Generalized Numerical Entanglement For Reliable Linear, Sesquilinear And Bijective Operations On Integer Data Streams

Mohammad Ashraful Anam, Ijeoma Anarado and Yiannis Andreopoulos, *Senior Member, IEEE*

***Abstract*—We propose a new technique for the mitigation of fail-stop failures and/or silent data corruptions (SDCs) within linear, sesquilinear or bijective (LSB) operations on M integer data streams ($M \geq 3$). In the proposed approach, the M input streams are linearly superimposed to form M *numerically entangled* integer data streams that are stored in-place of the original inputs, i.e., no additional (aka. "checksum") streams are used. An arbitrary number of LSB operations can then be performed in M processing cores using these entangled data streams. The output results can be extracted from any $M - K$ entangled output streams by additions and arithmetic shifts, thereby mitigating K fail-stop failures ($K \leq \left\lfloor \frac{M-1}{2} \right\rfloor$), or detecting up to K SDCs per M-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 LSB operations. Our proposal is validated within an 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 0.75% 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 LSB 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.**

***Index Terms*—linear operations, sum-of-products, fault tolerance, silent data corruptions, numerical entanglement**

I. Introduction

MODERN computing clusters exhibit decreasing mean-time-to-failure characteristics due to their composition from diverse commercial-off-the-shelf cores with varying levels of reliability, combined with the aggressive voltage/frequency scaling applied on each core [1] and the varying degrees of reliability of runtime components [2]. Therefore, it is now becoming imperative for distributed computing systems to provide for: *(i)* fail-stop failure mitigation [3], i.e., mitigate cases where one (or more) of their processor cores becomes unresponsive (or does not return the results within a predetermined time-to-completion constraint); *(ii)* detection and mitigation of silent data corruptions (SDCs) [4], i.e., detect and mitigate bit flips in memory or disk drives that do not interfere with the algorithmic or software execution flow, thereby causing data corruptions that remain undetected by runtime or hardware checkpointing methods.

The authors are with the Electronic and Electrical Engineering Department, University College London, Roberts Building, Torrington Place, London, WC1E 7JE, U.K.; tel. +442076797303, fax. +442073889325, email: {mohammad.anam.10, ijeoma.anarado.12, i.andreopoulos}@ucl.ac.uk. This work was supported by EPSRC, grant EP/M00113X/1. The work of I. Anarado was supported by the Federal Government of Nigeria under the PRESSID Scheme.

Applications that are particularly prone to fail-stop failures and SDCs include distributed systems like grid computing [5], sensor data processing systems [6], webpage, or multimedia retrieval applications [7], relevance ranking [8], object or face recognition in images [2], [9], encryption and data compression [10], [11], financial computing [12], machine learning and security applications [13], etc. The compute- and memory-intensive parts of these applications comprise linear, sesquilinear (also known as "one-and-half linear") and bijective operations, collectively termed as "LSB" operations in this paper. These operations are typically performed using single or double-precision floating-point inputs or, for high-performance systems requiring exact reproducibility and reduced hardware complexity, 32-bit or 64-bit integer or fixed-point inputs. Thus, ensuring robust recovery from fail-stop failures for applications comprising integer LSB operations is of paramount importance.

A. Summary of Prior Work

Fail-stop failures and SDCs in parallel LSB routines are currently mitigated via *roll-back* or *roll-forward* methods. Roll-back methods are based on periodic checkpointing and recomputation if failures or SDCs are detected. Roll-back techniques essentially comprise methods for 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 [4], [14]–[17]. Several BER studies show that, depending on the desired level of resilience to core failures, substantial resources may be spent on checkpointing and recomputation. This has been identified as a major challenge for future exascale systems [4], [18], [19].

Roll-forward methods ensure result recovery from the functioning processor cores *without* recomputation when fail-stop failures or SDCs are detected in the system. Examples include: forward error recovery (FER) methods that recover the lost data from pre-established (and stored) input data checksum relationships without repeating computations [20]–[23], algorithm based fault tolerance [24] [25] and dual modular redundancy (DMR) [14], [26] or similar techniques [27] [28] [29]. Computations examined in such proposals include matrix products [20], matrix factorization [30], digital filters [25] and iterative solvers [31]. The overarching concept of these methods within the context of LSB data stream processing is the production of checksum data streams in ways such that the performed computation can be applied on the checksum elements alongside the original integer data streams. These additional data streams can then be used for FER by solving a system of linear equations if core failures are detected. Therefore, all roll-forward approaches incur significant overhead due to the storage and processing of the checksum data streams. Methods to reduce the amount of additional checksum data streams have been investigated [25], but the requirement of additional cores for checksum processing decreases the achievable peak performance, as less cores are dedicated to actual input-data computations. Nevertheless, for fail-stop failure resilience and SDC mitigation, roll-forward methods are preferable to roll-back methods as they achieve higher reliability and can immediately mitigate the effect of failures without service interruption due to checkpointing and recomputation.

B. Contribution

We propose a new FER method for linear, sesquilinear (also known as one-and-half linear) or bijective operations performed in integer data streams. Examples of such operations are element-by-element additions and multiplications, inner and outer vector products, sum-of-squares and permutation operations. They are the building blocks of algorithms of foundational importance, such as: matrix multiplication [24], [32], convolution/cross-correlation [33], template matching for search algorithms [34], covariance calculations [9], integer-to-integer transforms [35], and permutation-based encoding and scrambling systems [10], [11], which form the core of the applications discussed earlier. Because our method performs linear superpositions of input streams onto each other, it "entangles" input streams together and we term it as *numerical entanglement*. Our approach guarantees mitigation of any K fail-stop failures in M-core data stream processing ($M \geq 3$, $K \leq \left\lfloor \frac{M-1}{2} \right\rfloor$), or detection of K SDCs occurring in each M-tuple of outputs. Thus, it generalizes our earlier work on single-core fail-stop failure mitigation [36]. Importantly, generalized numerical entanglement does not generate any additional "checksum" or duplicate streams and does not depend on the specifics of the LSB operation performed. Therefore, it is found to be extremely efficient in comparison to the equivalent checksum-based methods that require the generation and processing of K additional checksum streams.

C. Paper Organization

In Section II, we introduce checksum-based methods for fail-stop failure mitigation and SDC detection/correction in numerical stream processing. In Section III we propose the concept of generalized numerical entanglement and demonstrate its inherent reliability for LSB processing of integer streams. Section IV presents the complexity of numerical entanglements within integer linear or sesquilinear operations. Section V presents experimental comparisons, followed by Section VI that presents the application of the method for the detection of SDCs. Finally Section VII presents some concluding remarks.

II. Checksum-based Methods versus Numerical Entanglement

Consider M input streams of integers, each comprising N_{in} samples[1] ($M \geq 3$):

$$\mathbf{c}_m = \begin{bmatrix} c_{m,0} & \dots & c_{m,N_{\text{in}}-1} \end{bmatrix}, \; 0 \leq m < M. \tag{1}$$

These may be the elements of M rows of a matrix of integers, or a set of M input integer streams of data to be operated upon with an integer kernel $\mathbf{g}$. This operation is performed by:

$$\forall m \in \{0, \dots, M-1\} : \; \mathbf{d}_m = \mathbf{c}_m \, \text{op} \, \mathbf{g}$$
$$\text{op} \in \left\{ +, -, \times, \langle .,. \rangle, \otimes, \begin{pmatrix} \mathfrak{I} \\ \mathfrak{G} \end{pmatrix}, \star \right\} \tag{2}$$

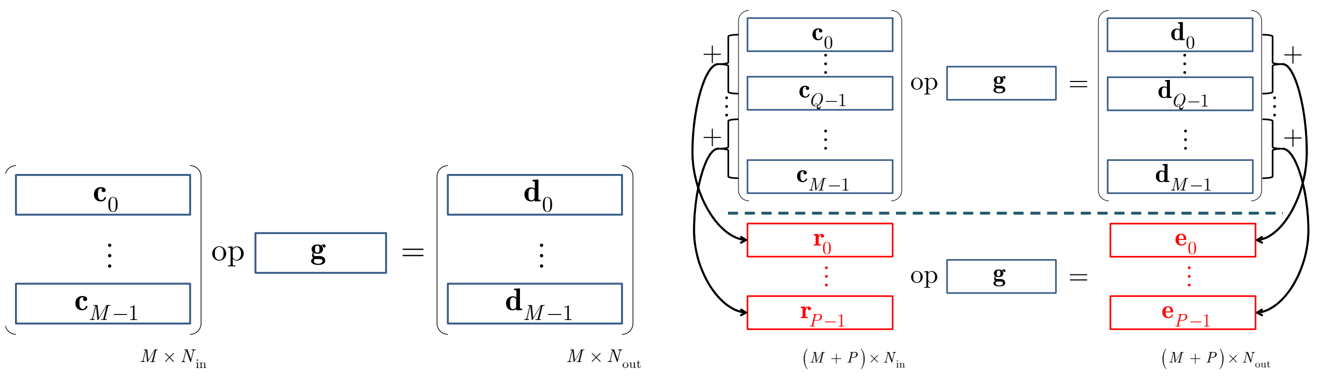
with $\mathbf{d}_m$ the mth vector of output results (containing N_{out} values) and op any LSB operator such as element-by-element addition/subtraction/multiplication, inner/outer product, permutation[2] (i.e., bijective mapping from the sequential index set $\mathfrak{I}$ to index set $\mathfrak{G}$ corresponding to $\mathbf{g}$) and circular convolution or cross-correlation with $\mathbf{g}$. An illustration of the application of (2) is given in Fig. 1(a). Beyond the single LSB operator indicated in (2), we can also assume *series* of such operators applied consecutively in order to realize higher-level algorithmic processing, e.g., multiple consecutive additions, subtractions and scaling operations with pre-established kernels, followed by circular convolutions and permutation operations. Conversely, the input data streams can also be left in their native state (i.e., stored in memory), e.g., if op $= \{\times\}$ and $\mathbf{g} = 1$.

A. Checksum-based Methods

In their original (or "pure") form, the input data streams of (1) are uncorrelated and one input or output element cannot be used for the recovery of another without inserting

[1]Notations: Boldface uppercase and lowercase letters indicate matrices and vectors, respectively; the corresponding italicized lowercase indicate their individual elements, e.g. $\mathbf{A}$ and $a_{m,n}$; calligraphic uppercase letters indicate operators; $\mathbb{N}^\star$ is the set of natural numbers excluding zero; $\hat{d}$ denotes the recovered value of d after disentanglement; all indices are integers. Basic operators: $\lfloor a \rfloor$ is the largest integer that is smaller or equal to a (floor operation); $\lceil a \rceil$ is the smallest integer that is larger or equal to a (ceiling operation); $a \ll b$ and $a \gg b$ indicate left and right arithmetic shift of integer a by b bits with truncation occurring at the most-significant or least significant bit, respectively; $a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b$ is the modulo operation.

[2]We remark that we consider LSB operations that are *not* data-dependent, e.g., permutations according to fixed index sets as in the Burrows-Wheeler transform [11].



(a) Conventional processing (b) FER using K checksum streams

Figure 1. (a) kernel $\mathbf{g}$ applied to M streams of input integers via LSB operator op; (b) corresponding application of $\mathbf{g}$ to M input streams, as well as to K checksum input streams, $\mathbf{r}_0,\ldots,\mathbf{r}_{K-1}$, used for forward error recovery. The K checksum input streams are produced via linear combinations with weight vectors $\mathbf{w}_0,\ldots,\mathbf{w}_{K-1}$.

some form of coding or redundancy. This is conventionally achieved via checksum-based methods [24], [37]–[42] and is pictorially illustrated in Fig. 1(b). Specifically, K *additional* input streams are created, which comprise *checksums* of the original inputs:

$$\mathbf{r}_k = \begin{bmatrix} r_{k,0} & \ldots & r_{k,N_{\text{in}}-1}\end{bmatrix},\; 0 \le k < K. \quad (3)$$

by using different checksum relationships with predefined weight vectors $\mathbf{w}_k = \begin{bmatrix} w_{k,0} & \ldots & w_{k,M-1}\end{bmatrix}$ [38], [39], [41], [42] $(0 \le n < N_{\text{in}},\ 0 \le k < K)$:

$$\forall k,n:\; r_{k,n} = \sum_{m=0}^{M-1} w_{k,m} c_{m,n}. \quad (4)$$

The LSB processing is then performed in all input streams $\mathbf{c}_0,\ldots,\mathbf{c}_{M-1}$ and checksum input streams $\mathbf{r}_0,\ldots,\mathbf{r}_{K-1}$ (each running on a different core) by:

$$\begin{bmatrix} \mathbf{d}_0 \\ \vdots \\ \mathbf{d}_{M-1} \\ \mathbf{e}_0 \\ \vdots \\ \mathbf{e}_{K-1} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_{M-1} \\ \mathbf{r}_0 \\ \vdots \\ \mathbf{r}_{K-1} \end{bmatrix} \text{op}\, \mathbf{g}. \quad (5)$$

Any K fail-stop core failures in the group of $M + K$ cores executing (5) can be recovered from the remaining M output streams [38], [39], [41], [42]. In addition, any K SDCs in any $(M+K)$-tuple of outputs at position n, $\begin{bmatrix} d_{0,n} & \ldots & d_{M-1,n} & e_{0,n} & \ldots & e_{K-1,n}\end{bmatrix}^{\text{T}}$, can be detected, and $K-1$ SDCs can be detected *and* corrected without recomputation [38], [39], [41], [42]. Evidently, this FER capability comes at the cost of using K additional cores to process the checksum input streams $\mathbf{r}_0,\ldots,\mathbf{r}_{K-1}$.

B. Generalized Numerical Entanglement

The proposed method of generalized numerical entanglement mixes the integer inputs prior to processing using linear superposition, and ensures the results can be recovered via a mixture of shift-add operations. Considering M ($M \ge 3$) input streams $\mathbf{c}_m$, $0 \le m < M$ (each comprising N_{in} integer samples), the nth element of the mth entangled stream, denoted by $\epsilon_{m,n}$ $(0 \le n < N_{\text{in}})$, comprises the superposition of $K+1$ distinct input elements $c_{x_0,n},\ldots,c_{x_K,n}$, with $0 \le x_0,\ldots,x_K < M$ and $K \le \left\lfloor \frac{M-1}{2} \right\rfloor$. The LSB operation op with kernel $\mathbf{g}$ is carried out with M independent cores utilizing the M entangled input streams directly, thereby producing the entangled output streams $\boldsymbol{\delta}_m$ (each comprising N_{out} integer samples). These can be disentangled to recover the final results $\hat{\mathbf{d}}_m$ and K fail-stop failures in the M processor cores can be mitigated from the results of the remaining $M-K$ cores. In addition, any K SDCs in any M-tuple of outputs at in-stream position n, $\begin{bmatrix} \delta_{0,n} & \ldots & \delta_{M-1,n}\end{bmatrix}$, can be detected, and $K-1$ SDCs can be detected and corrected without recomputation.

Importantly, the complexity of entanglement, disentanglement (extraction) and recovery does not depend on the complexity of the operator op, or on the length of the kernel (operand) $\mathbf{g}$. This is because the entangled inputs can be written in-place and no additional storage or additional operations are needed during the execution of the actual operation. That is, the entanglement of M input streams of N_{in} samples each, results in M entangled streams of N_{in} samples each, and each processing core uses the same operator op and operand $\mathbf{g}$. Therefore, if entanglement, processing, and disentanglement is performed in batch mode, the entire processing stage remains agnostic to the fact that the inputs comprise numerically-entangled inputs rather than the original stream values. Our approach is also suitable for stream processors with the entanglement/disentanglement stage applied during the data input/output. The only detriment of numerical entanglement is that the dynamic range of the entangled inputs $\boldsymbol{\epsilon}_m$ is increased in comparison to the original inputs $\mathbf{c}_m$. Therefore, under 32-bit or 64-bit integer representations, the maximum output dynamic range that can be supported under an entangled representation is smaller in comparison to the dynamic range of conventional processing. However, as it will be demonstrated in this paper, this decrease depends on the number of jointly-entangled inputs, M, and the number of fail-stop failures or SDCs (K) the system should be able to handle. Therefore, one can form the appropriate tradeoff between the reliability to fail-stop failures and SDCs and the increase in dynamic range. A summary of the features of numerical entanglement in comparison to checksum-based FER and DMR is presented in Table I.

III. Numerical Entanglement for Fail-Stop Reliability in LSB Operations

We first describe numerical entanglement for mitigation of one fail-stop failure [36]. This provides the basis for the generalization to K-failure mitigation in M data streams, with $K \leq \left\lfloor \frac{M-1}{2} \right\rfloor$.

A. Simplest Form of Numerical Entanglement

The simplest form of entanglement takes place with $(M, K) = (3, 1)$. In this case, each triplet of input samples of the three integer streams, $c_{0,n}$, $c_{1,n}$ and $c_{2,n}$, $0 \leq n < N_{\text{in}}$, produces an entangled triplet of inputs via the superposition operations [36]:

$$\begin{aligned} \epsilon_{0,n} &= \mathcal{S}_l\{c_{2,n}\} + c_{0,n} \\ \epsilon_{1,n} &= \mathcal{S}_l\{c_{0,n}\} + c_{1,n} \\ \epsilon_{2,n} &= \mathcal{S}_l\{c_{1,n}\} + c_{2,n} \end{aligned} \tag{6}$$

where:

$$\mathcal{S}_b\{a\} \equiv \begin{cases} (a \ll b), & \text{if } b \geq 0 \\ [a \gg (-b)], & \text{if } b < 0 \end{cases} \tag{7}$$

is the left or right arithmetic shift of a by b bits. If we assume that the utilized integer representation comprises w bits, the results of the l-bit left-shift operations of (6) must be upper-bounded by w to avoid overflow. Therefore, if the dynamic range of the input streams $\mathbf{c}_0$, $\mathbf{c}_1$, $\mathbf{c}_2$ is $l + k$ bits:

$$2l + k \leq w. \tag{8}$$

The values for l and k are chosen such that $l+k$ is maximum within the constraint of (8) and $k \leq l$. Each entangled input stream $\boldsymbol{\epsilon}_m$ ($0 \leq m < M$) is converted to the entangled output stream[3] $\boldsymbol{\delta}_m$ (which contains N_{out} values) via the application of LSB operations:

$$\forall m: \; \boldsymbol{\delta}_m = (\boldsymbol{\epsilon}_m \,\text{op}\, \mathbf{g}). \tag{9}$$

A conceptual illustration of the entangled outputs after (6) and (9) is given in Fig. 2. As a practical instantiation of (6), we can set $w = 32$, $l = 11$ and $k = 10$ in a signed 32-bit integer configuration. More broadly, regarding any $(M, 1)$ entanglement pattern, the allowable input dynamic range is given in Remark 4 and Remark 5 and practical examples given in Table II.

[3]For the particular cases of: op $\in \{+, -\}$, $\mathbf{g}$ must also be entangled with itself via: $g_n \leftarrow \mathcal{S}_l\{g_n\} + g_n$, in order to retain the homomorphism of the performed operation.

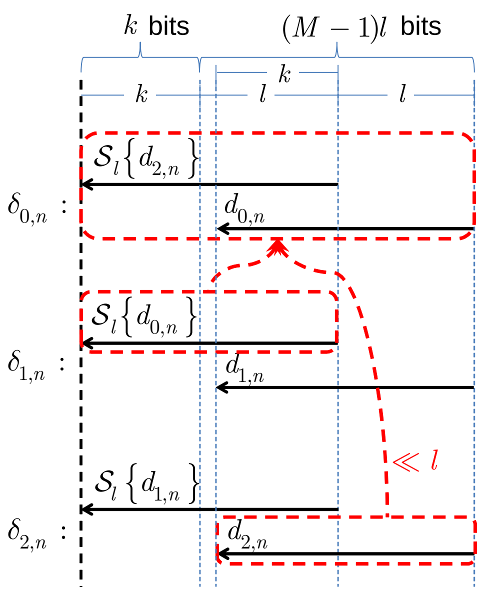


Figure 2. Illustration of three entangled outputs after integer LSB processing. The solid arrows indicate the maximum attainable dynamic range of each output $d_{0,n}$, $d_{1,n}$ and $d_{2,n}$. The dotted rectangles and arrows illustrate that the contents of entangled output $\delta_{0,n}$ are contained within the two other entangled outputs.

For each position n, $0 \leq n < N_{\text{out}}$, we can disentangle the outputs by first creating the $2w$-bit temporary variable[4]

$$d_{\text{temp}} = \delta_{2,n} - \mathcal{S}_l\{\delta_{1,n}\} \tag{10}$$

and then recover the outputs by:

$$\begin{aligned} \hat{d}_{2,n} &= \mathcal{S}_{-2(w-l)}\left\{\mathcal{S}_{2(w-l)}\left\{d_{\text{temp}}\right\}\right\} \\ \hat{d}_{0,n} &= \mathcal{S}_{-2l}\left\{-\left(d_{\text{temp}} - \hat{d}_{2,n}\right)\right\} \\ \hat{d}_{1,n} &= \delta_{1,n} - \mathcal{S}_l\left\{\hat{d}_{0,n}\right\} \end{aligned} \tag{11}$$

The correctness of (11) can be verified by using Fig. 2 to check the results of all steps. Notice that (10) and (11) do not use $\delta_{0,n}$. This is a crucial element of this approach: since $\hat{d}_{0,n}$, $\hat{d}_{1,n}$ and $\hat{d}_{2,n}$ were derived without using $\delta_{0,n}$, all outputs are recovered even under the complete loss of one entangled stream. We are able to do this because, for every n, $0 \leq n < N_{\text{out}}$, $\delta_{1,n}$ and $\delta_{2,n}$ contain $\hat{d}_{0,n}$ and $\hat{d}_{2,n}$, via which we can recreate $\delta_{0,n}$ if the latter is not available due to a fail-stop failure. This link is pictorially illustrated in Fig. 2. Since the entangled pattern is cyclically-symmetric, it is straightforward to demonstrate that mitigation of any single stream loss (out of the three) is possible following the same approach. Moreover, if all entangled outputs $\left[\delta_{0,n} \;\; \delta_{1,n} \;\; \delta_{2,n}\right]^{\text{T}}$ are obtained per position n, $0 \leq n < N_{\text{out}}$, any single SDC within them can be detected by recovering $\left[\hat{d}_{0,n} \;\; \hat{d}_{1,n} \;\; \hat{d}_{2,n}\right]^{\text{T}}$ in two different ways (e.g., without using $\delta_{0,n}$ and $\delta_{1,n}$) and cross-validating the recovered results. Therefore, l bits

[4]The disentanglement operation can be performed in w-bit representation if the operation of (10) is performed in two subsets of w bits as, as discussed later on in Remark 1.

Table I
SUMMARY OF FEATURES OF DIFFERENT METHODS FOR K-FAILURE MITIGATION WITHIN M STREAMS UNDER A w-BIT REPRESENTATION.

Method / Feature	Checksum-based FER [23], [24], [37], [43], [44]	Dual Modular Redundancy [21]	Proposed Generalized Numerical Entanglement
In-place storage	No	No	Yes
% of redundant LSB computations	$\frac{K}{M} \times 100\%$	100%	0%
Reduction of output bitwidth supported	$K = 1$: $\lceil \log_2 M \rceil$ $K > 1$: see Table II	0 bits	$K = 1$: $\lceil \frac{w}{M} \rceil$ $K > 1$: see Table II
Failure Mitigation	$K \leq \lfloor \frac{M-1}{2} \rfloor$ failures in $M + K$ streams, $M \geq 3$	$K = 1$ failure in $M = 2$ streams	$K \leq \lfloor \frac{M-1}{2} \rfloor$ failures in M streams, $M \geq 3$
Practical execution overhead	More than $\frac{K}{M} \times 100\%$	More than 100%	0.75% to 37.23% (decreases with increased operand length)

of dynamic range are being used *within* each entangled input/output in order to achieve recovery from *any single fail-stop failure* or *detection of any single SDC* in the computation of $\delta_{0,n}$, $\delta_{1,n}$ and $\delta_{2,n}$.

B. Generalized Numerical Entanglement in Groups of Five Inputs ($M = 5$, $K = 2$)

Generalizing numerical entanglement into a multitier process guarantees mitigation of more than a single fail-stop failure in the processing of M entangled streams with M cores, or detection and mitigation of one or more SDCs. In this subsection, we illustrate the basic case of mitigation of two fail-stop failures in five input streams, or detection and correction of one SDC within the quintuple of outputs at any in-stream position n.

1) Entanglement: In the two-tier entanglement with $M = 5$, each quintuple of input samples from the five streams, $c_{0,n}, \ldots, c_{4,n}$, $0 \leq n < N_{\text{in}}$, produces the quintuple of entangled samples, $\epsilon^{(2)}_{0,n}, \ldots, \epsilon^{(2)}_{4,n}$, via the following two-tier superposition operations:

$$\begin{bmatrix} \epsilon^{(1)}_{0,n} \\ \epsilon^{(1)}_{1,n} \\ \epsilon^{(1)}_{2,n} \\ \epsilon^{(1)}_{3,n} \\ \epsilon^{(1)}_{4,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \mathcal{S}_{l_1} \\ \mathcal{S}_{l_1} & 1 & 0 & 0 & 0 \\ 0 & \mathcal{S}_{l_1} & 1 & 0 & 0 \\ 0 & 0 & \mathcal{S}_{l_1} & 1 & 0 \\ 0 & 0 & 0 & \mathcal{S}_{l_1} & 1 \end{bmatrix} \begin{bmatrix} c_{0,n} \\ c_{1,n} \\ c_{2,n} \\ c_{3,n} \\ c_{4,n} \end{bmatrix} \tag{12}$$

and

$$\begin{bmatrix} \epsilon^{(2)}_{0,n} \\ \epsilon^{(2)}_{1,n} \\ \epsilon^{(2)}_{2,n} \\ \epsilon^{(2)}_{3,n} \\ \epsilon^{(2)}_{4,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \mathcal{S}_{l_2} \\ \mathcal{S}_{l_2} & 1 & 0 & 0 & 0 \\ 0 & \mathcal{S}_{l_2} & 1 & 0 & 0 \\ 0 & 0 & \mathcal{S}_{l_2} & 1 & 0 \\ 0 & 0 & 0 & \mathcal{S}_{l_2} & 1 \end{bmatrix} \begin{bmatrix} \epsilon^{(1)}_{0,n} \\ \epsilon^{(1)}_{1,n} \\ \epsilon^{(1)}_{2,n} \\ \epsilon^{(1)}_{3,n} \\ \epsilon^{(1)}_{4,n} \end{bmatrix} \tag{13}$$

with

$$3l_1 + k_1 \leq l_2 + k_2 \tag{14}$$
$$2l_2 + k_2 \leq w. \tag{15}$$

and the maximum dynamic range supported for each signed input, $c_{0,n}, \ldots, c_{4,n}$, being proportional to $2l_1 + k_1$ bits.

The values for l_2 and k_2 are chosen such that $2l_2 + k_2$ is maximum within the constraint of (15) and $k_2 \leq l_2$. Similarly, the values for l_1 and k_1 are chosen such that $3l_1 + k_1$ is maximum within the constraint of (14) and $k_1 \leq l_1$. Via the application of LSB operations of (9), $\boldsymbol{\epsilon}^{(2)}_m = \left[\epsilon^{(2)}_{m,0} \ \ldots \ \epsilon^{(2)}_{m,N_{\text{in}}}\right]$ $(0 \leq m < M)$ are converted to the entangled outputs, $\boldsymbol{\delta}^{(2)}_m = \left[\delta^{(2)}_{m,0} \ \ldots \ \delta^{(2)}_{m,N_{\text{out}}}\right]$. A conceptual illustration of the entangled outputs after (13) and (9) is given in Fig. 3. Similarly as before, $l_1 + l_2$ bits of dynamic range are sacrificed in order to achieve recovery from failures. Since it is assumed that the dynamic range of the inputs does not exceed $2l_1 + k_1$ bits, the entangled representation is contained within $2l_2 + k_2$ bits and never overflows. As a practical instantiation of (13), we can set $w = 32$, $l_1 = 6$, $l_2 = 11$, $k_1 = 3$ and $k_2 = 10$ in a signed 32-bit integer configuration.

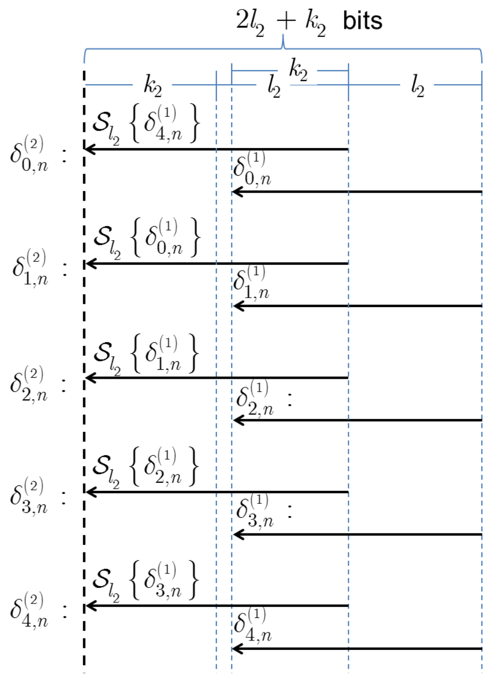
We now describe the disentanglement and recovery process in case of any two failures. The reader can also consult Fig. 3 in order to verify the results of all the presented steps.

2) Disentanglement: The proposed method mitigates $K \in \{1, 2\}$ failures. Equivalently, if all M 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 $\left[\delta^{(2)}_{0,n} \ \ldots \ \delta^{(2)}_{4,n}\right]^{\mathrm{T}}$.

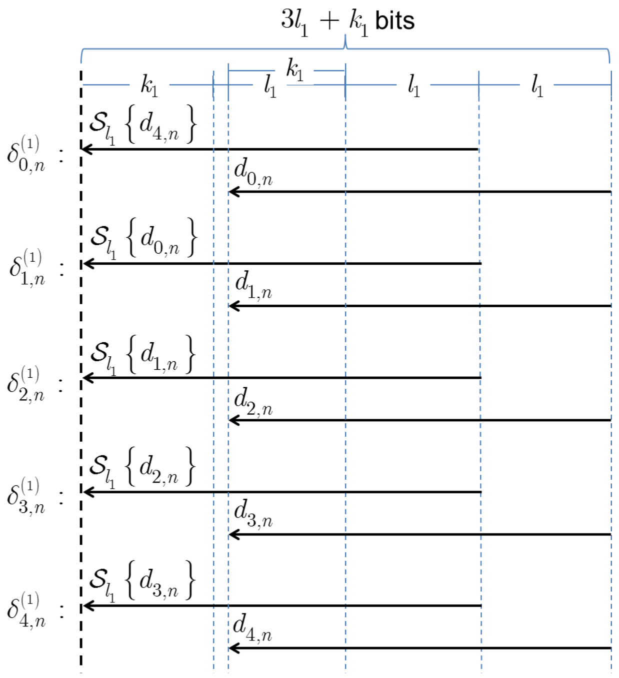
$\underline{\text{For the case of loss of a single stream}}$ $\boldsymbol{\delta}^{(2)}_r$, $0 \leq r < M$, due to a single fail-stop failure, we recover the outputs of the first tier by $(0 \leq n < N_{\text{out}})$:

$$\begin{aligned} d^{(1)}_{\text{temp}} &= \delta^{(2)}_{(r+2)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\delta^{(2)}_{(r+1)\text{mod}5,n}\right\} \\ \hat{\delta}^{(1)}_{(r+2)\text{mod}5,n} &= \mathcal{S}_{-2(w-l_2)}\left\{\mathcal{S}_{2(w-l_2)}\left\{d^{(1)}_{\text{temp}}\right\}\right\} \\ \hat{\delta}^{(1)}_{r,n} &= \mathcal{S}_{-2l_2}\left\{-\left(d^{(1)}_{\text{temp}} - \hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}\right)\right\} \\ \hat{\delta}^{(1)}_{(r+1)\text{mod}5,n} &= \delta^{(2)}_{(r+1)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\hat{\delta}^{(1)}_{r,n}\right\} \\ \hat{\delta}^{(1)}_{(r+3)\text{mod}5,n} &= \delta^{(2)}_{(r+3)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}\right\} \\ \hat{\delta}^{(1)}_{(r+4)\text{mod}5,n} &= \delta^{(2)}_{(r+4)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\hat{\delta}^{(1)}_{(r+3)\text{mod}5,n}\right\} \end{aligned} \tag{16}$$

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



(a) Outputs of LSB processing



(b) Contents of $\delta^{(1)}_{0,n},\ldots,\delta^{(1)}_{4,n}$ showing the original outputs $d_{0,n},\ldots,d_{4,n}$

Figure 3. Illustration of entanglement of five outputs after integer LSB processing: (a) second tier of superimposed outputs; (b) first tier of superimposed outputs, showing the original output values $d_{0,n},\ldots,d_{4,n}$ that are entangled within.

$$\begin{aligned}
d_{\text{temp}} &= \hat{\delta}^{(1)}_{0,n} - \mathcal{S}_{l_1}\left\{\hat{\delta}^{(1)}_{4,n}\right\} + \mathcal{S}_{2l_1}\left\{\hat{\delta}^{(1)}_{3,n}\right\}\\
\hat{d}_{0,n} &= \mathcal{S}_{-(2w-3l_1)}\left\{\mathcal{S}_{(2w-3l_1)}\left\{d_{\text{temp}}\right\}\right\}\\
\hat{d}_{1,n} &= \hat{\delta}^{(1)}_{1,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{0,n}\right\}\\
\hat{d}_{2,n} &= \hat{\delta}^{(1)}_{2,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{1,n}\right\}\\
\hat{d}_{3,n} &= \hat{\delta}^{(1)}_{3,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{2,n}\right\}\\
\hat{d}_{4,n} &= \hat{\delta}^{(1)}_{4,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{3,n}\right\}
\end{aligned} \tag{17}$$

Explanation of (16) and (17)—see also Fig. 3: The first part of (16) creates a composite number comprising $\hat{\delta}^{(1)}_{r,n}$ in the $l_2 + k_2$ most-significant bits and $\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}$ in the $2l_2$ least-significant bits (therefore, $d^{(1)}_{\text{temp}}$ requires $3l_2 + k_2$ bits). In the second part, $\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}$ 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 (16) uses $\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}$ to recover $\hat{\delta}^{(1)}_{r,n}$ and the last three parts of (16) use: *(i)* $\hat{\delta}^{(1)}_{r,n}$ to recover $\hat{\delta}^{(1)}_{(r+1)\text{mod}5,n}$, *(ii)* $\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}$ to recover $\hat{\delta}^{(1)}_{(r+3)\text{mod}5,n}$ and, finally, *(iii)* $\hat{\delta}^{(1)}_{(r+3)\text{mod}5,n}$ to recover $\hat{\delta}^{(1)}_{(r+4)\text{mod}5,n}$. Finally, having recovered all $\hat{\delta}_{0,n},\ldots,\hat{\delta}_{4,n}$, via (17), we recover the final outputs $\hat{d}_{0,n},\ldots,\hat{d}_{4,n}$.

For the case of two fail-stop failures, $\delta^{(2)}_{r_1,n}$ and $\delta^{(2)}_{r_2,n}$ with $0 \le r_1, r_2 < M$ and $r_2 > r_1$, we first define the stream index r based on the distance $r_2 - r_1$ between the two streams. Specifically, we set:

$$r \equiv \begin{cases} r_2, & \text{if } r_2 - r_1 < 3\\ r_1, & \text{if } r_2 - r_1 \ge 3 \end{cases} \tag{18}$$

If $r_2 - r_1 = 1$, we can disentangle the tier-one outputs by $(0 \le n < N_{\text{out}})$

$$\begin{aligned}
d^{(1)}_{\text{temp}} &= \delta^{(2)}_{(r+2)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\delta^{(2)}_{(r+1)\text{mod}5,n}\right\}\\
\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n} &= \mathcal{S}_{-2(w-l_2)}\left\{\mathcal{S}_{2(w-l_2)}\left\{d^{(1)}_{\text{temp}}\right\}\right\}\\
\hat{\delta}^{(1)}_{r,n} &= \mathcal{S}_{-2l_2}\left\{-\left(d^{(1)}_{\text{temp}} - \hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}\right)\right\}\\
\hat{\delta}^{(1)}_{(r+1)\text{mod}5,n} &= \delta^{(2)}_{(r+1)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\hat{\delta}^{(1)}_{r,n}\right\}\\
\hat{\delta}^{(1)}_{(r+3)\text{mod}5,n} &= \delta^{(2)}_{(r+3)\text{mod}5,n} - \mathcal{S}_{l_2}\left\{\hat{\delta}^{(1)}_{(r+2)\text{mod}5,n}\right\}
\end{aligned} \tag{19}$$

and, on the next stage,

$$\begin{aligned}
d_{\text{temp}} &= \hat{\delta}^{(1)}_{(r+2)\text{mod}5,n} - \mathcal{S}_{l_1}\left\{\hat{\delta}^{(1)}_{(r+1)\text{mod}5,n}\right\}\\
&+ \mathcal{S}_{2l_1}\left\{\hat{\delta}^{(1)}_{(r)\text{mod}5,n}\right\}\\
\hat{d}_{(r+2)\text{mod}5,n} &= \mathcal{S}_{-(2w-3l_1)}\left\{\mathcal{S}_{(2w-3l_1)}\left\{d_{\text{temp}}\right\}\right\}\\
\hat{d}_{(r+4)\text{mod}5,n} &= \mathcal{S}_{-2l_1}\left\{d_{\text{temp}} - \hat{d}_{(r+2)\text{mod}5,n}\right\}\\
\hat{d}_{r,n} &= \hat{\delta}^{(1)}_{(r)\text{mod}5,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{(r+4)\text{mod}5,n}\right\}\\
\hat{d}_{(r+1)\text{mod}5,n} &= \hat{\delta}^{(1)}_{(r+1)\text{mod}5,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{r,n}\right\}\\
\hat{d}_{(r+3)\text{mod}5,n} &= \hat{\delta}^{(1)}_{(r+3)\text{mod}5,n} - \mathcal{S}_{l_1}\left\{\hat{d}_{(r+2)\text{mod}5,n}\right\}
\end{aligned} \tag{20}$$

If $r_2 - r_1 = 2$, we can disentangle the outputs by $(0 \le n < N_{\text{out}})$

$$
\begin{aligned}
d_{\text{temp}}^{(1)} &= \delta_{(r+2)\text{mod}5,n}^{(2)} - \mathcal{S}_{l_2}\left\{\delta_{(r+1)\text{mod}5,n}^{(2)}\right\} \\
\hat{\delta}_{(r+2)\text{mod}5,n}^{(1)} &= \mathcal{S}_{-2(w-l_2)}\left\{\mathcal{S}_{2(w-l_2)}\left\{d_{\text{temp}}^{(1)}\right\}\right\} \\
\hat{\delta}_{r,n}^{(1)} &= \mathcal{S}_{-2l_2}\left\{-\left(d_{\text{temp}}^{(1)} - \hat{\delta}_{(r+2)\text{mod}5,n}^{(1)}\right)\right\} \qquad (21)\\
\hat{\delta}_{(r+1)\text{mod}5,n}^{(1)} &= \delta_{(r+1)\text{mod}5,n}^{(2)} - \mathcal{S}_{l_2}\left\{\hat{\delta}_{r,n}^{(1)}\right\}
\end{aligned}
$$

and on the next stage

$$
\begin{aligned}
d_{\text{temp}} &= \hat{\delta}_{(r+2)\text{mod}5,n}^{(1)} - \mathcal{S}_{l_1}\left\{\hat{\delta}_{(r+1)\text{mod}5,n}^{(1)}\right\} \\
&+ \mathcal{S}_{2l_1}\left\{\hat{\delta}_{r,n}^{(1)}\right\} \\
\hat{d}_{(r+2)\text{mod}5,n} &= \mathcal{S}_{-(2w-3l_1)}\left\{\mathcal{S}_{(2w-3l_1)}\left\{d_{\text{temp}}\right\}\right\} \\
\hat{d}_{(r+4)\text{mod}5,n} &= \mathcal{S}_{-2l_1}\left\{d_{\text{temp}} - \hat{d}_{(r+2)\text{mod}5,n}\right\} \\
\hat{d}_{r,n} &= \hat{\delta}_{(r)\text{mod}5,n}^{(1)} - \mathcal{S}_{l_1}\left\{\hat{d}_{(r+4)\text{mod}5,n}\right\} \qquad (22)\\
\hat{d}_{(r+1)\text{mod}5,n} &= \hat{\delta}_{(r+1)\text{mod}5,n}^{(1)} - \mathcal{S}_{l_1}\left\{\hat{d}_{r,n}\right\} \\
\hat{d}_{(r+3)\text{mod}5,n} &= \left(\hat{\delta}_{(r+4)\text{mod}5,n}^{(2)} - \mathcal{S}_{(l_1+l_2)}\left\{\hat{d}_{(r+2)\text{mod}5,n}\right\}\right. \\
&- \left.\hat{d}_{(r+4)\text{mod}5,n}\right) \times \left(2^{l_1} + 2^{l_2}\right)^{-1}
\end{aligned}
$$

The last part of (22) can be implemented without division with factor $\left(2^{l_1} + 2^{l_2}\right)$ if one additional temporary variable, one addition and two arithmetic shifts are used.

Finally, if $r_2 - r_1 \in \{3,4\}$, disentanglement and recovery of all outputs $\hat{d}_{0,n}, \ldots, \hat{d}_{4,n}$ follows (19)–(22), albeit with $r \equiv r_1$ as per (18). We conclude the presentation of the case of $(M,K) = (5,2)$ 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 (16), the first two parts of (17), and the first three parts of (19)–(22) are presented under the assumption of a $2w$-bit integer representation since $d_{\text{temp}}^{(1)}$ and d_{temp} require $2w$ bits. However, it is straightforward to implement them via w-bit integer operations by separating $d_{\text{temp}}^{(1)}$ and d_{temp} into two parts of w bits and performing the operations separately within these parts.

Remark 2 (recovery without the use of $K \in \{1,2\}$ streams): Notice that, for the case of a single fail-stop failure, (16) does not use stream $\boldsymbol{\delta}_r^{(2)}$. This is a crucial element of our approach: since streams $\hat{\mathbf{d}}_0, \ldots, \hat{\mathbf{d}}_4$ were derived without using $\boldsymbol{\delta}_r^{(2)}$, full recovery of all outputs takes place even with the loss of one entangled stream. Similarly, for the case of two failures, (19) and (21) do not use streams $\boldsymbol{\delta}_{r_1}^{(2)}$ and $\boldsymbol{\delta}_{r_2}^{(2)}$ for the recovery of $\hat{\mathbf{d}}_0, \ldots, \hat{\mathbf{d}}_4$.

Remark 3 (detection and correction of any single SDC within $\left[\delta_{0,n}^{(2)} \ \ldots \ \delta_{4,n}^{(2)}\right]^{\mathrm{T}}$): Given that we can recover all outputs from three entangled streams, to detect and correct any single SDC within any quintuple $\left[\delta_{0,n}^{(2)} \ \ldots \ \delta_{4,n}^{(2)}\right]^{\mathrm{T}}$, we can recover five versions of all outputs $\left[\hat{d}_{0,n} \ \ldots \ \hat{d}_{4,n}\right]^{\mathrm{T}}$ using $(\forall r \in \{0,4\})$:

$$
\left\{\hat{d}_{0,n}, \ldots, \hat{d}_{4,n}\right\}^{(r)} \xleftarrow{\text{(19)-(22)}} \left\{\delta_{r,n}^{(2)}, \delta_{(r+1)\text{mod}5,n}^{(2)}, \delta_{(r+2)\text{mod}5,n}^{(2)}\right\}. \qquad (23)
$$

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 $\delta_{0,n}^{(2)}$ is erroneous due to an SDC, then the disentanglements of (23) corresponding to $r \in \{0,3,4\}$ will not agree with the ones of $r \in \{1,2\}$, which demonstrates than an SDC occurred in $\delta_{0,n}^{(2)}$ (similar for the other cases)—therefore, to mitigate the SDC occurrence, the results of the $r \in \{1,2\}$ disentanglements of (23) should be used.

Remark 4 (dynamic range): Bit $2l_1 + k_1$ within each recovered output $\hat{d}_{0,n}, \ldots, \hat{d}_{4,n}$ represents its sign bit. Given that: *(i)* each entangled output comprises the addition of two outputs (with one of them left-shifted by l_1 bits); *(ii)* the entangled outputs must not exceed $3l_1 + k_1$ bits, we conclude that the outputs of the LSB operations must not exceed the range

$$
\forall n:\ d_{0,n}, \ldots, d_{4,n} \in \left\{-\left(2^{2l_1+k_1-1} - 2^{2l_1}\right), 2^{2l_1+k_1-1} - 2^{2l_1}\right\}. \qquad (24)
$$

with l_1 and k_1 defined within the constraints of (14) and (15). Therefore, (24) comprises the range permissible for the LSB operations of (9) with the entangled representation of (12) and (13). Thus, we conclude that, for integer outputs with range bounded by (24), the extraction mechanism of (16)–(22) is *necessary and sufficient* for the recovery of *any two streams from $\boldsymbol{\delta}_0^{(2)}, \ldots, \boldsymbol{\delta}_4^{(2)}$*.

C. Generalized Entanglement in Groups of M Inputs ($M \geq 5$)

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

First, for K levels of linear superpositions of pairs of inputs, the conditions that ensure no overflow occurs are generalized to the following group of K inequalities:

$$
\begin{aligned}
(K+1)\,l_1 + k_1 &\leq (K-1)\,l_2 + k_2 \\
Kl_2 + k_2 &\leq (K-2)\,l_3 + k_3 \\
&\vdots \qquad\qquad\qquad\qquad (25)\\
3l_{K-1} + k_{K-1} &\leq l_K + k_K \\
2l_K + k_K &\leq w
\end{aligned}
$$

with the values for l_x and k_x $(\forall x:\ 1 \leq x \leq K)$ selected such that the left side of each inequality of (25) is maximum and $k_x \leq l_x$. The dynamic range supported for all outputs is $(\forall m,n)$:

$$
d_{m,n} \in \left\{-2^{Kl_1}\left(2^{k_1-1} - 1\right), 2^{Kl_1}\left(2^{k_1-1} - 1\right)\right\}. \qquad (26)
$$

We now define the following operator that generalizes the proposed numerical entanglement process:

$$\boldsymbol{\mathcal{E}}_l = \begin{bmatrix} 1 & 0 & \cdots & 0 & \mathcal{S}_l \\ \mathcal{S}_l & 1 & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & \cdots & \mathcal{S}_l & 1 & 0 \\ 0 & \cdots & 0 & \mathcal{S}_l & 1 \end{bmatrix}_{M\times M} \tag{27}$$

with $\boldsymbol{\mathcal{E}}_l$ the circulant matrix operator comprising cyclic permutations of the $1\times M$ vector $\begin{bmatrix}1 & 0 & \cdots & 0 & \mathcal{S}_l\end{bmatrix}$.

Similarly as before, per entanglement stage, two inputs are entangled together (with one of the two shifted by l bits) to create each entangled input stream of data. Any LSB operation is then performed directly on these input streams and *up to* K fail-stop failures can be mitigated within each group of M outputs. For every input stream position n, $0 \le n < N_{\text{in}}$, the entanglement vector $\begin{bmatrix}\epsilon_{0,n}^{(K)} & \cdots & \epsilon_{M-1,n}^{(K)}\end{bmatrix}^{\text{T}}$ is formed by:

$$\begin{aligned} \begin{bmatrix}\epsilon_{0,n}^{(1)} & \cdots & \epsilon_{M-1,n}^{(1)}\end{bmatrix}^{\text{T}} &= \boldsymbol{\mathcal{E}}_{l_1}\left\{\begin{bmatrix}c_{0,n} & \cdots & c_{M-1,n}\end{bmatrix}^{\text{T}}\right\} \\ \begin{bmatrix}\epsilon_{0,n}^{(2)} & \cdots & \epsilon_{M-1,n}^{(2)}\end{bmatrix}^{\text{T}} &= \boldsymbol{\mathcal{E}}_{l_2}\left\{\begin{bmatrix}\epsilon_{0,n}^{(1)} & \cdots & \epsilon_{M-1,n}^{(1)}\end{bmatrix}^{\text{T}}\right\} \\ &\vdots \\ \begin{bmatrix}\epsilon_{0,n}^{(K)} & \cdots & \epsilon_{M-1,n}^{(K)}\end{bmatrix}^{\text{T}} &= \boldsymbol{\mathcal{E}}_{l_K}\left\{\begin{bmatrix}\epsilon_{0,n}^{(K-1)} & \cdots & \epsilon_{M-1,n}^{(K-1)}\end{bmatrix}^{\text{T}}\right\} \end{aligned} \tag{28}$$

After the application of (9) on $\begin{bmatrix}\epsilon_{0,n}^{(K)} & \cdots & \epsilon_{M-1,n}^{(K)}\end{bmatrix}^{\text{T}}$, and by following a similar process as before, we can disentangle the outputs $d_{m,n}$, $0 \le m < M$, $0 \le n < N_{\text{out}}$. The remainder of this section is dedicated to proving that recovery from K fail-stop failures is possible and examining properties of the generalized entanglement.

The proof is constructed by induction. Initially, it is noted that the cases $(M,K) \in \{(3,1),(5,2)\}$ hold, since they have been demonstrated in Sections III-A and III-B.

Let us now assume that this holds for K entanglement levels derived via (28) with $M \ge 2K+1$ and l_x and k_x $(1 \le x \le K)$ selected such that the conditions of (25) hold and $\forall x : k_x \le l_x$. We shall show that, under this assumption, this also holds for $K_+ \equiv K+1$ entanglement levels and K_+ failures in $M_+ \ge 2K+3$ streams, with K_+ constraints given by (25) under the replacement of K by K_+.

We first note that it suffices to prove this result for the equality case, i.e., $M_+ = 2K+3$, as having more than $2K+3$ streams will not influence the classification of failure patterns and recovery steps discussed in the following parts of the proof. After $K+1$ failures in $2K+3$ streams, we shall have $K+2$ available streams. Out of the $\binom{2K+3}{K+2}$ possible failure patterns, it is straightforward to show that any failure pattern with three or more consecutive streams will lead to the recovery of (at least) $K+3$ streams at level K, thereby having only (up to) K missing streams at level K, 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, $\exists r \in \{0,\ldots,M_+-1\}$ such that all of the following conditions hold:

- the pair of streams $\boldsymbol{\delta}_r^{(K+1)}$ and $\boldsymbol{\delta}_{(r+1)\bmod M_+}^{(K+1)}$ are available;
- streams $\boldsymbol{\delta}_{(r-1)\bmod M_+}^{(K+1)}$ and $\boldsymbol{\delta}_{(r+2)\bmod M_+}^{(K+1)}$ are *not* available;
- 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 entanglement level K.

From the pair of available streams of level $K+1$, we can extract the following three streams of entanglement level K: $\left\{\hat{\delta}_{(r-1)\bmod M_+,n}^{(K)}, \hat{\delta}_{r,n}^{(K)}, \hat{\delta}_{(r+1)\bmod M_+,n}^{(K)}\right\}$, via $(0 \le n < N_{\text{out}})$

$$\begin{aligned} d_{\text{temp}}^{(K)} &= \delta_{(r+1)\bmod M_+,n}^{(K+1)} - \mathcal{S}_{l_{K+1}}\left\{\delta_{r,n}^{(K+1)}\right\} \\ \hat{\delta}_{(r+1)\bmod M_+,n}^{(K)} &= \mathcal{S}_{-2(w-l_{K+1})}\left\{\mathcal{S}_{2(w-l_{K+1})}\left\{d_{\text{temp}}^{(K)}\right\}\right\} \\ \hat{\delta}_{(r-1)\bmod M_+,n}^{(K)} &= \mathcal{S}_{-2l_{K+1}}\left\{-\left(d_{\text{temp}}^{(K)} - \hat{\delta}_{(r+1)\bmod M_+,n}^{(K)}\right)\right\} \\ \hat{\delta}_{r,n}^{(K)} &= \delta_{r,n}^{(K+1)} - \mathcal{S}_{l_{K+1}}\left\{\hat{\delta}_{(r-1)\bmod M_+,n}^{(K)}\right\}. \end{aligned} \tag{29}$$

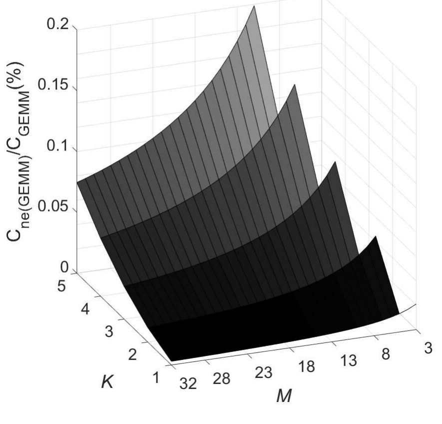
The last set of equations guarantees recovery because: *(i)* the last condition of (25) ensures that both $\hat{\delta}_{(r+1)\bmod M_+,n}^{(K)}$ and $\hat{\delta}_{(r-1)\bmod M_+,n}^{(K)}$ can be extracted from $d_{\text{temp}}^{(K)}$ and no overflow occurs; *(ii)* the penultimate condition of (25) ensures that the dynamic range at entanglement level K does not exceed the bitwidth available within the $(K+1)$-level inputs.

Given the availability of three consecutive streams at level K: $\hat{\boldsymbol{\delta}}_{(r-1)\bmod M_+}^{(K)}$, $\hat{\boldsymbol{\delta}}_r^{(K)}$ and $\hat{\boldsymbol{\delta}}_{(r+1)\bmod M_+}^{(K)}$, we can recover four consecutive streams at level $K-1$ and, by carrying the recovery process across all $K+1$ entanglement levels, we can recover $K+3$ consecutively-numbered output streams: $\hat{\mathbf{d}}_{(r-K-2)\bmod M_+},\ldots,\hat{\mathbf{d}}_{(r+1)\bmod M_+}$. Therefore, K output streams will be unavailable after this recovery process. However, these are guaranteed to be recoverable from the K available and unused streams of level $K+1$, since we have K linear equations and K unknowns in the system of $2K+3$ streams of level $K+1$. Therefore, we can mitigate $K+1$ failures in $M_+ \ge 2K+3$ streams. This means we can mitigate any $K' > K$ failures if K' entanglement levels are carried out and $M' \ge 2K'+1$ and the conditions of (25) hold with the replacement of K by K'.

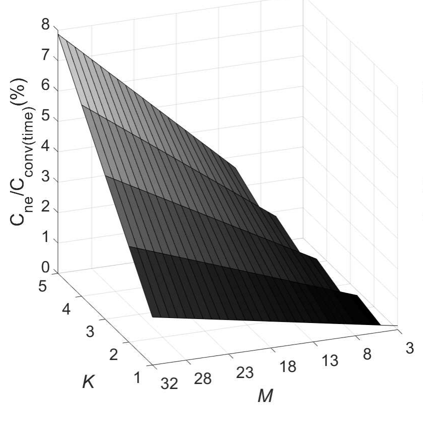
Remark 5 (detection and correction capabilities, and dynamic range of generalized entanglement): Given that we can recover all outputs from $M-K$ entangled streams, to detect and correct up to K SDCs within any M-tuple $\begin{bmatrix}\delta_{0,n}^{(K)} & \ldots & \delta_{M-1,n}^{(K)}\end{bmatrix}^{\text{T}}$, we recover M versions of all outputs $\begin{bmatrix}\hat{d}_{0,n} & \ldots & \hat{d}_{M-1,n}\end{bmatrix}^{\text{T}}$ $(\forall r \in \{0, M-1\})$:

$$\left\{\hat{d}_{0,n},\ldots,\hat{d}_{M-1,n}\right\}^{(r)} \leftarrow \left\{\delta_{r,n}^{(K)},\ldots,\delta_{(r+M-K-1)\bmod M,n}^{(K)}\right\} \tag{30}$$

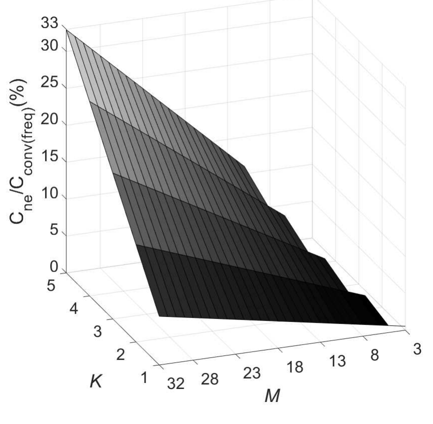
and detect any errors in the rth recovery attempt by cross-comparing the results with their remaining recovered versions, since, *at least* $M-K$ versions of $\left\{\hat{d}_{0,n},\ldots,\hat{d}_{M-1,n}\right\}^{(r)}$ of (30) will be identical. If the rth recovery attempt is deemed to be erroneous, the correct recovery from the remaining $M-K$ streams is used instead.



(a) GEMM



(b) Convolution (time)



(c) Convolution (freq)

Figure 4. Ratios of arithmetic operations for numerical entanglement, extraction, error checking and recovery versus the arithmetic operations of: generic matrix multiplication, time-domain convolution and frequency-domain convolution, with M the number of streams (or the number of subblocks within a GEMM operation) and K the number of SDCs or fail-stop failures that can be tolerated.

The dynamic range of the outputs of the generalized entanglement is expressed by (26). Examples for the maximum bitwidth achievable for different cases of M and K are given in Table II assuming a 32-bit representation. We also present the dynamic range permitted by the equivalent checksum method of (4) and (5) in order to ensure that its checksum streams do not overflow under a 32-bit representation. Evidently, the proposed approach incurs loss of 1 to 13 bits of dynamic range against the checksum method for $K > 1$, while it allows for comparable (or even superior) dynamic range for $M \geq 7$ and $K = 1$ or $M \geq 11$ and $K \in \{1, 2\}$. At the same time, our proposal does not require the overhead of applying the LSB operations to *any* additional streams, as it "overlays" the information of each input onto another input via the numerical entanglement of pairs of inputs.

Table II
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 K FAILURES IN M STREAMS (OR K SDCS IN EACH M-TUPLE $\{d_0, \ldots, d_{M-1}\}$) CAN BE MITIGATED (OR DETECTED) UNDER BOTH FRAMEWORKS, WITH THE CHECKSUM-BASED METHOD REQUIRING K ADDITIONAL STREAMS.

M	K	l_1	k_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	4	4	19	27
7	3	3	3	12	25
8	1	4	4	28	29
8	2	3	3	19	26
8	3	2	2	12	24
11	1	3	2	29	28
11	2	2	2	24	25
11	3	5	5	15	21

IV. COMPLEXITY IN LSB OPERATIONS WITH NUMERICAL ENTANGLEMENTS

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

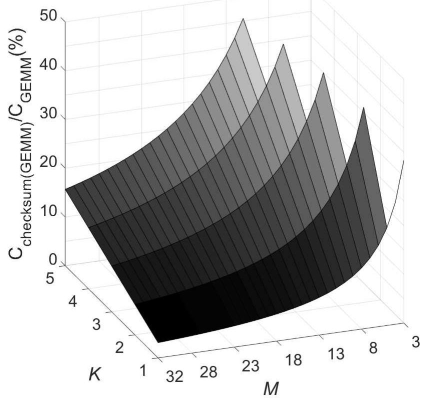


Figure 5. Ratio of arithmetic operations for checksum generation, error checking and recovery versus the arithmetic operations of: generic matrix multiplication, with M the number of streams (i.e., GEMM subblocks) and K the number of SDCs or fail-stop failures that can be tolerated.

Consider M input integer data streams, each comprising several samples and consider that an LSB operation op with kernel $\mathbf{g}$ 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 [45]. If the kernel $\mathbf{g}$ 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 [33] 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 [32], [46], [47]. 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 M subblocks of $N \times N$ integers and a matrix kernel comprising $N \times N$ integers (see [32], [37], [44], [46] for example instantiations within high-performance computing environments) is: $\mathrm{C}_{\text{GEMM}} = MN^3$. For sesquilinear operations like convolution and cross-correlation of M input integer data streams (each comprising N samples) with kernel $\mathbf{g}$ [see Fig. 1(a)], depending on the utilized realization, the number of operations can range from $O\left(MN^2\right)$ for direct algorithms (e.g., time-domain convolution) to $O\left(MN\log_2 N\right)$ for fast algorithms (e.g., FFT-based convolution) [33]. 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 [33]: $\mathrm{C}_{\text{conv,time}} = 4MN^2$ for time domain processing and $\mathrm{C}_{\text{conv,freq}} = M\left[(45N+15)\log_2(3N+1)+3N+1\right]$ for frequency-domain processing.

As described in Section III, numerical entanglement of M input integer data streams (of N samples each) requires up to $O\left(KM^2N\right)$ operations for the entanglement, extraction, error checking and recovery per output sample [including the M-fold recovery and error check of (30) of Remark 5]. For example, ignoring all arithmetic-shifting operations (which take a negligible amount of time), based on the description of Section III the upper bound of the operations for numerical entanglement, extraction, error checking and recovery is: $C_{\text{ne,conv}} = 2KM^2N$. Similarly as before, for the special case of the GEMM operation using M subblocks of $N \times N$ integers, the upper bound of the overhead of numerical entanglement of all inputs is: $C_{\text{ne,GEMM}} = 2\left(KMN\right)^2$. We present the percentile values obtained for $\frac{\mathrm{C}_{\text{ne,GEMM}}}{\mathrm{C}_{\text{GEMM}}} \times 100\%$, $\frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,time}}} \times 100\%$ and $\frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,freq}}} \times 100\%$ in Fig. 4 for typical values of (M,K) and $N = 500$, which represents a typical subblock size in high-performance GEMM and convolution/cross-correlation operations. For sesquilinear operations like matrix products the overhead of numerical entanglement, extraction, error checking and recovery in terms of arithmetic operations is below 0.2%. For convolution and cross-correlation this overhead varies from below 8% to 32% depending on implementation. Moreover,

$$\lim_{N\to\infty}\frac{\mathrm{C}_{\text{ne,GEMM}}}{\mathrm{C}_{\text{GEMM}}} = \lim_{N\to\infty}\frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,time}}} = \lim_{N\to\infty}\frac{\mathrm{C}_{\text{ne,conv}}}{\mathrm{C}_{\text{conv,freq}}} = 0, \tag{31}$$

i.e., the overhead of the proposed approach approaches 0% as the dimension of the LSB processing increases.

For comparison purposes, Fig. 5 shows the percentile overhead of checksum methods [Fig. 1(b) [37], [43], [44]] under the same range of values for (M,K) and $N = 500$, i.e., for the same fail-stop failure or SDC mitigation capability[5]. 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 $\mathrm{C}_{\text{checksum,GEMM}} = 2K^2MN^2 + \frac{K}{M}\mathrm{C}_{\text{GEMM}}$, $\mathrm{C}_{\text{checksum,conv,time}} = 2KMN + \frac{K}{M}\mathrm{C}_{\text{conv,time}}$ and $\mathrm{C}_{\text{checksum,conv,freq}} = 2KMN + \frac{K}{M}\mathrm{C}_{\text{conv,freq}}$ represent the overhead in terms of operations count (additions/multiplications) for each case. Given that checksum methods for time-domain/frequency-domain convolution and GEMM exhibit the same percentile overhead (with variation between them limited to no more than 0.2%), Fig. 5 illustrates only the latter. As expected, the overhead of checksum methods converges to $\frac{K}{M} \times 100\%$ as the dimension of the LSB processing operations increases, i.e.,

$$\begin{aligned}\lim_{N\to\infty}\frac{\mathrm{C}_{\text{checksum,GEMM}}}{\mathrm{C}_{\text{GEMM}}} &= \lim_{N\to\infty}\frac{\mathrm{C}_{\text{checksum,conv,time}}}{\mathrm{C}_{\text{conv,time}}} && (32)\\ &= \lim_{N\to\infty}\frac{\mathrm{C}_{\text{checksum,conv,freq}}}{\mathrm{C}_{\text{conv,freq}}} && (33)\\ &= \frac{K}{M}.\end{aligned}$$

Therefore, checksum methods lead to substantial overhead (which can be surpass 45%) when high reliability is pursued, i.e., when $M \leq 8$ and $K > 1$. Even for the low reliability regime (i.e., when $M > 8$ and $K = 1$), Fig. 5 shows that checksum methods can incur more than 4% overhead in terms of arithmetic operations. On the other hand, with the exception of frequency-domain convolution with $K > 1$, the proposed method always incurs less than 8% overhead. This overhead reduces even further with increasing values for N. Overall, the comparison between Fig. 4 and Fig. 5 demonstrates that, for the high reliability regime (i.e., $M \leq 15$), the complexity overhead of the proposed approach is expected to be one to two orders of magnitude smaller than that of checksum methods.

V. 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. 6 and Fig. 7, we present throughput results for the mitigation of up to $K = 3$ core failures when running on $M = 7$ and $M = 11$ 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

[5]Unlike the row-column algorithm-based fault-tolerance method of Huang and Abraham [24], the checksum method for fail-stop failure mitigation in GEMM generates an additional (i.e., checksum) subblock, since the former cannot mitigate fail-stop core failures.

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[6]. Importantly, the failure intolerant computation, together with the proposed algorithm utilizes all M cores for data computation. On the other hand, the checksum method can only use $M_{\text{data}} = M - K$ cores for actual input data processing, while reserving K cores for checksum data computation. In our experiments, to create the redundant streams of (4) with $M = M_{\text{data}}$, we utilized the weighted checksum-based method of Luk *et. al.* [39] [38] with $K = 3$ and linear weights given by the $1 \times M_{\text{data}}$ vectors: $\mathbf{w}_0 = [1\,1 \cdots 1]$, $\mathbf{w}_1 = [1\,2 \cdots M_{\text{data}}]$ and $\mathbf{w}_2 = \left[2^0\,2^1 \cdots 2^{M_{\text{data}}-1}\right]$.

The results of Fig. 6 and Fig. 7 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 [47] for $M = \{7, 11\}$ and $K \in [1, 3]$. 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 (M, N) values. On the other hand, the throughput achieved by the checksum-based [39] [38] 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.

VI. 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 $(M, K) = \{(3,1), (5,1), (7,1)\}$, 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 M vector-matrix multiplications of dimensions 1×2000-by-2000×2000. Artificial fault injection is performed during the above multiplication via KULFI [48], an open source LLVM-based [49] fault injection tool. Transient faults are injected during one of the M multiplications, with all M streams ($M + 1$ 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 III 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 [38] [39]. In all cases, all faults were detected correctly by both the proposed and the checksum-based method.

VII. Conclusions

We propose a new approach for LSB processing of integer data streams that is based on the novel concept of numerical entanglement. Under M input streams ($M \geq 3$), the proposed approach provides for: *(i)* guaranteed mitigation of multiple (up to $K = \left\lfloor \frac{M-1}{2} \right\rfloor$) fail-stop failures or SDCs; *(ii)* complexity overhead that depends only on (M, K) and not on the complexity of the performed LSB operations, thus, quickly becoming negligible as the complexity of the LSB 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 I. As such, it is envisaged that it will find usage in a multitude of systems that require enhanced reliability against core failures in hardware with very low implementation overhead.

References

[1] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *Proc. 38th IEEE Int. Symp. Computer Archit. (ISCA), 2011*. IEEE, 2011, pp. 461–471.

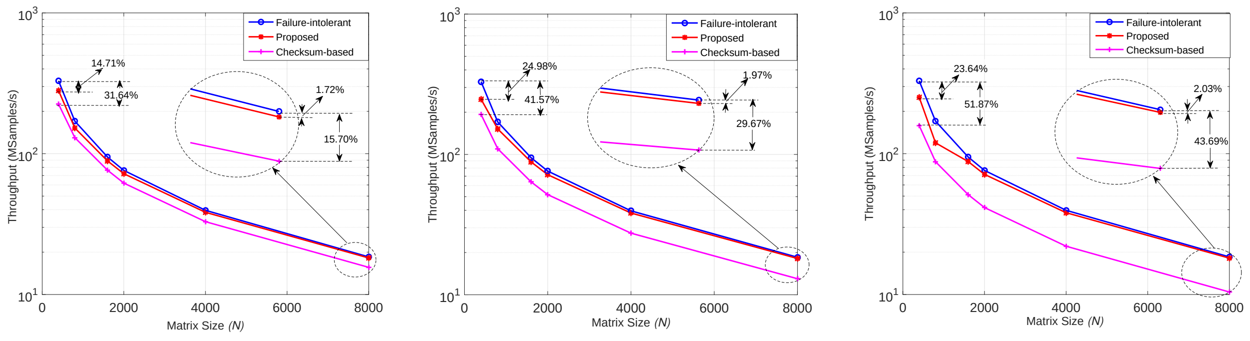
[2] Y. Andreopoulos, "Error tolerant multimedia stream processing: There's plenty of room at the top (of the system stack)," *IEEE Trans. on Multimedia*, vol. 15, no. 2, pp. 291–303, 2013.

[3] S. Gotoda et al., "Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 260–267.

[4] D. Fiala et al., "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 78.

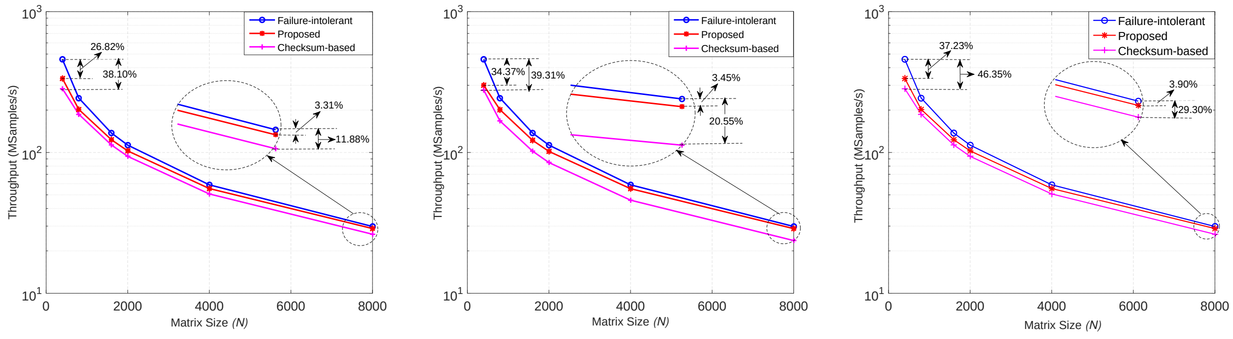
[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 2008, pp. 1–10.

[6]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) $(M,K)=(7,1)$ (b) $(M,K)=(7,2)$ (c) $(M,K)=(7,3)$

Figure 6. Throughput results for GEMM for the mitigation of core failures in $M=7$ streams. Failure-intolerant computation using Intel MKL 11.0 is used as a benchmark.



(a) $(M,K)=(11,1)$ (b) $(M,K)=(11,2)$ (c) $(M,K)=(11,3)$

Figure 7. Throughput results for GEMM for the mitigation of core failures in $M=11$ streams. Failure-intolerant computation using Intel MKL 11.0 is used as a benchmark.

Table III
AVERAGE EXECUTION (IN MICROSECONDS) AND PERCENTILE COMPARISONS AGAINST THE FAULT-INTOLERANT (CONVENTIONAL) INTEL MKL SGEMM FOR A SINGLE ERROR DETECTION WITHIN M MULTIPLICATIONS OF SIZE 1×2000-BY-2000×2000.

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

[6] W. Kurschl and W. Beer, "Combining cloud computing and wireless sensor networks," in *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2009, pp. 512–518.

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

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

[9] 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. Patt. Anal. and Machine Intel.*, vol. 26, no. 1, pp. 131–137, 2004.

[10] J.-S. Coron, T. Lepoint, and M. Tibouchi, "Batch fully homomorphic encryption over the integers," Tech. Rep., Cryptology ePrint Archive, Rep. 2013/036, 2013. http://eprint. iacr. org, 2013.

[11] P. M. Fenwick, "The Burrows–Wheeler transform for block sorting text compression: principles and improvements," *The Comp. J.*, vol. 39, no. 9, pp. 731–740, 1996.

[12] Y. Peng, B. Gong, H. Liu, and Y. Zhang, "Parallel computing for option pricing based on the backward stochastic differential equation," in *Springer High Perform. Comput. and Applic.*, pp. 325–330. 2010.

[13] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*, O'Reilly Media, Incorporated, 2008.

[14] 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.

[15] C. Wang et al., "A job pause service under lam/mpi+ blcr for transparent fault tolerance," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–10.

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

[17] M. Treaster, "A survey of fault-tolerance and fault-recovery techniques in parallel systems," pp. 1–11, 2005.

[18] I. Philp, "Software failures and the road to a petaflop machine," in *HPCRI: 1st Workshop on High Performance Computing Reliability Issues, in Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)*, 2005.

[19] J. T. Daly et al., "Application MTTFE vs. platform MTBF: A fresh perspective on system reliability and application throughput for computations at scale," in *Proc. 8th IEEE International Symposium*

on Cluster Computing and the Grid, CCGRID'08. IEEE, 2008, pp. 795–800.

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

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

[22] J.-Y. Jou and J.A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," in *Proc. of the IEEE*. IEEE, May 1986, vol. 74, pp. 732,741.

[23] Z. Chen, "Optimal real number codes for fault tolerant matrix operations," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 29.

[24] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. 100, no. 6, pp. 518–528, 1984.

[25] Z. Gao, P. Reviriego, Z. Xu, X. Su, J. Wang, and J. A. Maestro, "Efficient coding schemes for fault-tolerant parallel filters," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 7, pp. 666–670, July 2015.

[26] 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.

[27] P. Reviriego, S. Pontarelli, C. J. Bleakley, and J. A. Maestro, "Area efficient concurrent error detection and correction for parallel filters," *Electronics Letters*, vol. 48, no. 20, pp. 1258–1260, September 2012.

[28] S. Pontarelli and A. Salsano, "On the use of karatsuba formula to detect errors in gf((2(sup)n(/sup))(sup)2(/sup)) multipliers," *IET Circuits, Devices Systems*, vol. 6, no. 3, pp. 152–158, May 2012.

[29] S. Pontarelli, P. Reviriego, C. J. Bleakley, and J. A. Maestro, "Low complexity concurrent error detection for complex multiplication," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1899–1903, Sept 2013.

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

[31] Z. Chen, "Algorithm-based recovery for iterative methods without checkpointing," in *Proc. 20th Int. Symposium on High performance Distributed Computing*. ACM, 2011, pp. 73–84.

[32] K. Goto and R. A Van De Geijn, "Anatomy of high-performance matrix multiplication," *ACM Trans. Math. Soft*, vol. 34, no. 3, pp. 12, 2008.

[33] M. A. Anam and Y. Andreopoulos, "Throughput scaling of convolution for error-tolerant multimedia applications," *IEEE Trans. Multimedia*, vol. 14, no. 3, pp. 797–804, 2012.

[34] D. Anastasia and Y. Andreopoulos, "Software designs of image processing tasks with incremental refinement of computation," *IEEE Trans. Image Process.*, vol. 19, no. 8, pp. 2099–2114, 2010.

[35] C. Lin, B. Zhang, and Y. F. Zheng, "Packed integer wavelet transform constructed by lifting scheme," *IEEE Trans. Circ. and Syst. for Video Technol.*, vol. 10, no. 8, pp. 1496–1501, 2000.

[36] M. A. Anam and Y. Andreopoulos, "Failure mitigation in linear, sesquilinear and bijective operations on integer data streams via numerical entanglement," in *Proc. IEEE Int. On-Line Testing. Symp., IOLTS*, submitted, 2015.

[37] 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. 10th ACM SIGPLAN Symp. Princip. and Pract. Paral. Prog.*, 2005, pp. 213–223.

[38] F. T. Luk, "Algorithm-based fault tolerance for parallel matrix equation solvers," *SPIE, Real-Time Signal processing VIII*, vol. 564, pp. 631–635, 1985.

[39] V. K. Stefanidis and K. G. Margaritis, "Algorithm based fault tolerance: Review and experimental study," in *International Conference of Numerical Analysis and Applied Mathematics*. IEEE, 2004.

[40] V.S.S Nair and J.A. Abraham, "General linear codes for fault tolerant matrix operations on processor arrays," in *Int. Symp. Fault Tolerant Comput.* IEEE, 1988, pp. 180–185.

[41] J. Sloan, R. Kumar, and G. Bronevetsky, "Algorithmic approaches to low overhead fault detection for sparse linear algebra," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. IEEE, 2012, pp. 1–12.

[42] N.K. Rexford, J.; Jha, "Algorithm-based fault tolerance for floating-point operations in massively parallel systems," in *Proceedings., 1992 IEEE International Symposium on Circuits and Systems*. IEEE, May 1992, vol. 2, pp. 649,652.

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

[44] D. G Murray and S. Hand, "Spread-spectrum computation," in *Proc. USENIX 4th Conf. Hot Top. in Syst. Dependab.*, 2008, pp. 5–9.

[45] G. H Golub and C. F Van Loan, *Matrix computations*, Johns Hopkins University Press, 1996.

[46] D. Anastasia and Y. Andreopoulos, "Throughput-distortion computation of generic matrix multiplication: Toward a computation channel for digital signal processing systems," *IEEE Trans. Signal Process.*, vol. 60, no. 4, pp. 2024–2037, 2012.

[47] MKL Intel, "Intel math kernel library," 2007.

[48] V. C. Sharma et al., "Towards formal approaches to system resilience," in *Proc. IEEE 19th Pacific Rim Int. Symp. on Depend. Comp. (PRDC)*. IEEE, 2013, pp. 41–50.

[49] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Int. Symp. on Code Gen. and Optim., 2004, (CGO)*. IEEE, 2004, pp. 75–86.

Mohammad Ashraful Anam obtained the PhD in Electronic Engineering from University College London (Lombardi Prize for the Best PhD thesis in Electronic Engineering) and is currently post-doctoral research associate in the Department of Electronic and Electrical Engineering, University College London, London, UK. His research interests are in error tolerant computing, and reliable cloud computing.

Ijeoma Anarado is currently pursuing the Ph.D. degree at the Department of Electronic and Electrical Engineering, University College London, U.K. Her research interests include the design of system level algorithms for fault tolerance in data computations and throughput acceleration in signal processing tasks. Her PhD is funded by the Federal Government of Nigeria under the PRESSID Scheme.

Yiannis Andreopoulos (M'00-SM'14) obtained the Electrical Engineering Diploma and an MSc in Signal and Image Processing Systems from the University of Patras, Greece, and the PhD in Applied Sciences from the Vrije Universiteit Brussel, Belgium. He is Reader (Assoc. Professor) in Data and Signal Processing Systems in the Department of Electronic and Electrical Engineering of University College London (U.K.). His research interests are in wireless sensor networks, error-tolerant computing and multimedia systems. He received the 2007 Most-Cited Paper Award from the Elsevier EURASIP Signal Processing: Image Communication journal and a best paper award from the 2009 IEEE Workshop on Signal Processing Systems. He was Special Sessions Co-Chair of the 10th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2009) and Programme Co-Chair of the 18th International Conference on Multimedia Modeling (MMM 2012) and the 9th International Conference on Body Area Networks (BODYNETS 2014). He has been an Associate Editor of the IEEE Transactions on Multimedia, the IEEE Signal Processing Letters and Image and Vision Computing (Elsevier).