.14), and the first three parts of (4.16)–(4.19) are presented under the assumption of a $2w$-bit integer representation since $\widehat{\mathbf{r}}^{(1)}_{\text{temp}}$ and $\widehat{\mathbf{r}}_{\text{temp}}$ require $2w$ bits.

Remark 2 (recovery without the use of $F \in \{1, 2\}$ streams): Notice that, for the case of a single fail-stop failure, (4.13) does not use stream(s) $\widehat{\mathbf{r}}^{(2)}_x$.

This is a crucial element of our approach: since streams $\widetilde{\mathbf{r}}_0, \ldots, \widetilde{\mathbf{r}}_4$ were derived without using $\widehat{\mathbf{r}}_x^{(2)}$, full recovery of all outputs takes place even with the loss of one entangled stream. Similarly, for the case of two failures, (4.16) and (4.18) do not use streams $\widehat{\mathbf{r}}_{x_1}^{(2)}$ and $\widehat{\mathbf{r}}_{x_2}^{(2)}$ for the recovery of $\widetilde{\mathbf{r}}_0, \ldots, \widetilde{\mathbf{r}}_4$.

Remark 3 (detection of two SDCs and correction of any single SDC within $\left[\widehat{r}_0^{(2)}[n] \quad \ldots \quad \widehat{r}_4^{(2)}[n]\right]^{\mathrm{T}}$*):* Given that we can recover all outputs from three entangled streams, to detect and correct any single SDC within any quintuple $\left[\widehat{r}_0^{(2)}[n] \quad \ldots \quad \widehat{r}_4^{(2)}[n]\right]^{\mathrm{T}}$ ($\forall n : 0 \leq n < N_{\text{out}}$), we can recover five versions of all outputs $\left[\widetilde{r}_0[n] \quad \ldots \quad \widetilde{r}_4[n]\right]^{\mathrm{T}}$ using ($\forall x \in [0,4]$):

$$\{\widetilde{r}_0[n], \ldots, \widetilde{r}_4[n]\}^{(x)} \xleftarrow[\text{via ((4.16))-((4.19))}]{\text{disentangle}} \left\{\widehat{r}_x^{(2)}[n], \widehat{r}_{(x+1)\bmod 5}^{(2)}[n], \widehat{r}_{(x+2)\bmod 5}^{(2)}[n]\right\}. \tag{4.20}$$

The pattern between the agreed results will show the stream number where the SDC occurred, and the recovery can retain the results that do not stem from that stream. For example, if $\widehat{r}_0^{(2)}[n]$ is erroneous due to an SDC, then the disentanglements of (4.20) corresponding to $x \in \{0,3,4\}$ will not agree with the ones of $x \in \{1,2\}$, which demonstrates than an SDC occurred in $\widehat{r}_0^{(2)}[n]$ (similar for the other cases)—therefore, to mitigate the SDC occurrence, the results of the $x \in \{1,2\}$ disentanglements of (4.20) should be used. On the other hand, the occurrence of two SDCs within any quintuple $\left[\widehat{r}_0^{(2)}[n] \quad \ldots \quad \widehat{r}_4^{(2)}[n]\right]^{\mathrm{T}}$ will result in recovery of outputs that do not agree with each other for all x values after the disentanglements of (4.20). Therefore, when (4.20) returns nonidentical results for all values of x at index n, more than one SDC has occurred for the

index n, and correction can only be achieved by re-computation of all five outputs.

Remark 4 (dynamic range): Bit $2k_1 + s_1$ within each recovered output $\widetilde{r}_0[n], \ldots, \widetilde{r}_4[n]$ represents its sign bit. Given that: *(i)* each entangled output comprises the addition of two outputs (with one of them left-shifted by k_1 bits); *(ii)* the entangled outputs must not exceed $3k_1 + s_1$ bits, we conclude that the outputs of the linear operations must not exceed the range:

$$\forall n : \; r_0[n], \ldots, r_4[n] \in \left\{ -\left(2^{2k_1+s_1-1} - 2^{2k_1}\right), \, 2^{2k_1+s_1-1} - 2^{2k_1} \right\}. \tag{4.21}$$

with k_1 and s_1 defined within the constraints of (4.11) and (4.12).

4.2.2 Generalized Numerical Entanglement for Two Fail-Stop Failure Mitigating $(L \geq 5, \; F = 2)$

We now focus on generalizing numerical entanglement for mitigation F fail-stop failures with the condition that $F \leq \left\lfloor \frac{L-1}{2} \right\rfloor$. As a first step, we draw up a generalization for the case of $L \geq 5, \; F = 2$ in the following sets of equations and the description that follow.

The entanglement parameters, $\{k_1, s_1, k_2, s_2\}$ that control the dynamic range of data outputs are given by:

$$\begin{aligned} P k_1 + s_1 &\leq (P-2)\, k_2 + s_2 \\ (P-1)\, k_2 + s_2 &\leq w \end{aligned} \tag{4.22}$$

where $P = \lceil \frac{L}{2} \rceil$. From the expression of (4.22), we can deduce the number of "zones" (cf Fig. 4.3) required for constructing the tier-1 and tier-2 entanglements of the input data streams. For example, for $L = \{11, 12\}$, the first part of (4.22) indicates that there exists $P = 6$ zones of k_1-bits and one zone of s_1-bits for the tier-1 entanglement, while the second part of the equation shows that the tier-2 entanglement is constructed using $P - 1 = 5$ zones of k_2-bits and one zone of s_2-bits. Using the zones representation, it is also easy to deduce that the maximum bit-width that can be supported for all data elements within the performed LS operation is $(P - 1) \cdot k_1 + s_1$ bits. In addition, following the constraints of (4.22), the optimal entanglement parameters that will maximize the available bit-width can be derived. Therefore, for the example of $L = \{11, 12\}$ and for $w = 32$, the optimal parameters are given by: $\{k_1, s_2, k_2, s_2\}$=$\{4, 2, 6, 2\}$.

Entanglement follows the same pattern as (4.9) and (4.10) and LS operations can be performed on the entangled inputs with no additional/redundant resources for improved reliability.

We present a generalized algorithm for disentanglement for the two failure cases in this class, i.e., $F = \{1, 2\}$.

For the case of loss of a single stream $\widehat{\mathbf{r}}_x^{(2)}$, $0 \leq x < L$, due to a single

fail-stop failure, we recover the outputs of the first tier by $\left(P = \lceil \frac{L}{2} \rceil\right)$:

$$\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp}} &= \sum_{i=0}^{(P-2)} (-1)^i \cdot \left(\mathcal{S}_{i\cdot k_2}\left\{\widehat{\mathbf{r}}^{(2)}_{(x+P-1-i) \bmod L}\right\}\right) \\
\widehat{\mathbf{r}}^{(1)}_{(x+P) \bmod L} &= \mathcal{S}_{-(2w-(P-1)k_2)}\left\{\mathcal{S}_{2w-(P-1)k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{\text{temp}}\right\}\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+1) \bmod L} &= \mathcal{S}_{-(P-1)k_2}\left\{(-1)^P \cdot \left(\widehat{\mathbf{r}}^{(1)}_{\text{temp}} - \widehat{\mathbf{r}}^{(1)}_{(x+P) \bmod L}\right)\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+i+1) \bmod L} &= \widehat{\mathbf{r}}^{(2)}_{(x+i) \bmod L} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+i) \bmod L}\right\} \;\; \forall i : 1 \leq i < L,\; i \neq P-1
\end{aligned} \tag{4.23}$$

On the next stage, we recover the final outputs by:

$$\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}} &= \sum_{i=0}^{(P-1)} (-1)^i \cdot \left(\mathcal{S}_{i\cdot k_1}\left\{\widehat{\mathbf{r}}^{(1)}_{(L-i-1)}\right\}\right) \\
\widetilde{\mathbf{r}}_0 &= \mathcal{S}_{-(2w-Pk_1)}\left\{\mathcal{S}_{(2w-Pk_1)}\left\{\widehat{\mathbf{r}}_{\text{temp}}\right\}\right\} \\
\widetilde{\mathbf{r}}_i &= \widehat{\mathbf{r}}^{(1)}_{(i-1)} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{i-1}\right\};\; \forall i : 1 \leq i < L
\end{aligned} \tag{4.24}$$

For the case of two fail-stop failures, $\widehat{\mathbf{r}}^{(2)}_{x_1}$ and $\widehat{\mathbf{r}}^{(2)}_{x_2}$ with $0 \leq x_1, x_2 < L$, $x_2 > x_1$, $x_{\text{diff}} = x_2 - x_1$ and

$$x \equiv \begin{cases} x_2, & \text{if } x_{\text{diff}} < P \\ x_1, & \text{if } x_{\text{diff}} \geq P \end{cases} \tag{4.25}$$

We present the recovery steps for three classes of failure locations based on the calculated difference x_{diff}.

- if $x_{\text{diff}} \in \{1, L-1\}$, the corresponding value for x is selected given the constraint of (4.25).

  Tier-one output disentanglement can be achieved by:

$$\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp}} &= \sum_{i=0}^{(P-2)} (-1)^i \cdot \left(\mathcal{S}_{i\cdot k_2}\left\{\widehat{\mathbf{r}}^{(2)}_{(x+P-1-i)\bmod L}\right\}\right) \\
\widehat{\mathbf{r}}^{(1)}_{(x+P)\bmod L} &= \mathcal{S}_{-(2w-(P-1)k_2)}\left\{\mathcal{S}_{2w-(P-1)k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{\text{temp}}\right\}\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+1)\bmod L} &= \mathcal{S}_{-(P-1)k_2}\left\{(-1)^P \cdot \left(\widehat{\mathbf{r}}^{(1)}_{\text{temp}} - \widehat{\mathbf{r}}^{(1)}_{(x+P)\bmod L}\right)\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+i+1)\bmod L} &= \widehat{\mathbf{r}}^{(2)}_{(x+i)\bmod L} - \mathcal{S}_{k_2}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+i)\bmod L}\right\};\ \forall i:\ 1 \le i < L-1,\ i \ne P-1
\end{aligned} \tag{4.26}$$

and, on the next stage,

$$\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}} &= \sum_{i=0}^{(P-1)} (-1)^i \cdot \left(\mathcal{S}_{i\cdot k_1}\left\{\widehat{\mathbf{r}}^{(1)}_{(x+P-i)\bmod L}\right\}\right) \\
\widetilde{\mathbf{r}}_{(x+P+1)\bmod L} &= \mathcal{S}_{-(2w-Pk_1)}\left\{\mathcal{S}_{(2w-Pk_1)}\left\{\widehat{\mathbf{r}}_{\text{temp}}\right\}\right\} \\
\widetilde{\mathbf{r}}_{(x+1)\bmod L} &= \mathcal{S}_{-(P-1)k_1}\left\{(-1)^{P-1} \cdot \left(\widehat{\mathbf{r}}_{\text{temp}} - \widetilde{\mathbf{r}}_{(x+P+1)\bmod L}\right)\right\} \\
\widetilde{\mathbf{r}}_{(x+i+1)\bmod L} &= \widehat{\mathbf{r}}^{(1)}_{(x+i)\bmod L} - \mathcal{S}_{k_1}\left\{\widetilde{\mathbf{r}}_{(x+i)\bmod L}\right\};\ \forall i:\ 1 \le i < L,\ i \ne P
\end{aligned} \tag{4.27}$$

- If $1 < x_{\text{diff}} < P$ or $P < x_{\text{diff}} < L-1$ or ($x_{\text{diff}} = P$ and $(L \bmod P) \ne 0$), the disentanglement algorithm is designed such that the derivation of $x_{\text{diff}} < P$ is equivalent to that of $L - x_{\text{diff}}$ with the corresponding value of x chosen with respect to the constraint of (4.25). The first stage of disentanglement can be computed by:

$$
\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp}} &= \sum_{i=0}^{(P-2)} (-1)^i \cdot \left(\mathcal{S}_{i \cdot k_2} \left\{ \widehat{\mathbf{r}}^{(2)}_{(x+P-1-i) \bmod L} \right\} \right) \\
\widehat{\mathbf{r}}^{(1)}_{(x+P) \bmod L} &= \mathcal{S}_{-(2w-(P-1)k_2)} \left\{ \mathcal{S}_{(2w-(P-1)k_2)} \left\{ \widehat{\mathbf{r}}^{(1)}_{\text{temp}} \right\} \right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+1) \bmod L} &= \mathcal{S}_{-(P-1)k_2} \left\{ (-1)^P \cdot \left(\widehat{r}^{(1)}_{\text{temp}} - \widehat{\mathbf{r}}^{(1)}_{(x+P) \bmod L} \right) \right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x+i+1) \bmod L} &= \widehat{\mathbf{r}}^{(2)}_{(x+i) \bmod L} - \mathcal{S}_{k_2} \left\{ \widehat{\mathbf{r}}^{(1)}_{(x+i) \bmod L} \right\}; \ \forall i : 1 \leq i < x_{d1}, \ i \neq P-1 \\
& \text{with } x_{d1} \equiv \begin{cases} L - x_{\text{diff}}, & \text{if } x_{\text{diff}} < P \\ x_{\text{diff}}, & \text{if } x_{\text{diff}} \geq P \end{cases}
\end{aligned}
\tag{4.28}
$$

and on the next stage

$$
\begin{aligned}
\widehat{\mathbf{r}}_{\text{temp}} &= \sum_{i=0}^{(P-1)} (-1)^i \cdot \left(\mathcal{S}_{i \cdot k_1} \left\{ \widehat{\mathbf{r}}^{(1)}_{(x+P-i) \bmod L} \right\} \right) \\
\widetilde{\mathbf{r}}_{(x+P+1) \bmod L} &= \mathcal{S}_{-(2w-Pk_1)} \left\{ \mathcal{S}_{(2w-Pk_1)} \left\{ \widehat{\mathbf{r}}_{\text{temp}} \right\} \right\} \\
\widetilde{\mathbf{r}}_{(x+1) \bmod L} &= \mathcal{S}_{-(P-1)k_1} \left\{ (-1)^{P-1} \cdot \left(\widehat{\mathbf{r}}_{\text{temp}} - \widetilde{\mathbf{r}}_{(x+P+1) \bmod L} \right) \right\} \\
\widetilde{\mathbf{r}}_{(x+i+1) \bmod L} &= \widehat{\mathbf{r}}^{(1)}_{(x+i) \bmod L} - \mathcal{S}_{k_1} \left\{ \widetilde{\mathbf{r}}_{(x+i) \bmod L} \right\}; \ \forall i : 1 \leq i \leq x_{d1}, \ i \neq P \\
0 &= \widehat{\mathbf{r}}^{(2)}_{(x+L-i) \bmod L} - \mathcal{S}_{(k_1+k_2)} \left\{ \widetilde{\mathbf{r}}_{(x+L-i) \bmod L} \right\} \\
&\quad - \mathcal{S}_{k_1} \left\{ \widetilde{\mathbf{r}}_{(x+L-i+1) \bmod L} \right\} - \mathcal{S}_{k_2} \left\{ \widetilde{\mathbf{r}}_{(x+L-i+1) \bmod L} \right\} \\
&\quad - \widetilde{\mathbf{r}}_{(x+L-i+2) \bmod L}; \ \forall i : 1 \leq i < L - x_{d1}
\end{aligned}
\tag{4.29}
$$

The last part of (4.29) requires that a linear system with $(L - x_{d1} - 1)$ unknowns must be solved in order to completely recover all output data streams. For $F = 2$ fail-stop failures, a linear system comprising one or two unknowns will be solved to be able to efficiently recover the processed data belonging to all computing nodes. There-

fore, by inspecting the last part of (4.29), a linear system of equations with two unknowns is obtained, and can be solved using any well known method.

- Finally, If $x_{\text{diff}} = P$ and $(L \bmod P) = 0$, the first stage of disentanglement can be achieved by:

$$\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp1}} &= \sum_{i=0}^{(P-2)} (-1)^i \cdot \left(\mathcal{S}_{i \cdot k_2} \left\{\widehat{\mathbf{r}}^{(2)}_{(x_2-1-i)}\right\}\right) \\
\widehat{\mathbf{r}}^{(1)}_{x_2} &= \mathcal{S}_{-(2w-(P-1)k_2)} \left\{\mathcal{S}_{2w-(P-1)k_2} \left\{\widehat{\mathbf{r}}^{(1)}_{\text{temp}}\right\}\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x_2-P+1)} &= \mathcal{S}_{-(P-1)k_2} \left\{(-1)^P \cdot \left(\widehat{\mathbf{r}}^{(1)}_{\text{temp}} - \widehat{\mathbf{r}}^{(1)}_{x_2}\right)\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(i+1)} &= \widehat{\mathbf{r}}^{(2)}_{i} - \mathcal{S}_{k_2} \left\{\widehat{\mathbf{r}}^{(1)}_{i}\right\};\ \forall i : (x_2 - P + 1) \leq i < (x_2 - 1)
\end{aligned} \tag{4.30}$$

$$\begin{aligned}
\widehat{\mathbf{r}}^{(1)}_{\text{temp2}} &= \sum_{i=0}^{(P-2)} (-1)^i \cdot \left(\mathcal{S}_{i \cdot k_2} \left\{\widehat{\mathbf{r}}^{(2)}_{(x_1+L-1-i) \bmod L}\right\}\right) \\
\widehat{\mathbf{r}}^{(1)}_{x_1} &= \mathcal{S}_{-(2w-(P-1)k_2)} \left\{\mathcal{S}_{2w-(P-1)k_2} \left\{\widehat{\mathbf{r}}_{\text{temp2}}\right\}\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(x_1+P+1) \bmod L} &= \mathcal{S}_{-(P-1)k_2} \left\{(-1)^P \cdot \left(\widehat{\mathbf{r}}^{(1)}_{\text{temp2}} - \widehat{\mathbf{r}}^{(1)}_{x_1}\right)\right\} \\
\widehat{\mathbf{r}}^{(1)}_{(i+1) \bmod L} &= \widehat{\mathbf{r}}^{(2)}_{i} - \mathcal{S}_{k_2} \left\{\widehat{\mathbf{r}}^{(1)}_{i}\right\};\ \forall i : (x_1 + P + 1) \bmod L \leq i < (x_1 - 1)
\end{aligned} \tag{4.31}$$

It is evident that the first stage disentanglement of (4.30) and (4.31) produces all tier-1 entangled streams required for the second stage of disentanglement such that (4.24) can be employed for the second stage of disentanglement.

We have therefore shown in the set of equations above that any single or double fail-stop failures can indeed be mitigated for all $L \geq 5$.

More explicitly, the derivation of (4.22) is based on the knowledge that for any value of L and for $F = 2$, there exists a minimum number of PUs

with consecutive indices available for the disentanglement process, irrespective of the failure indices, $\{x_1, x_2\}$. These consecutive PUs are used for the computation of the set of $\widehat{\mathbf{r}}^{(1)}_{\text{temp}}$ values of (4.4), (4.28) or (4.31). For example, for $L = 7$ and $\{x_1, x_2\}$=$\{3, 4\}$, five PUs with indices $\{5, 6, 0, 1, 2\}$ will always be available for the computation of set of $\widehat{\mathbf{r}}^{(1)}_{\text{temp}}$ values. However, for $\{x_1, x_2\}$=$\{3, 5\}$ and $\{x_1, x_2\}$=$\{3, 6\}$, up to four and three PUs with consecutive indices exist after failure respectively. Therefore, for all combinations of $\{x_1, x_2\}$ for $\{L, F\} = \{7, 2\}$, at least three PUs with consecutive indices will always be available for data recovery. Generalizing this for all values of L and for $F = 2$, we found the minimum number of PUs with consecutive indices to be equal to $(P - 1)$. Hence, an optimally designed numerical entanglement algorithm with at most $P - 1$ consecutive nodes, would have $(P - 2)$-zones of overlap and only $P - 2$ arithmetic shift operations, in order to obtain the set of $\widehat{\mathbf{r}}^{(1)}_{\text{temp}}$ values, such that each $\widehat{r}^{(1)}_{\text{temp}}[n]$ holds two integer values in a separable "packed" format as described in Section 4.1.2 and as illustrated in Fig. 4.4(b) and Fig. 4.5(b). These two "packed" values can subsequently be extracted using the second and third parts of (4.4), (4.28) or (4.31).

Furthermore, it can be observed from Fig. 4.1 and Fig. 4.3, that an N-zones overlap in the numerical entanglement construct, implies that each individual value held within an entangled representation can occupy a maximum of $(N \cdot k_i + s_i)$-bits, while the entire entangled value must be held within $((N + 1) \cdot k_i + s_i)$-bits , with $i \in [1, F]$.

Putting these two points together, the second part of (4.22) ensures that each entangled value at tier-2 of numerical entanglement with $P - 2$

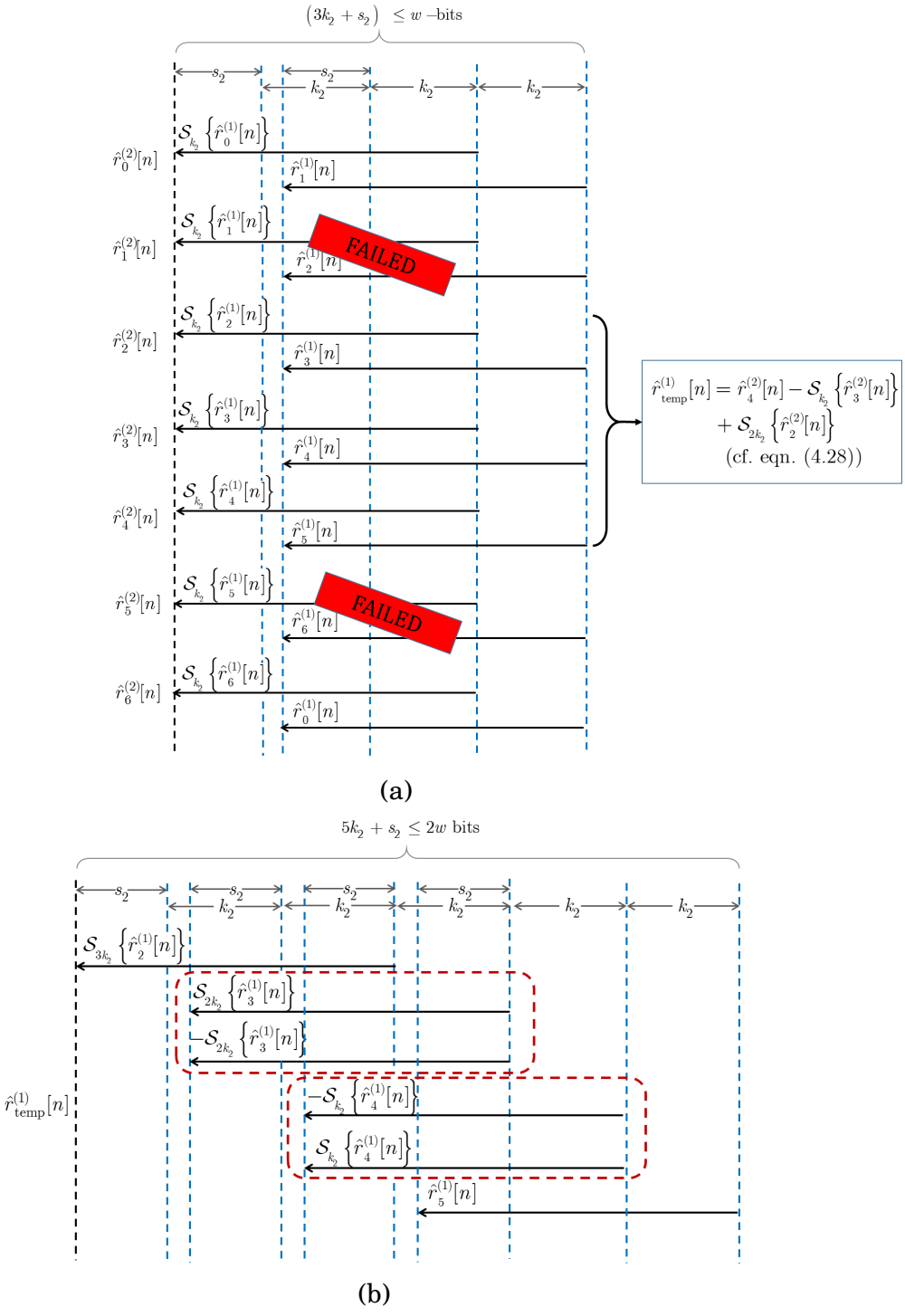


Figure 4.4: Illustration of entanglement of seven outputs with two failures after integer linear processing: (a) second tier of superimposed outputs showing the worst case failure locations; (b) derivation of the first set of temporary variables, $\hat{r}^{(1)}_{\text{temp}}[n]$, using the consecutive from (a) to obtain two physically separable integer values, $0 \leq n < N_{\text{out}}$. The maximum bit-width requirement for this tier of numerical entanglement is also highlighted.

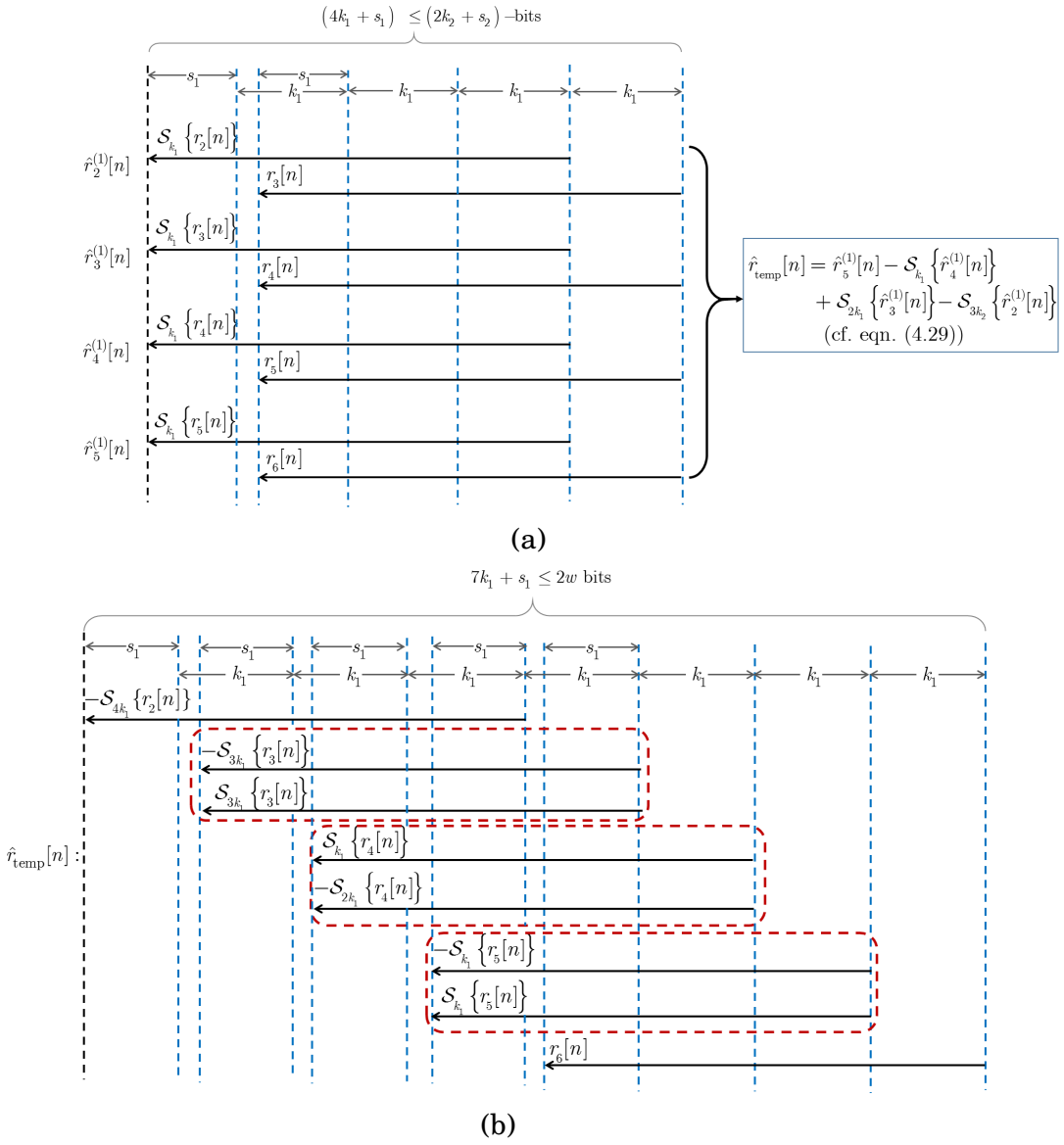


Figure 4.5: Illustration of first tier of numerical entanglement of seven outputs after integer linear processing: (a) outputs derived using the three consecutive nodes from the first stage of disentanglement; (b) derivation of the second set of $\widehat{r}_{\text{temp}}[n]$ values using the consecutive from (a) to obtain two physically separable integer values, $0 \leq n < N_{\text{out}}$. The maximum bit-width requirement for this tier of numerical entanglement is also highlighted.

"zones" of overlap and occupying a total of $((P-1)\cdot k_2 + s_2)$-bits, must not exceed the total available bit-width of integer representation, w. Also, after the first stage of disentanglement using $P-1$ consecutive nodes, we are guaranteed to have a minimum of P consecutive nodes available at the tier-1 of numerical entanglement as illustrated in Fig. 4.5(a). Following the same principle as above, and keeping in mind that each entangled value in tier-1 is held within an individual value that make up an entanglement for tier-2, it is straightforward to derive the constraint of the first part of (4.22). This description is used to derive the entanglement parameters for the generalization of numerical entanglement for any L, F value in subsequent sections. We illustrate in Figs (4.4) and (4.5), the worst case failure location for $\{L, F\} = \{7, 2\}$, the derivation of the separable "packed" values held within $\widehat{\mathbf{r}}^{(1)}_{\text{temp}}$ using $P-1 = \lceil \frac{L}{2} \rceil - 1 = 3$ consecutive nodes and $(P-2) = 2$ arithmetic shift operations, as well as the constraints on the supported bit-width at the two tiers of entanglement.

Finally, from the data layout of Fig. (4.5), and given the two addition operations required at the two tiers of numerical entanglement for the mitigation of fail-stop failures, the maximum dynamic range of data outputs is bounded by:

$$\forall n:\ r_0[n], \ldots, r_{L-1}[n] \in \left[-\left(2^{(P-1)k_1+s_1-1} - 2^{(P-1)k_1}\right),\ \left(2^{(P-1)k_1+s_1-1} - 2^{(P-1)k_1}\right)\right]. \tag{4.32}$$

Therefore, (4.32) comprises the range permissible for the computed LS operations with the generalized entangled representation derived from (4.9) and (4.10). Thus, we conclude that, for integer outputs with range

bounded by (4.32), the extraction mechanism of (4.23)–(4.31) is *necessary and sufficient* for the recovery of *any two stream losses from* $\widehat{\mathbf{r}}_0^{(2)}, \ldots, \widehat{\mathbf{r}}_{(L-1)}^{(2)}$.

4.2.3 Generalized Entanglement in Groups of L Inputs ($L \geq 3,\ F \leq \lfloor \frac{L-1}{2} \rfloor$)

We now examine the general case of (L, F) numerical entanglement, which uses F levels of linear superpositions of pairs of inputs to mitigate F fail-stop failures or detect F SDCs in each L-tuple of outputs, with $L \geq 2F + 1$.

First, for F levels of linear superpositions of pairs of inputs, similar description presented in Section 4.2.2 that considers the minimum number of consecutive PU indices required for the first stage of disentanglement for any $\{L, F\}$ and for $P = \lceil \frac{L}{F} \rceil$, leads to the derivation of the entanglement parameters given by:

$$\begin{aligned}
(P+F-2)\,k_1 + s_1 &\leq (P+F-4)\,k_2 + s_2 \\
(P+F-3)\,k_2 + s_2 &\leq (P+F-5)\,k_3 + s_3 \\
&\vdots \\
(P+1)\,k_{F-2} + s_{F-2} &\leq (P-1)\,k_{F-1} + s_{F-1} \\
P k_{F-1} + s_{F-1} &\leq (P-2)\,k_F + s_F \\
(P-1)\,k_F + s_F &\leq w
\end{aligned} \tag{4.33}$$

where k_x and s_x ($\forall x : 1 \leq x \leq F$) are selected to maximize the constraint of (4.33) and $s_x \leq k_x$. As before, the maximum bit-width available for representing each output is dependent on the pair $\{k_1, s_1\}$ and is given by

$(P+F-3)\,k_1+s_1$. However, in order to avoid integer overflow considering the addition operations required for entanglement at the F superposition stages, the maximum dynamic range supported by numerical entanglement for mitigating F fail-stop failures within a cluster of L processing nodes is given by:

$$\forall n : r_0[n],\cdots,r_{L-1}[n] \in \left[-\left(2^{(P+F-3)k_1+s_1-1}-2^{(P+F-3)k_1}\right),\left(2^{(P+F-3)k_1+s_1-1}-2^{(P+F-3)k_1}\right)\right]. \tag{4.34}$$

Examples of the tier-1 entanglement parameters together with the maximum bitwidth available for data outputs for different cases of L and F are given in Table 4.2 assuming a 32-bit representation. We also present the bitwidth permitted by the equivalent checksum method of Section 2.3.1 in order to ensure that its checksum streams do not overflow under a 32-bit representation and using linear checksum weights [5] [4], $\mathbf{w}_1 = [1,1,\cdots,1]^{\mathrm{T}}$, $\mathbf{w}_2 = [1,2,\cdots,L]^{\mathrm{T}}$ and $\mathbf{w}_3 = [2^0,2^1,\cdots,2^{L-1}]^{\mathrm{T}}$. More elaborately, we show in Fig. 4.6 the relationship between L, F and the maximum allowable bitwidth of data outputs when checksum-based and numerical entanglement-based failure mitigation methods are employed. The results show that numerical entanglement approaches the optimal bitwidth of the failure-intolerant processing as L increases for a fixed value of F. However, because of the non linear relationship between L and the maximum achievable bitwidth via numerical entanglement based on the allowable values of k_f and s_f of (4.33), certain combinations of L and F are sub-optimal as shown in the plot of Fig. 4.6. For example, for $\{L,F\} = \{18,1\}$, (4.33) shows that k_1 and s_1 can only take values of unity, leaving up to 14-bits unused within the 32-bit integer represen-

Table 4.2: Examples of bitwidth supported for the output data under $w = 32$ bit integer representation and: *(i)* the proposed approach; *(ii)* checksum-based failure mitigation. Any F failures in L streams (or F SDCs in each L-tuple $\{r_0, \ldots, r_{L-1}\}$) can be mitigated (or detected) under both frameworks, with the checksum-based method requiring F additional streams.

L	F	k_1	s_1	Maximum bitwidth supported by	
				Proposed	Checksum-based
3	1	11	10	21	30
5	1	7	4	25	29
5	2	5	5	15	28
7	1	5	2	27	29
7	2	5	4	19	27
7	3	3	3	12	25
8	1	4	4	28	29
8	2	5	4	19	26
8	3	3	3	12	24
11	1	3	2	29	28
11	2	4	2	22	25
11	3	3	3	15	21

tation. Comparatively, for $\{L, F\} = \{17, 1\}$, $k_1 = 2, s_1 = 0$ maximizes the available data bitwidth. Therefore, the results of Fig. 4.6 provide the optimal values of $\{L, F\}$, that maximize the dynamic range of data outputs. On the other hand, the use of checksum data for reliability improvement achieves close to optimal bitwidth requirements for smaller values of L, while they become almost impractical when exponential weight vectors (e.g. $\mathbf{w}_3 = [2^0, 2^1, \cdots, 2^{L-1}]^{\mathrm{T}}$) are used within larger values of L.

For the pre-processing of data inputs for failure mitigation, two inputs are entangled together (with one of the two shifted by k bits) to create each entangled input stream of data of each F stage of entanglement. Any LS operation is then performed directly on these input streams and

up to F fail-stop failures can be mitigated within each group of L outputs. The generalized entanglement vectors for $0 \leq l < L$ are obtained by:

$$\begin{aligned}
\widehat{\mathbf{a}}_l^{(1)} &= \mathcal{S}_{k_1}\left\{\mathbf{a}_l\right\} + \mathbf{a}_{l \bmod L} \\
\widehat{\mathbf{a}}_l^{(2)} &= \mathcal{S}_{k_2}\left\{\widehat{\mathbf{a}}_l^{(1)}\right\} + \widehat{\mathbf{a}}_{l \bmod L}^{(1)} \\
&\vdots \\
\widehat{\mathbf{a}}_l^{(F)} &= \mathcal{S}_{k_F}\left\{\widehat{\mathbf{a}}_l^{(F-1)}\right\} + \widehat{\mathbf{a}}_{l \bmod L}^{(F-1)}
\end{aligned} \tag{4.35}$$

After the application of the linear operation on $\left\{\widehat{\mathbf{a}}_0^{(F)}, \widehat{\mathbf{a}}_1^{(F)}, \cdots, \widehat{\mathbf{a}}_{(L-1)}^{(F)}\right\}$ to obtain the entangled outputs $\left\{\widehat{\mathbf{r}}_0^{(F)}, \widehat{\mathbf{r}}_1^{(F)}, \cdots, \widehat{\mathbf{r}}_{(L-1)}^{(F)}\right\}$, disentanglement ensues in order to obtain the desired output data streams, $\left\{\widetilde{\mathbf{r}}_0^{(F)}, \widetilde{\mathbf{r}}_1^{(F)}, \cdots, \widetilde{\mathbf{r}}_{(L-1)}^{(F)}\right\}$. The remainder of this section is dedicated to proving that recovery from F fail-stop failures is possible.

The proof is constructed by induction. Initially, it is noted that the cases $(L, F) \in \{(L,1), (L,2)\}$ hold, since they have been demonstrated in Sections 4.1 and 4.2.1.

Let us now assume that this holds for F entanglement levels derived via (4.35) with $L \geq 2F+1$ and k_x and s_x $(1 \leq x \leq F)$ selected such that the conditions of (4.33) hold and $\forall x : s_x \leq k_x$. We shall show that, under this assumption, this also holds for $F_+ \equiv F+1$ entanglement levels and F_+ failures in $L_+ \geq 2F+3$ streams, with F_+ constraints given by (4.33) under the replacement of F by F_+.

We first note that it suffices to prove this result for the equality case, i.e., $L_+ = 2F+3$, as having more than $2F+3$ streams will not influence the classification of failure patterns and recovery steps discussed in the following

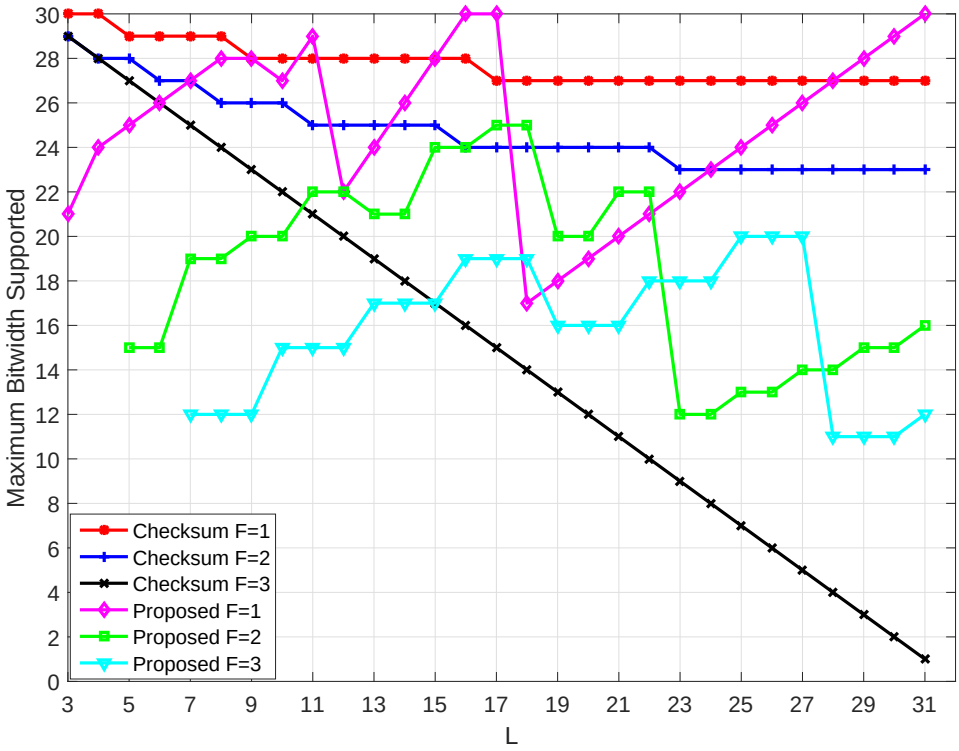


Figure 4.6: Maximum bitwidth supported for data outputs undergoing LS operation using 32-bit integer representation and mitigating up to F fail-stop failures for L data streams (or processing nodes). The checksum implementation uses checksum weights as proposed in [5] [4], i.e., $\mathbf{w}_1 = [1, 1, \cdots, 1]^{\mathrm{T}}$, $\mathbf{w}_2 = [1, 2, \cdots, L]^{\mathrm{T}}$ and $\mathbf{w}_3 = [2^0, 2^1, \cdots, 2^{L-1}]^{\mathrm{T}}$

parts of the proof (it simply provides more alternatives for recovery). After $F+1$ failures in $2F+3$ streams, we shall have $F+2$ available streams. Out of the $\binom{2F+3}{F+2}$ possible patterns, it is straightforward to show that any failure pattern that allows the availability of $\lfloor \frac{F+2}{2} \rfloor$ pairs of consecutively-numbered streams with at least one failure between any of the available pairs, will lead to the recovery of (at least) $F+3$ streams at level F, thereby having only (up to) F missing streams at level F, i.e., guaranteed recovery by the inductive assumption. However, in order to complete the proof, we also have to consider the worst case amongst all possible failure patterns, i.e., the failure pattern where *only* a single pair of consecutively-numbered streams is available. That is, $\exists x \in \{0, \ldots, L_+ - 1\}$ such that all of the following conditions hold:

- streams $\widehat{\mathbf{r}}_x^{(F+1)}$ and $\widehat{\mathbf{r}}_{(x+1) \bmod L_+}^{(F+1)}$ are available;
- stream $\widehat{\mathbf{r}}_{(x-1) \bmod L_+}^{(F+1)}$ and $\widehat{\mathbf{r}}_{(x+2) \bmod L_+}^{(F+1)}$ is *not* available and
- all other available streams are preceded and succeeded by a failed stream; therefore, they cannot be used for the direct extraction of any stream of level F.

The above condition is equivalent to the availability of $P-1$ PUs with consecutive nodes described in Section 4.2.2. From the pair of available streams of level $F+1$, we can extract the following three streams of entanglement level F: $\left\{\widehat{\mathbf{r}}_{(x-1) \bmod L_+}^{(F)}, \widehat{\mathbf{r}}_x^{(F)}, \widehat{\mathbf{r}}_{(x+1) \bmod L_+}^{(F)}\right\}$, via:

$$\begin{aligned}
\widehat{\mathbf{r}}^{(F)}_{\text{temp}} &= \widehat{\mathbf{r}}^{(F+1)}_{(x+1)\bmod L_+} - \mathcal{S}_{k_{F+1}}\left\{\widehat{\mathbf{r}}^{(F+1)}_{x}\right\} \\
\widehat{\mathbf{r}}^{(F)}_{(x+2)\bmod L_+} &= \mathcal{S}_{-2(w-k_{F+1})}\left\{\mathcal{S}_{2(w-k_{F+1})}\left\{\widehat{\mathbf{r}}^{(F)}_{\text{temp}}\right\}\right\} \\
\widehat{\mathbf{r}}^{(F)}_{x} &= \mathcal{S}_{-2k_{F+1}}\left\{-\left(\widehat{\mathbf{r}}^{(F)}_{\text{temp}} - \widehat{\mathbf{r}}^{(F)}_{(x+2)\bmod L_+}\right)\right\} \\
\widehat{\mathbf{r}}^{(F)}_{(x+1)\bmod L_+} &= \widehat{\mathbf{r}}^{(F+1)}_{x} - \mathcal{S}_{k_{(F+1)}}\left\{\widehat{\mathbf{r}}^{(F)}_{x}\right\}.
\end{aligned} \tag{4.36}$$

The last set of equations guarantees recovery because: *(i)* the last condition of (4.33) ensures that both $\widehat{\mathbf{r}}^{(F)}_{(x+2)\bmod L_+}$ and $\widehat{\mathbf{r}}^{(F)}_{x}$ can be extracted from $\widehat{\mathbf{r}}^{(F)}_{\text{temp}}$ and no overflow occurs; *(ii)* the penultimate condition of (4.33) ensures that the dynamic range at entanglement level F does not exceed the bitwidth available within the $(F+1)$-level inputs.

Given the availability of three consecutive streams at level F: $\widehat{\mathbf{r}}^{(F)}_{x}$,$\widehat{\mathbf{r}}^{(F)}_{(x+1)\bmod L_+}$ and $\widehat{r}^{(F)}_{(x+2)\bmod L_+}$, we can recover four consecutive streams at level $F-1$ and, by carrying the recovery process across all $F+1$ entanglement levels, we can recover $F+3$ consecutively-numbered output streams: $\left\{\widetilde{\mathbf{r}}_{x},\ldots,\widetilde{\mathbf{r}}_{(x+F+2)\bmod L_+}\right\}$. Therefore, F output streams will be unavailable after this recovery process. However, these are guaranteed to be recoverable from the F available and unused streams of level $F+1$, since we have F linear equations and F unknowns in the system of $2F+3$ streams of level $F+1$. Therefore, we can mitigate $F+1$ failures in $L_+ \geq 2F+3$ streams. This means we can mitigate any $F' > F$ failures if F' entanglement levels are carried out and $L' \geq 2F'+1$ and the conditions of (4.33) hold with the replacement of F by F'.

Remark 5 (detection and correction capabilities): Given that we can re-

cover all outputs from $L-F$ entangled streams, to detect F SDCs and to detect and correct up to $(F-1)$ SDCs within any L-tuple $\left[\widehat{r}_0^{(F)}[n] \quad \dots \quad \widehat{r}_{L-1}^{(F)}[n]\right]^{\mathrm{T}}$, for all output locations, $0 \leq n < N_{out}$, we recover L versions of all outputs $\left[\widetilde{r}_0[n] \quad \dots \quad \widetilde{r}_{L-1}[n]\right]^{\mathrm{T}}$ using ($\forall x \in \{0,1,\cdots,L-1\}$):

$$\{\widetilde{r}_0[n],\dots,\widetilde{r}_{L-1}[n]\}^{(x)} \xleftarrow{\text{disentangle}} \left\{\widehat{r}_x^{(F)}[n],\dots,\widehat{r}_{(x+L-F-1)\bmod L}^{(F)}[n]\right\} \tag{4.37}$$

and detect any errors in the xth recovery attempt by cross-comparing the results with their remaining recovered versions, since, *at least* two versions of $\{\widetilde{r}_0[n],\dots,\widetilde{r}_{L-1}[n]\}^{(x)}$ of (4.37) will be identical. If the xth recovery attempt is deemed to be erroneous, the correct recovery from the remaining $L-F$ streams is used instead. However, the occurrence of at least F SDCs will lead to nonidentical outputs for all recovery attempts and recomputation of all L data elements corresponding to the index n will be required.

Finally, the results of Table 4.2 and the derivation of (4.32) show that increased fault tolerance via numerical entanglement is achieved by sacrificing the dynamic range of data outputs for the performed LS computation. Specifically, it can be seen that $2 \sim 10$ bits of 32-bit integer representation is sacrificed for the mitigation of a single fail-stop failure. The equivalent checksum-based failure mitigation is achieved at a loss of $1 \sim 3$ bits for the L values presented in Table 4.2. However, numerical entanglement obtains comparable or wider bitwidth in comparison to the checksum-based approach for $L \geq 7$ and $F = 1$ or $L \geq 11$ and $F \in \{1,2\}$. At the same time, our proposal does not require the overhead of applying

the LS operations to *any* additional data, as it "overlays" the information of each input onto another input via the numerical entanglement of pairs of inputs.

4.3 Complexity in LS Operations with Numerical Entanglements

We now turn our attention to the cost of performing numerical entanglement, result extraction and error checking/ failure recovery versus the cost of the LS operation itself.

Consider L input integer data streams, each comprising several samples and consider that an LS operation $\boxtimes$ with kernel $\mathbf{b}$ is performed in each stream. This is the case, for example, under inner-products performed for GEMM or convolution/cross-correlation between multiple input streams for similarity detection or filtering applications or matrix-vector products in Lanczos iterations and iterative methods [34]. If the kernel $\mathbf{b}$ has substantially smaller length than the length of each input stream, the effective input stream size can be adjusted to the kernel length under overlap-save or overlap-add operation in convolution and cross-correlation [128] and several (smaller) overlapping input blocks can be processed independently. Similarly, block-major reordering is used in matrix products and transform decompositions for increased memory efficiency [35, 36, 129]. Thus, in the remainder of this section we assume that N expresses *both* the input data stream and kernel dimension.

The operations count (additions/multiplications) for stream-by-stream sum-of-products between a matrix comprising L subblocks of $N \times N$ integers and a matrix kernel comprising $N \times N$ integers (see [36,97,129,155] for example instantiations within high-performance computing environments) is: $\mathrm{C}_{\mathrm{GEMM}} = L\left(2N^3 - N^2\right)$. For sesquilinear operations like convolution and cross-correlation of L input integer data streams (each comprising N samples) with kernel $\mathbf{b}$, depending on the utilized realization, the number of operations can range from $O\left(N^2\right)$ for direct algorithms (e.g., time-domain convolution) to $O\left(N \log_2 N\right)$ for fast algorithms (e.g., FFT-based convolution) [128]. For example, for convolution or cross-correlation under these settings and an overlap-save realization for consecutive block processing, the number of operations (additions/multiplications) is [128]: $\mathrm{C}_{\mathrm{conv,time}} = 4LN^2$ for time domain processing and $\mathrm{C}_{\mathrm{conv,freq}} = L\left[(45N + 15)\log_2(3N + 1) + 3N + 1\right]$ for frequency-domain processing.

As described in Section 4.2.3, numerical entanglement of L input integer data streams (of N samples each) requires an $O\left(N\right)$ number of operations for the entanglement, extraction and SDC mitigation or failure recovery [including the L-fold recovery and error check of (4.37) of Remark 5]. For example, ignoring all arithmetic-shifting operations (which take a negligible amount of time [143]), based on the description of Section 4.2.3, the approximate number of operations for numerical entanglement, extraction and data recovery is: $C_{\mathrm{ne,conv}} = (2L + P - 2)\,FN$.

Similarly as before, for the special case of the GEMM operation using L subblocks of $N \times N$ integers, the approximate overhead of numerical

entanglement of all inputs is: $C_{\text{ne,GEMM}} = (2L + P - 2)\, FN^2$. We present the percentile values obtained for $\frac{\mathrm{C}_{\text{ne,GEMM}}}{\mathrm{C}_{\text{GEMM}}} \times 100\%$ in Fig. 4.7 for typical values of (L, F) and $N = \{500, 2000\}$, which represents a typical subblock size in high-performance GEMM. In addition, the values of $\frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,time}}} \times 100\%$ and $\frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,freq}}} \times 100\%$ are shown in Fig. 4.8 for convolution/ cross-correlation operations for kernel length, $N = 500$. For sesquilinear operations like matrix products the overhead of numerical entanglement, extraction and recovery in terms of arithmetic operations is below 3.5%. For convolution and cross-correlation this overhead varies from between 1.6% to 7% depending on the implementation as shown in Fig. 4.8. Moreover,

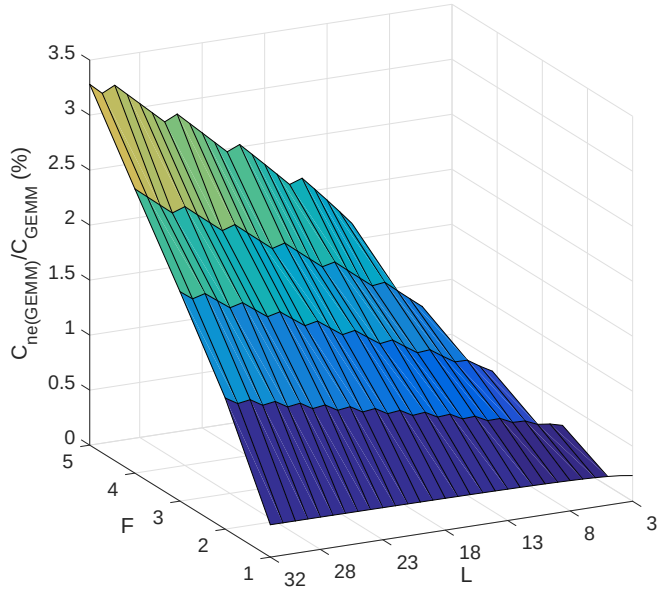
$$\lim_{N\to\infty} \frac{\mathrm{C}_{\text{ne,GEMM}}}{\mathrm{C}_{\text{GEMM}}} = \lim_{N\to\infty} \frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,time}}} = \lim_{N\to\infty} \frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,freq}}} = 0, \tag{4.38}$$

i.e., the overhead of the proposed approach approaches 0% as the dimension of the LS processing increases as illustrated for the case of GEMM in Fig. 4.7.

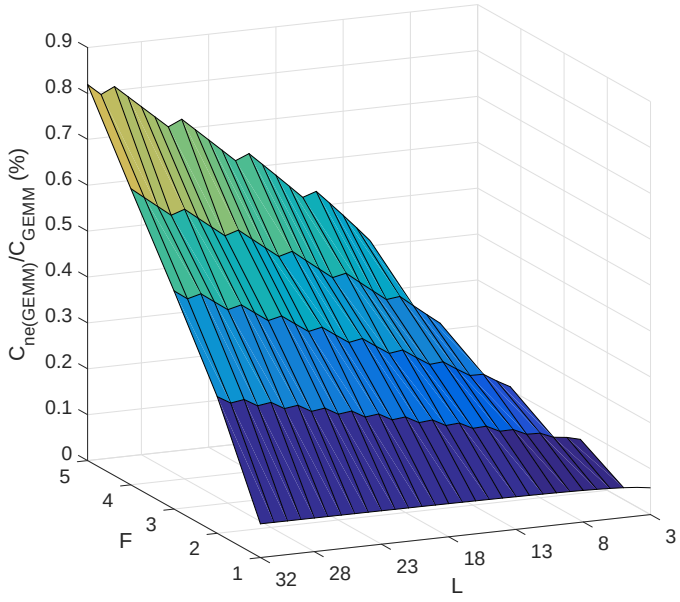
For comparison purposes, Fig. 4.9 shows the percentile overhead of GEMM computation using the checksum methods of [96, 97, 155] and under the same range of values for (L, F) and $N = 500$, i.e., for the same fail-stop failure or SDC mitigation capability[2].

Specifically, we examine the ratios: $\frac{\mathrm{C}_{\text{checksum,GEMM}}}{\mathrm{C}_{\text{GEMM}}} \times 100\%$, $\frac{\mathrm{C}_{\text{checksum,conv,time}}}{\mathrm{C}_{\text{conv,time}}} \times 100\%$ and $\frac{\mathrm{C}_{\text{checksum,conv,freq}}}{\mathrm{C}_{\text{conv,freq}}} \times 100\%$, where $\mathrm{C}_{\text{checksum,GEMM}} = (2LF - F - L) \cdot N^2 + \frac{F}{L}\mathrm{C}_{\text{GEMM}}$, $\mathrm{C}_{\text{checksum,conv,time}} = (2LF - F - L) \cdot N + \frac{F}{L}\mathrm{C}_{\text{conv,time}}$ and

[2]Unlike the row-column algorithm-based fault-tolerance method of Huang and Abraham [101], the checksum method for fail-stop failure mitigation in GEMM generates an additional (i.e., checksum) subblock, since the former cannot mitigate fail-stop failures.

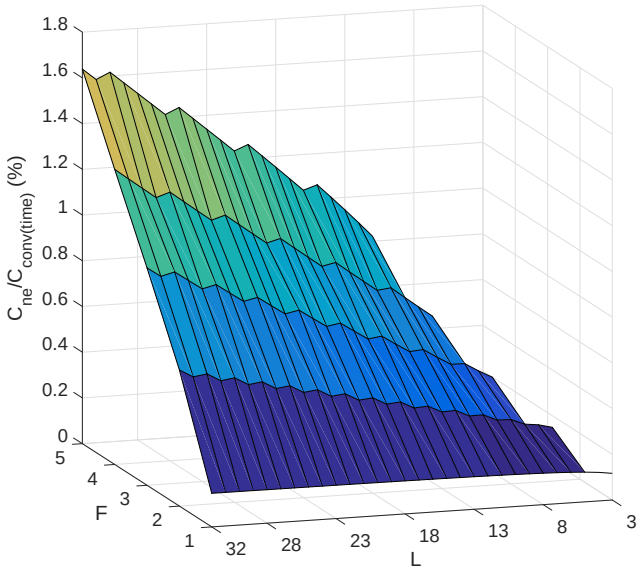


(a)

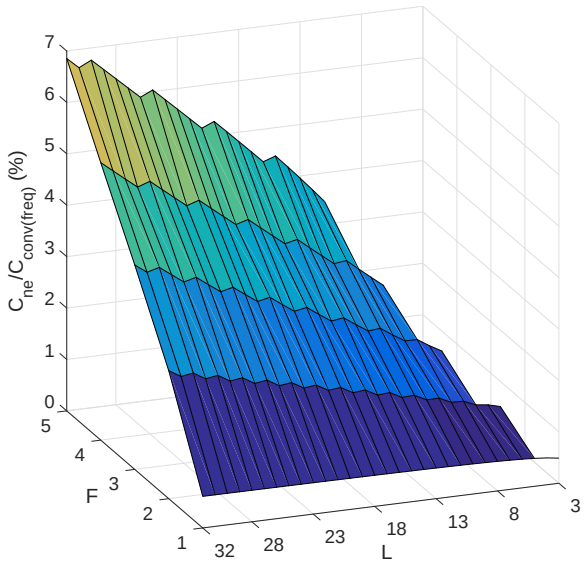


(b)

Figure 4.7: Ratios of arithmetic operations for numerical entanglement, extraction and failure recovery versus the arithmetic operations of L concurrent $N \times N$-by- $N \times N$ GEMM computations for: (a) $N = 500$; and (b) $N = 2000$. L is the number of processing nodes and F the number of fail-stop failures that can be mitigated.



(a)



(b)

Figure 4.8: Ratios of arithmetic operations for numerical entanglement, extraction and failure recovery versus the arithmetic operations of L concurrent convolution computations with kernel length, $N = 500$ for: (a) time-domain convolution; and (b) frequency-domain convolution. L is the number of processing nodes and F the number of fail-stop failures that can be mitigated.

$\mathrm{C}_{\text{checksum,conv,freq}} = (2LF - F - L) \cdot N + \frac{F}{L}\mathrm{C}_{\text{conv,freq}}$ represent the overhead in terms of operations count (additions/multiplications) for each case assuming the failures occurred within the checksum. Therefore, the presented complexity of the checksum-based approach is the lower bound on the number of operations required to mitigate fail-stop failures. Given that the checksum method for GEMM exhibits similar or better percentile overhead in comparison to the equivalent time-domain/frequency-domain convolution computation, Fig. 4.9 illustrates only the former. As expected, the overhead of checksum methods converge to $\frac{F}{L} \times 100\%$ as the dimension of the LS processing operations increases, i.e.,

$$\begin{aligned} \lim_{N\to\infty} \frac{\mathrm{C}_{\text{checksum,GEMM}}}{\mathrm{C}_{\text{GEMM}}} &= \lim_{N\to\infty} \frac{\mathrm{C}_{\text{checksum,conv,time}}}{\mathrm{C}_{\text{conv,time}}} \\ &= \lim_{N\to\infty} \frac{\mathrm{C}_{\text{checksum,conv,freq}}}{\mathrm{C}_{\text{conv,freq}}} \\ &= \frac{F}{L}. \end{aligned} \tag{4.39}$$

Therefore, checksum methods lead to substantial overhead (which can surpass 45%) when high reliability is pursued, i.e., when $L \leq 8$ and $F > 1$. Even for the low reliability regime (i.e., when $L > 8$ and $F = 1$), Fig. 4.9 shows that checksum methods can incur more than 4% overhead in terms of arithmetic operations. On the other hand, the proposed method always incurs less than 8% overhead for all presented sesquilinear operations. This overhead reduces even further with increasing values for N as shown for the case of GEMM in Fig. 4.7(b). Overall, the comparison between Fig. 4.7 and Fig. 4.9 demonstrates that the complexity overhead of the proposed approach is expected to be one to two orders of magnitude

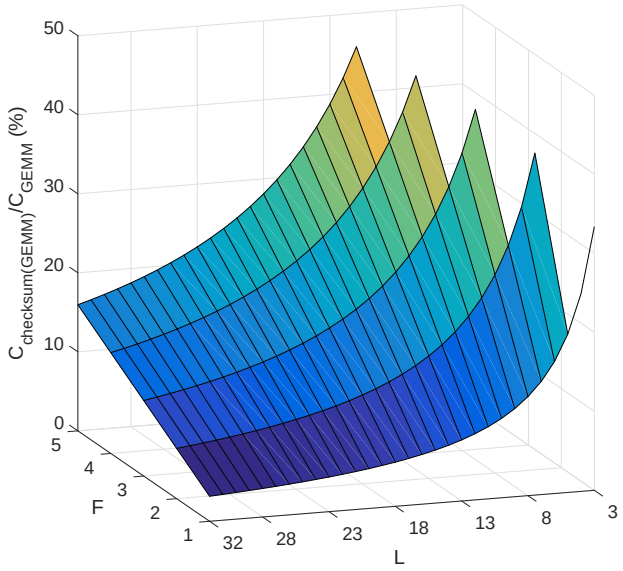


Figure 4.9: Ratio of arithmetic operations for checksum generation and recovery versus the arithmetic operations of generic matrix multiplication, with L the number of computing nodes and F the number of fail-stop failures that can be mitigated.

smaller than that of checksum methods.

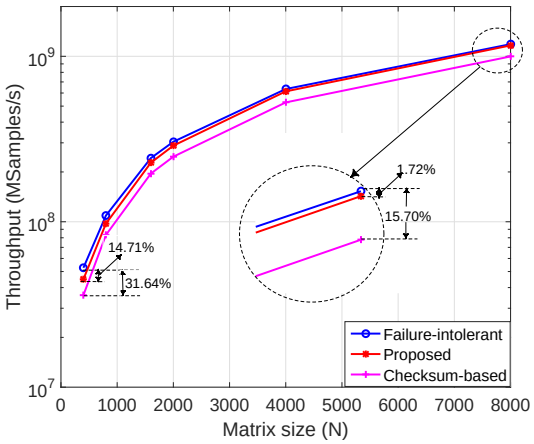
4.4 Experimental Validation

We present results using an Intel Xeon E5-2666v3 2.9GHz instance of Amazon EC2 (compute-optimized `c4.8xlarge`, reserved instance type, Windows Server 2012, Intel C++ 15.0 Compiler). All experiments were performed using the physical cores of the computing platform by setting the command prompt system affinity appropriately to ensure parallel execution of the Intel MKL GEMM routine. In Fig. 4.10 and Fig. 4.11, we present throughput results for the mitigation of up to $F = 3$ core failures

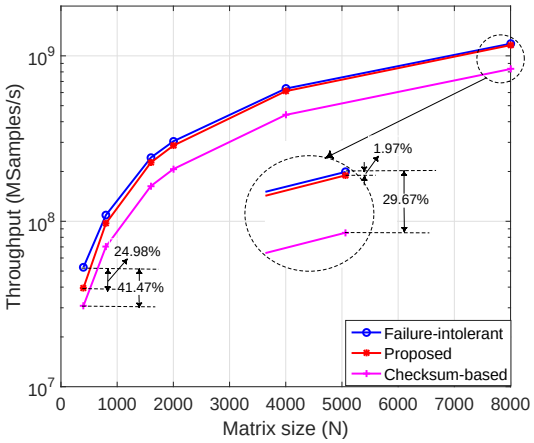
when running on $L = 7$ and $L = 11$ cores of the computing platform for the parallel $N \times N$ by $N \times N$ GEMM computation, with $N \in [400, 8000]$. All entanglement, disentanglement and checksum generation make use of the OpenMP framework, as well as the increased optimization offered by AVX2 SIMD instructions. In addition, all pre and post processing steps of all presented algorithms are performed using 32-bit integers, while inputs and outputs of the `dGEMM` routine undergo the appropriate casting/rounding from integer to floating point number representation and vice versa[3]. Importantly, the failure intolerant computation, together with the proposed algorithm utilize all L cores for data computation. On the other hand, the checksum method can only use $L_{\text{data}} = L - F$ cores for actual input data processing, while reserving F cores for checksum data computation. In our experiments, to create the redundant matrices required for fault tolerance, we utilized the weighted checksum-based parameters of Luk *et. al.* [5] [4] for $F = 3$ fail-stop failures with checksum weights given by the $1 \times L_{\text{data}}$ vectors: $\mathbf{w}_1 = [1, 1, \cdots, 1]^{\text{T}}$, $\mathbf{w}_2 = [1, 2, \cdots, L_{\text{data}}]^{\text{T}}$ and $\mathbf{w}_3 = [2^0, 2^1, \cdots, 2^{L_{\text{data}}-1}]^{\text{T}}$.

The results of Fig. 4.10 and Fig. 4.11 show the decrease in throughput (in Mega samples per second) against the failure intolerant parallel GEMM kernel realization based on the Intel MKL GEMM subroutine [35] for $L = \{7, 11\}$ and $F \in [1, 3]$. Specifically, the proposed algorithm incurs $1.72\% \sim 37.23\%$ decrease in computational throughput for the mitigation of (up to) 3 core failures during GEMM for the presented $\{L, N\}$ values. On

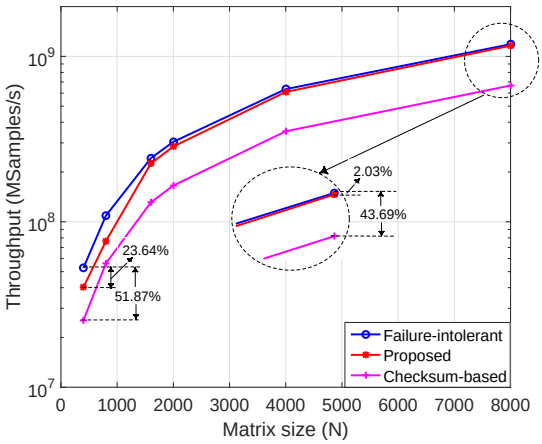
[3]Like all high-performance MKLs for general-purpose processors, Intel MKL only supports single and double-precision floating point; we opt for the latter to avoid approximations incurred from the loss of dynamic range in floating-point representations.



(a) $(L, F) = (7, 1)$

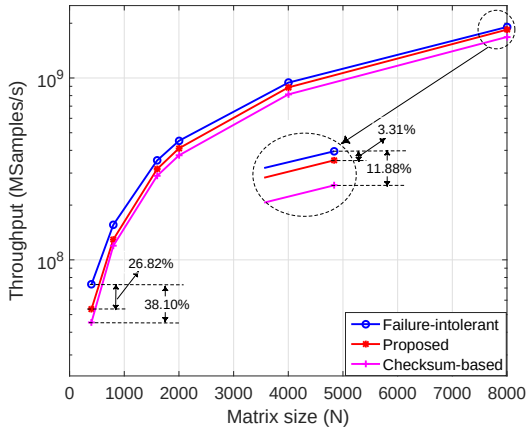


(b) $(L, F) = (7, 2)$

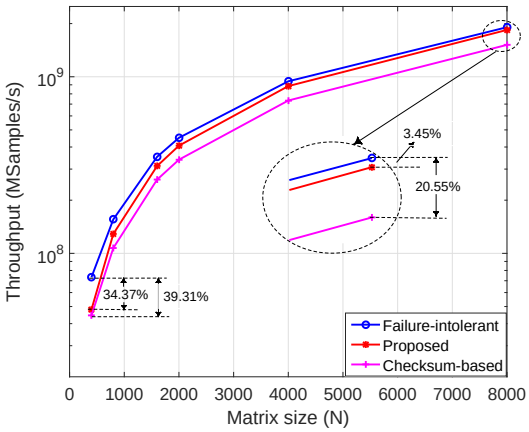


(c) $(L, F) = (7, 3)$

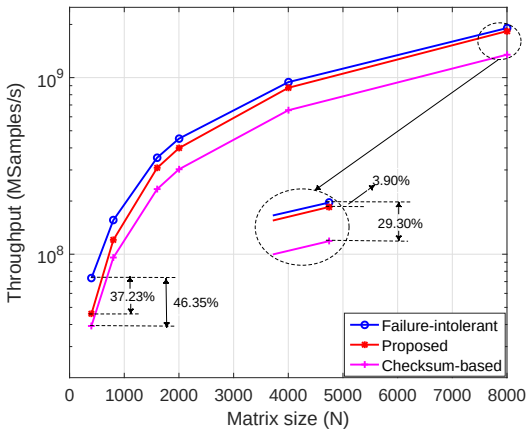
Figure 4.10: Throughput results for the $N \times N$-by-$N \times N$ GEMM computation for mitigating $F = \{1, 2, 3\}$ PU failures in $L = 7$ PUs. Failure-intolerant computation using Intel MKL 11.0 is used as a benchmark. Throughput axis is shown on a logarithmic scale.



(a) $(L, F) = (11, 1)$



(b) $(L, F) = (11, 2)$



(c) $(L, F) = (11, 3)$

Figure 4.11: Throughput results for the $N \times N$-by-$N \times N$ GEMM computation for mitigating $F = \{1, 2, 3\}$ PU failures in $L = 11$ PUs. Failure-intolerant computation using Intel MKL 11.0 is used as a benchmark. Throughput axis is shown on a logarithmic scale.

the other hand, the throughput achieved by the checksum-based [5] [4] failure mitigation method is 11.88%–51.87% lower than that of the failure-intolerant computation. Overall, generalized numerical entanglement is shown to incur 40% to 95% lower overhead than checksum methods while providing same or better level of recoverability.

4.5 Numerical Entanglement for SDC Detection

Beyond the mitigation of fail-stop failures in distributed computing platforms, generalized numerical entanglement can be used for the detection of fail-continue failures (i.e., silent data corruptions) in data computations. In this section we demonstrate this for the example cases of $(L, F) = \{(3,1), (5,1), (7,1)\}$, i.e., detection of one SDC within each triple, quintuple or septuple of outputs. In order to achieve the same level of detectability, the checksum approach produces a single checksum data stream (per case), which must be processed with the input kernel.

All experiments were performed on an Intel Xeon CPU E5-2670 v2 2.50GHz running Linux Ubuntu and using the clang3.2 compiler. Results are presented for a set of L vector-matrix multiplications of dimensions 1×2000-by-2000×2000. Artificial fault injection is performed during the above multiplication via KULFI [3], an open source LLVM-based [132] fault injection tool. Transient faults are injected during one of the L multiplications, with all L streams ($L+1$ streams for the checksum method) being

equiprobable.

To determine the number of injected SDCs for each fault injection experiment, KULFI performs two executions of the binary file: one error free and the other susceptible to SDCs. The outputs of the executions are compared in order to categorize the effect of the injected errors on output data with a typical error summary output presented as:

```
Segmentation Faults:  8, Benign Faults:  33, Out of
                Bounds:  2, SDC: 41
```

Table 4.3 shows the average execution time out of 200 single fault injection experiments using randomly generated inputs for the detection and correction of a single error within the M output data streams. Execution time for GEMM is measured during the error-free execution of KULFI, while the pre/post-processing execution time (including error correction by recomputation of the erroneous results) is measured for the error-prone execution of the same KULFI experiment. The results demonstrate that the overhead incurred by the proposed approach for SDC detection is negligible (less than 0.75%), and remains 2 orders of magnitude lower than that of the equivalent checksum-based method [4] [5]. In all cases, all faults were detected correctly by both the proposed and the checksum-based method.

Table 4.3: Average execution (in microseconds) and percentile comparisons against the fault-intolerant (conventional) Intel MKL `sGEMM` for a single error detection within L multiplications of size 1×2000-by-2000×2000.

	Preprocessing	Postprocessing	GEMM	% increase
$L = 3$				
Conventional	- - -	- - -	12322.40	- - -
Proposed	16.46	71.59	12322.40	0.71
Checksum	20.87	51.36	16420.73	33.26
$L = 5$				
Conventional	- - -	- - -	20554.80	- - -
Proposed	29.93	104.22	20554.80	0.65
Checksum	30.09	59.01	24669.09	20.02
$L = 7$				
Conventional	- - -	- - -	28899.00	- - -
Proposed	42.68	144.09	28899.00	0.65
Checksum	41.95	80.06	32772.71	13.40

4.6 Conclusions

We propose a new approach for LS processing of integer data streams that is based on the novel concept of numerical entanglement. Under L input streams ($L \geq 3$), the proposed approach provides for: *(i)* guaranteed mitigation of multiple (up to $F = \left\lfloor \frac{L-1}{2} \right\rfloor$) fail-stop failures or SDCs; *(ii)* complexity overhead that depends only on (L, F) and not on the complexity of the performed LS operations, thus, quickly becoming negligible as the complexity of the LS operations increases. These two features demonstrate that the proposed solution forms a *third family* of roll-forward fail-stop failure (or SDC) mitigation techniques (i.e., beyond the well-known and widely-used checksum-based methods and modular redundancy) and offers unique advantages, summarized in Table 4.1. As such, it is envisaged that it will find usage in a multitude of systems that require en-

hanced reliability against fail-stop failures in hardware with very low implementation overhead.

Chapter 5

Conclusions and Future Work

We have presented in this thesis a new class of methods for highly-reliable integer linear and sesquilinear computations. In Chapter 1, we reviewed the problem of transient faults and their proliferation as SDCs, the requirement for power-aware fault tolerance algorithms especially for the case of fail-stop failures and existing works on algorithmic-level SDC/fail-stop failure mitigation. We introduced numerical packing for SDC detection for matrix products in Chapter 2, while analyzing its complexity and reliability. All analysis carried out in Chapter 2 and subsequent chapters in this thesis were done in comparison to conventional fault-intolerant processing, the well-established method of checksum processing and the generalized method of hardware/computation replication. Our analysis shows that the proposed approach incurs less than 1% additional overhead in comparison to the traditional ABFT approach when no SDCs are detected within output GEMM products and becomes over 30% more efficient than ABFT when multiple errors exist. Practical fault injection

using the tool, KULFI, provided the basis for further analysis of the execution time and energy consumption overhead of numerical packing and other existing SDC mitigation methods, thus verifying our theoretical propositions.

The concept of numerical packing is further re-designed for single core/processor failure mitigation within a cluster of shared/distributed memory computers in Chapter 3. While numerical packing eliminates the use of checksum data elements by sacrificing 37.5% of maximum output dynamic range of data outputs, we show experimentally that no performance degradation is exhibited when running medium-scale multimedia retrieval computations. On the contrary, the need for high throughput within such algorithms that are often real time based implies that the reservation of a fraction of the computing resources for checksum data processing incurs a substantial degradation in processing throughput. For a multimedia retrieval task deployed within a cluster of AWS EC2 spot (i.e., low-cost albeit terminable) instances, our proposal leads to: *(i)* 16%–23% cost reduction against the equivalent checksum-based method and *(ii)* more than 70% cost reduction against conventional failure intolerant processing based on AWS EC2 on-demand (i.e., higher cost albeit guaranteed) instances.

Finally, we generalize our numerical representation-based redundancy technique by proposing a multiple failure mitigation technique: numerical entanglement. Multi-tier numerical entanglement as proposed in this thesis builds on the earlier proposed method of single-tier entanglement such that increased reliability is achieved by sacrificing the dy-

namic range of data outputs. We derive expressions for failure recovery assuming the failed node indexes are known and prove that numerical entanglement can indeed mitigate up to F failures within a cluster size of $L \geq 2F + 1$.

Our proposals form a new class of algorithm level fault tolerance methods different from the well established methods of hardware duplication/replication and checksum generation, storage and processing. While the application of the numerical packing proposal of Chapter 2 is limited to SDC detection in GEMM, the property of error location confinement to a quadruple of GEMM outputs, ensures minimal execution time overhead for error correction especially in multiple error cases. On the other hand, the numerical packing construct of Chapter 3 tolerates both single fail-stop and fail-continue failures for all classes of sum-of-product computations. However, more computation is required for SDC correction as the error locality extends to a sextuple of data outputs. Finally, the multi-tier numerical entanglement construct proposed in Chapter 4 allows for the mitigation of multiple fail-stop and fail-continue failures within all classes of linear and sesquilinear operations. As opposed to the simpler construction of numerical packing, numerical entanglement requires a careful derivation of the entanglement parameters, in order to support the dynamic range requirements of practical applications, while avoiding integer data overflow. In summary, all proposed algorithms offer comparable or better execution time overhead in comparison to the equivalent checksum-based fault tolerance methods for data computations.

Further investigation is required towards generalizing the proposed numerical-representation-based redundancy technique to non-linear integer processing algorithms including sorting, sum of squares and sum of absolute difference computations. This would facilitate the development of a software suite for reliable multimedia processing, as such non-linear computations are ubiquitous.

In addition, the derivation of a checksum-less failure mitigation algorithm for real number data computations is of great interest especially for low to medium-scale real-time data processing applications. Although the proposals of this thesis do not apply to floating-point operations (due to their non-linear nature), it is possible that reduced precision reliable processing using fixed point and logarithmic number systems may be achievable and could be investigated as future work.

Furthermore, a hardware implementation of the numerical packing/entanglement algorithm for SDC mitigation is necessary for validating the theoretical complexity derivations of this work. This can be achieved by enabling reliable computation within application-specific devices, as these devices (some of which are operated in sub-normal conditions, e.g. voltage scaled) have become widespread. Specifically, by targeting GEMM/CONV-dominated algorithms running on application-specific devices, the performance and energy consumption requirements of the proposals of this thesis can be investigated. Holistic reliability can also be achieved by duplicating non-linear computations that cannot be protected via numerical packing/entanglement.

Finally, because the design of our numerical representation based redun-

dancy scheme belongs to the class of non-systematic codes, the need for data transfer for post-processing in both cases of error and error free situations can be undesirable for some applications. Therefore, the investigation of local entanglement groups, similar to the locally repairable codes of [158] is suggested as a future research work. The results of Fig. 4.6 provides information on optimal group sizes for different L and F sizes that maximize the dynamic range of data outputs.

Bibliography

[1] S.-M. Kang and Y. Leblebici, *CMOS digital integrated circuits*. Tata McGraw-Hill Education, 2003. xi, 3

[2] A. A. Al-Yamani *et al.*, "Performance evaluation of checksum-based ABFT," in *Proc. IEEE Int. Symp. Defect and Fault Tol. in VLSI Systems*, pp. 461–466, IEEE, October 2001. xi, 16, 71

[3] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "Towards formal approaches to system resilience," in *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*, pp. 41–50, IEEE, 2013. xii, 22, 45, 65, 161

[4] F. T. Luk, "Algorithm-based fault tolerance for parallel matrix equation solvers," *SPIE Real-Time Signal processing VIII*, vol. 564, pp. 631–635, 1985. xii, xiii, 15, 18, 60, 61, 64, 79, 93, 100, 144, 147, 158, 161, 162

[5] V. K. Stefanidis and K. G. Margaritis, "Algorithm based fault tolerance: Review and experimental study," in *Proc. Int. Conf. of Numer. Anal. and Appl. Math.*, IEEE, 2004. xii, xiii, 18, 50, 60, 61, 64, 79, 93, 144, 147, 158, 161, 162

[6] N. Rexford, J.; Jha, "Algorithm-based fault tolerance for floating-point operations in massively parallel systems," in *Proc. IEEE Int. Symp. on Circ. and Syst.*, vol. 2, pp. 649,652, IEEE, May 1992. xii, 15, 17, 18, 42, 60, 61, 64, 79, 100

[7] J.-Y. Jou and J. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," in *Proc. of the IEEE*, vol. 74, pp. 732,741, IEEE, May 1986. xii, 14, 17, 60, 61, 79

[8] T. Davies, C. Karlsson, H. Liu, C. Ding, and Z. Chen, "High performance linpack benchmark: a fault tolerant implementation without checkpointing," in *Proc. Int. Conf. on Supercomputing*, pp. 162–171, ACM, 2011. xii, 19, 60, 61, 70, 79

[9] H. Jégou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Trans. Pat. Anal. and Machine Intel.*, vol. 34, no. 9, pp. 1704–1716, 2012. xii, 21, 24, 25, 74, 75, 108

[10] S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala, and S. A. Wender, "Predicting the number of fatal soft errors in los alamos national laboratory's asc q supercomputer," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 329–335, 2005. 1

[11] J. Sloan, R. Kumar, and G. Bronevetsky, "Algorithmic approaches to low overhead fault detection for sparse linear algebra," in *Proc.*

42nd Annual IEEE/IFIP Int. Conf. on Depend. Syst. and Netw. (DSN), 2012, pp. 1–12, IEEE, 2012. 1, 5, 19, 84

[12] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 305–316, 2005. 2, 11

[13] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *Proceedings of the 50th Annual Design Automation Conference*, p. 99, ACM, 2013. 2

[14] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010. 2, 4

[15] A. Timor, A. Mendelson, Y. Birk, and N. Suri, "Using underutilized CPU resources to enhance its reliability," *IEEE Trans. on Depend. and Secure Comp.*, vol. 7, no. 1, pp. 94–109, 2010. 2

[16] M. Nicolaidis, L. Anghel, Y. Zorian, T. Karnik, K. Bowman, J. Tschanz, S.-L. Lu, C. Tokunaga, A. Raychowdhury, M. Khellah, *et al.*, "Design for test and reliability in ultimate cmos," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 677–682, EDA Consortium, 2012. 2

[17] A. R. Alameldeen *et al.*, "Energy-efficient cache design using variable-strength error-correcting codes," in *38th IEEE Int. Symp.*

on Computer Archit. (ISCA), 2011, pp. 461–471, IEEE, 2011. 4, 11, 76, 78

[18] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," 2008. 4, 8, 11, 13, 14, 21

[19] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, 2010. 4, 83

[20] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Conference Series*, vol. 78, p. 012022, IOP Publishing, 2007. 4

[21] N. Taerat, N. Nakisinehaboon, C. Chandler, J. Elliot, C. Leangsuksun, G. Ostrouchov, and S. Scott, "Using log information to perform statistical analysis on failures encountered by large-scale hpc deployments," in *Proceedings of the 2008 High Availability and Performance Computing Workshop*, vol. 4, pp. 29–43, 2008. 4

[22] M. Snir, W. D. Gropp, and P. Kogge, "Exascale research: preparing for the post-moore era," *Computer-Science Whitepapers*, 2011. 4

[23] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proc. IEEE Int.*

Conf. on High Perf. Comp., Netw., Stor. and Anal., p. 78, IEEE Computer Society Press, 2012. 4, 12, 13, 83

[24] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Europ. Conf. on Comp. Syst. (Eurosys)*, p. 18, ACM, 2015. 4, 5

[25] Y. Gong, A. C. Zhou, and B. He, "Monetary cost optimizations for HPC applications on Amazon clouds: Checkpoints and replicated execution," *Supercomputing conference, SC'14 (Poster)*, 2014. 4

[26] C. S. Chan, B. Pan, K. Gross, K. Vaidyanathan, and T. Š. Rosing, "Correcting vibration-induced performance degradation in enterprise servers," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 3, pp. 83–88, 2014. 5

[27] W. Zhu *et al.*, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, pp. 59–69, May 2011. 5

[28] Y. Wen, X. Zhu, J. Rodrigues, and C. W. Chen, "Cloud mobile media: Reflections and outlook," *IEEE Trans. on Multimedia*, vol. 16, pp. 885–902, June 2014. 5

[29] B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas, "Million query track 2009 overview," in *Proc. TREC'09*, vol. 9, 2009. 5, 83

[30] Y.-G. Jiang, Q. Dai, T. Mei, Y. Rui, and S.-F. Chang, "Super fast event recognition in internet videos," *IEEE Trans. on Multimedia*, vol. 17, pp. 1174–1186, Aug 2015. 5, 83

[31] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: bringing order to the web.," 1999. 5, 83

[32] J. Yang, D. Zhang, A. F. Frangi, and J.-Y. Yang, "Two-dimensional PCA: a new approach to appearance-based face representation and recognition," *IEEE Trans. on Patt. Anal. and Machine Intell.*, vol. 26, no. 1, pp. 131–137, 2004. 5, 21, 24, 84

[33] Z. Zhong, J. Zhu, and H. S.C.H., "Fast object retrieval using direct spatial matching," *IEEE Trans. on Multimedia*, vol. 17, pp. 1391–1397, Aug 2015. 5, 84

[34] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. Johns Hopkins University Press, 1996. 5, 21, 151

[35] M. Intel, "Intel math kernel library," 2007. 5, 6, 25, 84, 104, 151, 158

[36] K. Goto and R. A. Van De Geijn, "Anatomy of high-performance matrix multiplication," *ACM Trans. Math. Soft*, vol. 34, no. 3, p. 12, 2008. 5, 6, 25, 26, 34, 70, 84, 151, 152

[37] C. F. Van Loan, "The ubiquitous kronecker product," *Journal of computational and applied mathematics*, vol. 123, no. 1, pp. 85–100, 2000. 5, 84

[38] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 3, p. 11, 2008. 5, 84

[39] Y. Andreopoulos and M. van der Schaar, "Incremental refinement of computation for the discrete wavelet transform," *IEEE Trans. on Signal Process.*, vol. 56, no. 1, pp. 140–157, 2008. 5, 24, 84

[40] Y. Andreopoulos, "Error tolerant multimedia stream processing: There's plenty of room at the top (of the system stack)," *IEEE Trans. on Multimedia*, vol. 15, no. 2, pp. 291–303, 2013. 5, 20, 22, 73, 84, 85

[41] D. Anastasia and Y. Andreopoulos, "Linear image processing operations with operational tight packing," *IEEE Sig. Process. Let.*, vol. 17, no. 4, pp. 375–378, 2010. 5, 32, 38, 39, 84, 88

[42] A. Kadyrov and M. Petrou, "The" invaders' algorithm: Range of values modulation for accelerated correlation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1882–1886, 2006. 5, 27, 84

[43] E. Stewart, "Intel integrated performance primitives: How to optimize software applications using Intel IPP," 2004. 6, 84, 92

[44] M. Anam and Y. Andreopoulos, "Reliable linear, sesquilinear and bijective operations on integer data streams via numerical entanglement," *IEEE Transactions on Signal Processing*, vol. PP, no. 99, pp. 1–1, 2016. 8, 114, 115, 125

[45] W. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007. 8

[46] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, *et al.*, "Clear: Cross-layer exploration for architecting resilience-combining hardware and software techniques to tolerate soft errors in processor cores," *arXiv preprint arXiv:1604.03062*, 2016. 8, 11, 13, 14, 21

[47] T. Semiconductor, "Soft errors in electronic memory-a white paper," 2004. 9

[48] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pp. 243–247, IEEE, 2005. 9, 10

[49] M. Nicolaidis, "Circuit-level soft-error mitigation," in *Soft Errors in Modern Electronic Systems*, pp. 203–252, Springer, 2011. 9

[50] R. Hegde and N. R. Shanbhag, "Algorithmic noise-tolerance for low-power signal processing in the deep submicron era," in *Signal Processing Conference, 2000 10th European*, pp. 1–4, IEEE, 2000. 9

[51] R. Baumann *et al.*, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction," in *International Electron Devices Meeting*, pp. 329–332, IEEE; 1998, 2002. 10

[52] S. Mitra, K. Brelsford, Y. M. Kim, H.-H. Lee, and Y. Li, "Robust system design to overcome cmos reliability challenges," *Emerging and*

Selected Topics in Circuits and Systems, IEEE Journal on, vol. 1, no. 1, pp. 30–41, 2011. 10

[53] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *Computer*, no. 2, pp. 43–52, 2005. 10

[54] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 389–398, IEEE, 2002. 10, 11

[55] F. Firouzi, M. E. Salehi, F. Wang, and S. M. Fakhraie, "An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects," *Microelectronics Reliability*, vol. 51, no. 2, pp. 460–467, 2011. 10

[56] S. Yang, W. Wang, T. Lu, W. Wolf, N. Vijaykrishnan, and V. Xie, "Case study of reliability-aware and low-power design," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 16, no. 7, p. 861, 2008. 10

[57] M. Nicolaidis, "Design for soft error mitigation," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 405–418, 2005. 10

[58] R. R. Rao, D. Blaauw, and D. Sylvester, "Soft error reduction in combinational logic using gate resizing and flipflop selection," in *2006*

IEEE/ACM International Conference on Computer Aided Design, pp. 502–509, IEEE, 2006. 10

[59] D. Slocum, J. Mabra, A. Jordan, and T. Farris, "A radiation-hardened high speed, low power, 4-megabit sram fabricated using a 0.18 mum cmos commercial foundry," in *ESA Special Publication*, vol. 536, p. 355, 2004. 10

[60] Q. Zhou and K. Mohanram, "Cost-effective radiation hardening technique for combinational logic," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pp. 100–106, IEEE Computer Society, 2004. 10

[61] P. Hazucha, T. Karnik, S. Walstra, B. A. Bloechel, J. W. Tschanz, J. Maiz, K. Soumyanath, G. E. Dermer, S. Narendra, V. De, *et al.*, "Measurements and analysis of ser-tolerant latch in a 90-nm dual-v t cmos process," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, pp. 1536–1543, 2004. 10

[62] A. Narsale and M. C. Huang, "Variation-tolerant hierarchical voltage monitoring circuit for soft error detection," in *2009 10th International Symposium on Quality Electronic Design*, pp. 799–805, IEEE, 2009. 10

[63] R. P. Bastos, F. S. Torres, G. Di Natale, M. Flottes, and B. Rouzeyre, "Novel transient-fault detection circuit featuring enhanced bulk built-in current sensor with low-power sleep-mode," *Microelectronics Reliability*, vol. 52, no. 9, pp. 1781–1786, 2012. 10

[64] F. S. Torres and R. P. Bastos, "Detection of transient faults in nanometer technologies by using modular built-in current sensors," *Journal of Integrated Circuits and Systems*, vol. 8, no. 2, pp. 89–97, 2013. 10

[65] K. Mohanram and N. A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *ITC*, vol. 3, pp. 893–901, 2003. 10

[66] G. Neuberger, F. De Lima, L. Carro, and R. Reis, "A multiple bit upset tolerant sram memory," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 4, pp. 577–590, 2003. 11

[67] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950. 11

[68] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in srams," in *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pp. 222–225, IEEE, 2008. 11

[69] M. Richter, K. Oberlaender, and M. Goessel, "New linear sec-ded codes with reduced triple bit error miscorrection probability," in *2008 14th IEEE International On-Line Testing Symposium*, pp. 37–42, IEEE, 2008. 11

[70] I. Lee, M. Basoglu, M. Sullivan, D. H. Yoon, L. Kaplan, and M. Erez, "Survey of error and fault detection mechanisms," *University of Texas at Austin, Tech. Rep*, 2011. 11, 12

[71] P. J. Eibl, A. D. Cook, and D. J. Sorin, "Reduced precision checking for a floating point adder," in *2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 145–152, IEEE, 2009. 11

[72] K. Seetharam, L. C. T. Keh, R. Nathan, and D. J. Sorin, "Applying reduced precision arithmetic to detect errors in floating point multiplication," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 232–235, IEEE, 2013. 11

[73] P. Ampadu, M. Zhang, and V. Stojanovic, "Breaking the energy barrier in fault-tolerant caches for multicore systems," in *Proc. IEEE Int. Conf. on Des., Autom. and Test in Europe, DATE'13*, pp. 731–736, EDA Consortium, 2013. 11

[74] D. Rossi, N. Timoncini, M. Spica, and C. Metra, "Error correcting code analysis for cache memory high reliability and performance," in *Proc. IEEE Int. Conf. on Des., Autom. and Test in Europe, DATE'11*, pp. 1–6, IEEE, 2011. 11

[75] H. M. Quinn, A. De Hon, and N. Carter, "CCC visioning study: system-level cross-layer cooperation to achieve predictable systems from unpredictable components," tech. rep., Los Alamos National Laboratory (LANL), 2011. 11

[76] D. Li, Z. Chen, P. Wu, and J. S. Vetter, "Rethinking algorithm-based fault tolerance with a cooperative software-hardware approach," in *Proceedings of the International Conference on High Performance*

Computing, Networking, Storage and Analysis, p. 44, ACM, 2013. 11

[77] N. P. Carter, H. Naeimi, and D. S. Gardner, "Design techniques for cross-layer resilience," in *Proc. IEEE Int. Conf. on Des., Autom. and Test in Europe, DATE'10*, pp. 1023–1028, European Design and Automation Association, 2010. 11

[78] A. DeHon, H. M. Quinn, and N. P. Carter, "Vision for cross-layer optimization to address the dual challenges of energy and reliability," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 1017–1022, IEEE, 2010. 11, 21

[79] M. Abd-El-Barr, *Design and analysis of reliable and fault-tolerant computer systems*. World Scientific, 2006. 12

[80] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004. 12

[81] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990. 12

[82] M. Treaster, "A survey of fault-tolerance and fault-recovery techniques in parallel systems," *ACM Computing Research Repository (CoRR*, vol. 501002, pp. 1–11, 2005. 12, 20

[83] A. J. McAuley, "Reliable broadband communication using a burst erasure correcting code," in *ACM SIGCOMM Computer Communication Review*, vol. 20, pp. 297–306, ACM, 1990. 12

[84] G. E. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pjesivac-Grbovic, K. London, and J. J. Dongarra, "Extending the MPI specification for process fault tolerance on high performance computing systems," in *Proceedings of the International Supercomputer Conference (ICS)*, 2004. 12

[85] V. S. Sunderam, "Pvm: A framework for parallel distributed computing," *Concurrency: practice and experience*, vol. 2, no. 4, pp. 315–339, 1990. 12

[86] I. P. Egwutuoha *et al.*, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013. 12, 13

[87] C. Wang *et al.*, "A job pause service under LAM/MPI+ BLCR for transparent fault tolerance," in *IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2007.*, pp. 1–10, IEEE, 2007. 12

[88] G. Bronevetsky *et al.*, "Automated application-level checkpointing of mpi programs," *ACM Sigplan Notices*, vol. 38, no. 10, pp. 84–94, 2003. 12

[89] J. S. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 10, pp. 972–986, 1998. 12, 21

[90] A. J. Oliner, L. Rudolph, and R. K. Sahoo, "Cooperative checkpointing: a robust approach to large-scale systems reliability," in *Pro-*

ceedings of the 20th annual international conference on Supercomputing, pp. 14–23, ACM, 2006. 13

[91] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pp. 1–11, IEEE, 2010. 13

[92] J. T. Daly *et al.*, "Application MTTFE vs. platform MTBF: A fresh perspective on system reliability and application throughput for computations at scale," in *8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID'08.*, pp. 795–800, IEEE, 2008. 13

[93] J. W. Bennett, G. J. Atkinson, B. C. Mecrow, and D. J. Atkinson, "Fault-tolerant design considerations and control strategies for aerospace drives," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 5, pp. 2049–2058, 2012. 13

[94] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for hpc," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pp. 615–626, IEEE, 2012. 13

[95] C. Engelmann, H. Ong, and S. L. Scott, "The case for modular redundancy in large-scale high performance computing systems," in *Proc. IASTED Internat. Conf.*, vol. 641, p. 046, 2009. 13, 20, 21, 79, 125

[96] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *J. Paral. and Distrib. Comput.*, vol. 69, no. 4, pp. 410–416, 2009. 14, 15, 19, 41, 125, 153

[97] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, "Fault tolerant high performance computing by a coding approach," in *Proc. ACM SIGPLAN Symp. on Princ. and Pract. of Paral. Program.*, pp. 213–223, ACM, 2005. 14, 15, 41, 70, 125, 152, 153

[98] Z. Chen and J. Dongarra, "Numerically stable real number codes based on random matrices," in *Computational Science–ICCS 2005*, pp. 115–122, Springer, 2005. 14

[99] D. Hakkarinen, P. Wu, and Z. Chen, "Fail-stop failure algorithm-based fault tolerance for cholesky decomposition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1323–1335, 2015. 14

[100] Z. Chen and J. Dongarra, "Algorithm-based fault tolerance for fail-stop failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 12, pp. 1628–1641, 2008. 14, 19, 20, 93

[101] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. on Computers*, vol. 100, no. 6, pp. 518–528, 1984. 14, 15, 41, 64, 79, 125, 153

[102] J. Jou and J. A. Abraham, "Fault-tolerant matrix operations on multiple processor systems using weighted checksums," in *Proc.*

28th Annual Tech. Symp., pp. 94–101, International Society for Optics and Photonics, 1984. 14, 100

[103] P. Du, P. Luszczek, S. Tomov, and J. Dongarra, "Soft error resilient QR factorization for hybrid system with GPGPU," *J. of Computat. Sci.*, vol. 4, no. 6, pp. 457–464, 2013. 15, 50

[104] P. Wu, C. Ding, L. Chen, T. Davies, C. Karlsson, and Z. Chen, "Online soft error correction in matrix–matrix multiplication," *J. of Comput. Sci.*, vol. 4, no. 6, pp. 465–472, 2013. 15, 41

[105] C. Anfinson and F. Luk, "A linear algebraic model of algorithm-based fault tolerance," *IEEE Trans. on Computers*, vol. 37, no. 12, pp. 1599–1604, 1988. 17

[106] M. Hoemmen and M. A. Heroux, "Fault-tolerant iterative methods via selective reliability," in *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society*, vol. 3, p. 9, 2011. 18

[107] Z. Chen, "Online-abft: An online algorithm based fault tolerance scheme for soft error detection in iterative methods," in *ACM SIGPLAN Notices*, vol. 48, pp. 167–176, ACM, 2013. 18

[108] P. Du *et al.*, "Algorithm-based fault tolerance for dense matrix factorizations," *ACM SIGPLAN Notices*, vol. 47, no. 8, pp. 225–234, 2012. 19

[109] A. L. N. Reddy and P. Banerjee, "Algorithm-based fault detection for signal processing applications," *IEEE Transactions on Computers*, vol. 39, no. 10, pp. 1304–1308, 1990. 19

[110] Z. Gao, P. Reviriego, Z. Xu, X. Su, M. Zhao, J. Wang, and J. A. Maestro, "Fault tolerant parallel ffts using error correction codes and parseval checks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 769–773, 2016. 19

[111] H.-J. Wunderlich, C. Braun, and S. Halder, "Efficacy and efficiency of algorithm-based fault-tolerance on GPUs," in *IEEE Internat. On-Line Testing Symp., 2013*, pp. 240–243, IEEE, 2013. 19, 41

[112] L. Pilla, P. Rech, F. Silvestri, C. Frost, P. Navaux, M. S. Reorda, and L. Carro, "Software-based hardening strategies for neutron sensitive fft algorithms on gpus," *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1874–1880, 2014. 19

[113] Z. Chen, "Optimal real number codes for fault tolerant matrix operations," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, p. 29, ACM, 2009. 20, 93, 125

[114] Z. Chen, "Algorithm-based recovery for iterative methods without checkpointing," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, pp. 73–84, ACM, 2011. 20, 93

[115] A. Heinig, M. Engel, F. Schmoll, and P. Marwedel, "Improving transient memory fault resilience of an h. 264 decoder," in *Embedded*

Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on, pp. 121–130, IEEE, 2010. 20, 73

[116] I. Polian, B. Becker, M. Nakasato, S. Ohtake, and H. Fujiwara, "Low-cost hardening of image processing applications against soft errors," in *Proc. IEEE Int. Symp. Defect and Fault Tol, in VLSI Syst., DFT'06*, pp. 274–279, IEEE, 2006. 20, 25, 73

[117] K. Lee, A. Shrivastava, I. Issenin, N. Dutt, and N. Venkatasubramanian, "Mitigating soft error failures for multimedia applications by selective data protection," in *Proc. ACM Int. Conf. on Comp., Arch. and Synth. for Embed. Syst.*, pp. 411–420, ACM, 2006. 20, 25, 73

[118] Y. Mu, W. Liu, and S. Yan, "Video de-fencing," *IEEE Trans. on Circ. and Syst. for Video Technol.*, vol. 24, no. 7, pp. 1111–1121, 2014. 21, 24

[119] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "KULFI." https://github.com/soarlab/KULFI/. 22, 65

[120] MIT, "StarCluster." http://star.mit.edu/cluster/. 23, 85, 108

[121] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: applications to image and text data," in *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250, ACM, 2001. 24

[122] N. Kontorinis *et al.*, "Statistical framework for video decoding complexity modeling and prediction," *IEEE Trans. on Circ. and Syst. for Video Technol.*, vol. 19, no. 7, pp. 1000–1013, 2009. 24

[123] Y. Andreopoulos *et al.*, "A new method for complete-to-overcomplete discrete wavelet transforms," in *Proc. 14th IEEE Int. Conf. on Digital Signal Process., DSP 2002*, vol. 2, pp. 501–504, IEEE, July 2002. 24

[124] Y. Andreopoulos *et al.*, "A local wavelet transform implementation versus an optimal row-column algorithm for the 2d multilevel decomposition," in *Proc. IEEE Int. Conf. on Image Process., ICIP 2001*, vol. 3, pp. 330–333, IEEE, October 2001. 24

[125] A. Chadha and Y. Andreopoulos, "Region-of-interest retrieval in large image datasets with Voronoi VLAD," in *Computer Vision Systems ICVS*, pp. 218–227, Springer, 2015. 24

[126] V. B. Kleeberger, C. Gimmler-Dumont, C. Weis, A. Herkersdorf, D. Mueller-Gritschneder, S. R. Nassif, U. Schlichtmann, and N. Wehn, "A cross-layer technology-based study of how memory errors impact system resilience," *IEEE Micro*, vol. 33, no. 4, pp. 46–55, 2013. 25, 78

[127] D. Anastasia, *Software-based Approximate Computation Of Signal Processing Tasks*. PhD thesis, UCL (University College London), 2012. 27

[128] M. A. Anam and Y. Andreopoulos, "Throughput scaling of convolution for error-tolerant multimedia applications," *IEEE Transac-*

tions on Multimedia, vol. 14, no. 3, pp. 797–804, 2012. 27, 151, 152

[129] D. Anastasia and Y. Andreopoulos, "Throughput-distortion computation of generic matrix multiplication: Toward a computation channel for digital signal processing systems," *IEEE Trans. on Signal Processing*, vol. 60, no. 4, pp. 2024–2037, 2012. 27, 32, 34, 38, 39, 151, 152

[130] D. Anastasia and Y. Andreopoulos, "Software designs of image processing tasks with incremental refinement of computation," *IEEE Trans. on Image Processing*, vol. 19, no. 8, pp. 2099–2114, 2010. 27, 32, 38, 39

[131] I. Anarado, M. A. Anam, F. Verdicchio, and Y. Andreopoulos, "Mitigating silent data corruptions in integer matrix products: Toward reliable multimedia computing on unreliable hardware," *IEEE Transactions on Circuit and System for Video Technology*, 2016. 29, 70

[132] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. IEEE Int. Symp. Code Generation and Optimization, CGO 2004*, pp. 75–86, IEEE, 2004. 65, 161

[133] V. Narayanan and Y. Xie, "Reliability concerns in embedded system designs," *Computer*, vol. 39, no. 1, pp. 118–120, 2006. 73

[134] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-

time systems," in *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pp. 219–228, IEEE, September 2002. 73

[135] Y. Ikenaga, M. Nomura, S. Suenaga, H. Sonohara, Y. Horikoshi, T. Saito, Y. Ohdaira, Y. Nishio, T. Iwashita, M. Satou, *et al.*, "A 27% active-power-reduced 40-nm CMOS multimedia soc with adaptive voltage scaling using distributed universal delay lines," *IEEE J. Solid-State Circ.*, vol. 47, no. 4, pp. 832–840, 2012. 76

[136] Intel, "Intel power gadget 3.0 (windows 64-bit)," 2014. 77

[137] J. Sloan, R. Kumar, and G. Bronevetsky, "An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance," in *Proc. IEEE/IFIP Int. Conf. on Depend. Syst. and Net. (DSN'13)*, pp. 1–12, IEEE, 2013. 78

[138] F. J. Kurdahi, A. Eltawil, K. Yi, S. Cheng, and A. Khajeh, "Low-power multimedia system design by aggressive voltage scaling," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 18, no. 5, pp. 852–856, 2010. 78

[139] A. Suresh and J. Sartori, "Automated algorithmic error resilience for structured grid problems based on outlier detection," in *Proc. Annual IEEE/ACM Int. Symp. on Code Gen. and Opt.*, p. 240, ACM, February 2014. 78

[140] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo, "Intel AVX: New frontiers in performance improvements and energy efficiency," *Intel White paper*, 2008. 84

[141] S. Ohshima, K. Kise, T. Katagiri, and T. Yuba, "Parallel processing of matrix multiplication in a cpu and gpu heterogeneous environment," in *High Performance Computing for Computational Science-VECPAR 2006*, pp. 305–318, Springer, 2007. 86

[142] I. Anarado, M. A. Anam, D. Anastasia, F. Verdicchio, and Y. Andreopoulos, "Highly-reliable integer matrix multiplication via numerical packing," in *Proc. 19th IEEE Internat. On-Line Testing Sympos. (IOLTS'13)*, 2013. 88

[143] Intel, "Intel intrinsics guide." http://software.intel.com/sites/landingpage/IntrinsicsGuide/. 91, 152

[144] K. A. (Intel), "Recommended settings for calling intel MKL routines from multi-threaded applications." https://software.intel.com/en-us/articles/recommended-settings-for-calling-intel-mkl-routines-from-multi-threaded-applications, 2011. 102, 104

[145] C. Y. (Intel), "Intel MKL 10.x threading." https://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-intel-mkl-100-threading, 2010. 102, 104

[146] D. P. Ellis, C. V. Cotton, and M. I. Mandel, "Cross-correlation of beat-synchronous representations for music similarity," in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2008. ICASSP 2008.*, pp. 57–60, IEEE, 2008. 105, 106

[147] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011. 105

[148] K. Seyerlehner, G. Widmer, and T. Pohle, "Fusing block-level features for music similarity estimation," in *Proc. of the 13th Int. Conference on Digital Audio Effects (DAFx-10)*, pp. 225–232, 2010. 105

[149] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3304–3311, IEEE, 2010. 107, 108

[150] G. Amato, P. Bolettieri, F. Falchi, and C. Gennaro, "Large scale image retrieval using vector of locally aggregated descriptors," in *Similarity Search and Applications*, pp. 245–256, Springer, 2013. 107

[151] G. Tolias, Y. Avrithis, and H. Jégou, "To aggregate or not to aggregate: Selective match kernels for image search," in *2013 IEEE International Conference on Computer Vision (ICCV)*, pp. 1401–1408, IEEE, 2013. 108

[152] Amazon, "Amazon EC2 pricing." https://aws.amazon.com/ec2/pricing/. 109

[153] Amazon, "New EC2 spot instance termination notices." https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notices/. 109

[154] Amazon, "Amazon EC2 spot price history." http://docs.amazonaws.cn/cli/latest/reference/ec2/describe-spot-price-history.html. 110

[155] D. G. Murray and S. Hand, "Spread-spectrum computation," in *Proc. USENIX 4th Conf. Hot Top. in Syst. Dependab.*, pp. 5–9, 2008. 125, 152, 153

[156] I. Anarado and Y. Andreopoulos, "Core failure mitigation in integer sum-of-product computations on cloud computing systems," *IEEE Transactions on Multimedia*, vol. 18, no. 4, pp. 789–801, 2016. 125

[157] M. Anam, I. Anarado, and Y. Andreopoulos, "Generalized numerical entanglement for reliable linear, sesquilinear and bijective operations on integer data streams," *IEEE Trans. on Emerging Topics In Computing*, 2016. 125

[158] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, pp. 325–336, VLDB Endowment, 2013. 169