AVF-driven Parity Optimization for MBU Protection of In-core Memory Arrays

Michail Maniatakos
Electrical and Computer Engineering
New York University Abu Dhabi
michail.maniatakos@nyu.edu

Maria K. Michael
Electrical and Computer Engineering
University of Cyprus
mmichael@ucy.ac.cy

Yiorgos Makris
Electrical Engineering
University of Texas at Dallas
yiorgos.makris@utdallas.edu

***Abstract*—We propose an AVF-driven parity selection method for protecting modern microprocessor in-core memory arrays against MBUs. As MBUs constitute more than 50% of the upsets in latest technologies, error correcting codes or physical interleaving are typically employed to effectively protect out-of-core memory structures, such as caches. However, such methods are not applicable to high-performance in-core arrays, due to computational complexity, high delay and area overhead. To this end, we revisit parity as an effective mechanism to detect errors and we resort to pipeline flushing and checkpointing for correction. We demonstrate that optimal parity tree construction for MBU detection is a computationally complex problem, which we then formulate as an integer-linear-program (ILP). Experimental results on Alpha 21264 and Intel P6 in-core memory arrays demonstrate that optimal parity tree selection can achieve great vulnerability reduction, even when a small number of bits are added to the parity trees, compared to simple heuristics. Furthermore, the ILP formulation allows us to find better solutions by effectively exploring the solution space in the presence of multiple parity trees; results show that the presence of 2 parity trees offers a vulnerability reduction of more than 50% over a single parity tree.**

I. Introduction

Single event upsets (SEUs), attributed to alpha particle or neutron strikes [1], have been extensively studied over the last decade and various countermeasures have been developed to address the resulting transient errors affecting modern microprocessors [2]. Recent radiation-induced soft error rate (SER) scaling trends show that, while the single-bit soft error rate for SRAMs continues to decrease and the error rate for sequential and static combinational devices has not changed, the multi-bit SER has increased dramatically [3].

In the presence of a multi-bit upset (MBU), two or more physically adjacent SRAM bits are upset by a single neutron particle [4]. During an MBU, multiple bit errors in a single word can be introduced, as well as single bit errors in multiple adjacent words [5]. As contemporary memory structures exhibit an increasing multi-bit failure rate, the importance of MBU analysis has been highlighted in several recent publications [6], [7]. Recent work showed that MBUs can affect up to 8 adjacent cells [8].

Considering both single-bit and multi-bit upsets becomes particularly important when assessing vulnerability of modern microprocessors, as they typically include numerous in-core memory arrays in order to support high-performance execution. Besides the use of SRAMs for large memory structures, such as instruction and data caches, low power budget dictates the use of SRAM-based structures for various in-core memory arrays, such as the instruction queue or the register allocation table [9]. Modern microprocessor designs incorporate Content Addressable Memory (CAM)/RAM-based structures [10] instead of latch-based memories, in order to achieve power savings of 36% on average [11]. These structures are built using SRAM technology, with a CAM cell consisting of two SRAM cells [10]. As SRAMs come at the cost of increased susceptibility to single and multiple bit errors, counter measures against radiation induced errors need to be put in place.

Typical methodologies for MBU protection include physical interleaving [12], Error Correcting Codes (ECC) [13], and parity. Interleaving refers to the creation of logical checkwords from physically dispersed locations of the memory array, forcing the MBUs to appear as multiple single-bit errors, instead of a multi-bit error. Checkwords are generated based on a specified ECC scheme, thus interleaved memories rely on advanced error correcting codes. However, while applying ECC protection to out-of-core memories (such as the caches) is the state-of-the-art method for resiliency enhancement, generation of checkwords at core clock speed is infeasible, due to computational complexity. Thus, efficient ECC methods cannot be applied to the fast, in-core memory arrays.

Parity-based methods, however, which are far less complex, may still constitute a feasible solution. While parity offers only detection capabilities, it is sufficient for in-core memory arrays of modern microprocessors as other correction mechanisms, such as pipeline flushing and checkpoint restoring, can be applied after the fault has been detected. Yet, blindly applying parity across the board not only incurs significant area, power and delay overhead but may also reduce the achieved coverage. Furthermore, parity protection of certain bits is unnecessary, as they may, ultimately, have a low probability of affecting the application outcome. Instead, similar to the low-overhead parity selection optimization methods that were introduced in different contexts in [14] and [15], judicious parity construction is necessary to minimize vulnerability. As we discuss in Section II, optimal parity selection for MBU detection is not straightforward and simple heuristics yield sub-optimal results. Thus, in Section III we formulate the problem as an integer linear program (ILP). Results are presented in Section IV, followed by conclusions in Section V.

II. Selective Parity

While parity is a potentially viable option for protecting in-core memory arrays, adding all bits to a single parity tree is not a good idea for the following two reasons:

978-3-9815370-0-0/DATE13/©2013 EDAA

- In-core memory arrays in modern microprocessors are typically quite wide, in order to store all the information needed for out-of-order instruction execution. For example, the information appended to an instruction word in the Alpha 21264 ranges between 160 and 290 bits. Since up to 32 instructions can be in-flight, the microprocessor employs several large in-core memory arrays to support the pipelined execution engine. Hence, adding parity trees for all the bits in each word of these memory arrays would incur significant overhead in terms of area, power consumption, and delay.
- More importantly, such a parity tree would only detect MBUs causing an odd number of errors. Therefore, the parity scheme would fail to detect 2-bit MBUs, 4-bit MBUs etc., which constitute a significant portion of current MBU distributions.

Evidently, connecting only a carefully selected subset of bits to the parity tree might yield better overall protection from MBUs. Moreover, not all bits in such words are equally critical. Indeed, since the type of information stored by each bit is known in advance, we can characterize *a priori* its relative importance and vulnerability. For this purpose, we can use the Architectural Vulnerability Factor (AVF) of each bit, which was first introduced in [16] and which captures the probability that a bit-flip will cause a system-visible error. Consider, for instance, the word of a sample 8-bit memory array, shown in Fig. 1, and let us assume that bit i_0 has an AVF of 0.5. In this case, only half of the faults in this particular bit will affect the end-user. Similarly, let us assume that bit i_2 has an AVF of 0, therefore no faults affecting it can produce a visible error. An example of such a case could be a memory array storing information related to branch prediction, which is used to populate the branch history table. The impact of a fault affecting bit i_2 would only result in a different prediction, but not to a system-visible error.

By taking into account the vulnerability of each bit, we can select the most appropriate subset of bits to add to the parity tree, effectively introducing an AVF-driven parity optimization method. In our example, since bits i_2 and i_3 have an AVF of 0, including them in the parity tree is unnecessary. In other words, a parity tree including the remaining 6 bits would be equally effective as a parity tree including all 8 bits, yet it would incur less area, power, and delay overhead. Also, note that bit i_6, which has a very low AVF of 0.2, is adjacent to bit i_7, which has a very high AVF of 0.9. This implies that MBUs which affect both of these bits will be masked. Leaving i_6 out of the tree will enable detection of such MBUs, which have a high probability of becoming visible to the system, at the cost of allowing single errors on i_6 to propagate (with low probability) to the system level. In other words, careful AVF-driven selection of bits to include in the parity tree can be beneficial both in terms of overhead and in terms of coverage.

Nevertheless, any such single parity tree will continue to be ineffective in detecting MBUs that affect an even number of bits among those connected to the tree. To alleviate the problem, a possible solution is the addition of *multiple* parity trees. In our previous example, if a 2-bit wide upset affects bits i_0 and i_1, it will propagate undetected; however, if i_0 and i_1 are connected to different parity trees, an error affecting both will be detected separately by the two parity bits. Adding more parity trees comes at the cost of an extra parity bit which needs to be stored per word. However, assuming that the total number of bits connected to the parity trees is the same, it does not require additional XOR gates. In fact, it speeds up parity computation since the depth of each tree is smaller than that of a single tree. We also note that the maximum number of trees that one should consider does not exceed the maximum width of the expected MBUs. Indeed if, for example, a single event upset affects at most 2 adjacent bits of a memory word, addition of a third parity tree is superfluous since all the pairs of potentially erroneous bits can be split into the two trees. The same holds true when there are no adjacent bits with high AVF, essentially implying that a very small number of parity trees will suffice.

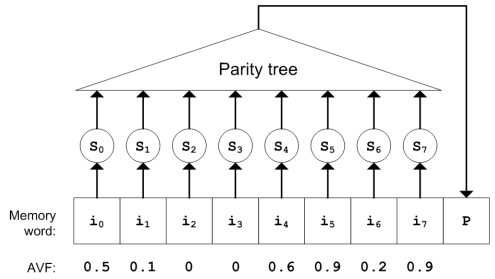


Fig. 1. Example of parity selection for protecting memory words

To summarize the problem at hand, given (i) the expected MBU distribution, (ii) the vulnerability of the individual bits, (iii) the maximum number b of bits that the budget allows to connect to parity trees, and (iv) the number t of available parity trees, we seek to choose which bits to add to each of the parity trees in order to maximize the protection of the memory word from MBUs.

A. Simple Algorithm

A straightforward algorithm for selecting which bits to add to each of the parity trees is to select and place the b most vulnerable bits to the t parity trees in a round-robin fashion. In the example shown in Fig. 1, for $b = 5$ and for $t = 2$, bits $\{i_0, i_4, i_5, i_6, i_7\}$ will be selected, with $\{i_0, i_5, i_7\}$ connected to the first parity tree and $\{i_4, i_6\}$ to the second.

This simple algorithm, however, may yield sub-optimal results, especially since it does not take into account the distribution of MBU faults. For instance, a 4-bit wide upset affecting bits $\{i_4, i_5, i_6, i_7\}$ will go undetected as both parity trees will experience an even number of errors. Given the high AVFs of the corresponding bits, this MBU will most likely affect the user. However, if i_6 was omitted from the trees, this particular MBU would be detected, although the word would now be vulnerable to single bit upsets affecting bit i_6. As the latter has a very low AVF (0.2), its exclusion can give better results than the previous configuration. Evidently, a wider range of solutions needs to be explored in order to obtain the optimal subset.

Given a word of k bits and a budget of b bits to connect to parity trees, the size of the solution space for a single parity tree is $\binom{b}{k}$. In a commercial design such as the Alpha 21264 which has a k=219-bit instruction queue memory array, this implies that with a budget of b=44 bits (20%), the number of possible solutions is $\binom{219}{44} \approx 2.21e+46$. This space increases dramatically as more trees are added. This huge solution space, in combination with the inability of simple heuristics to provide an optimal solution to this Cover-like problem (as further demonstrated in Section IV), pinpoint the need for a more general solution. To this end, in the next section we formulate parity selection as an Integer Linear Program and we use dedicated ILP solvers for obtaining the optimal solution.

III. Formulation of parity optimization ILP

A. ILP formulation

The goal of the parity optimization problem is to minimize the vulnerability of the in-core memory array. Since we add parity per memory word, the developed cost function will refer to each individual word. Thus, in order to formulate the cost function to be minimized, called *Memory Word Vulnerability Factor (MWVF)*, we define the following ILP:
Given the parameters:

- k: Number of bits in the memory word
- V_i: Architectural vulnerability factor of bit i, $V_i \in [0,1]$
- d: Maximum MBU distance, defined by fault model
- P_j: Probability of a j-wide MBU, defined by fault model, $P_j \in [0,1]$, $j \in \{1,2,...,d\}$, $\sum_{j=1}^{d} P_j = 1$
- t: The number of parity trees, $t \geq 1$
- b: Maximum number of bits to be added to the parity trees, $1 \geq b \geq k$

Solve for:

- $S_{i,r} \in \{0,1\}$
- $y_{i,j,m,r} \in \{0,1\}$
- $x_{i,j,m,r} \in \{0,1,...,\lfloor j/2 \rfloor\}$
- $z_{i,j,m} \in \{0,1,...,t-1\}$
- $w_{i,j,m} \in \{0,1\}$

in the domain $i \in \{1,2,...,k\}$, $j \in \{1,2,...,d\}$, $m \in \{1,2,...,j\}$, $r \in \{1,2,...,t\}$

Minimize cost function:

$$\sum_{i=1}^{k} V_i \sum_{j=1}^{d} \frac{P_j}{j} \sum_{m=1}^{j} w_{i,j,m} \tag{1}$$

Subject to constraints:

- $\sum_{i=1}^{k} \sum_{r=1}^{t} S_{i,j} \leq b$
- $\sum_{n=1}^{j} S_{i-j+m+n-1,r} = 2x_{i,j,m,r} + y_{i,j,m,r}$
- $\sum_{r=1}^{t} (1 - y_{i,j,m,r}) = t * w_{i,j,m} + z_{i,j,m}$

B. Formulating cost function

Let us now explain how this cost function was derived and why it reflects the choice and distribution of bits to parity trees which minimizes vulnerability of an in-core memory word to MBUs. This vulnerability, which we termed MWVF, is defined as the sum of the individual MBU vulnerabilities of all bits in the word. The vulnerability of each individual bit is defined as the product of the bit AVF (V_i) multiplied by the probability that a j-wide MBU will affect bit i. Thus, the initial formulation of MWVF is the following:

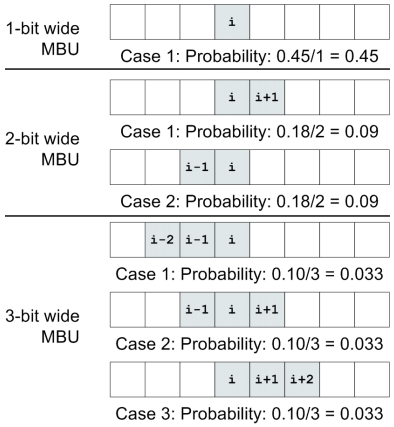


Fig. 2. Bitwise probability distribution in j-wide MBUs

$$\sum_{i=1}^{k} V_i * (\text{probability of a } j\text{-wide MBU affecting bit } i) \tag{2}$$

Given an MBU distribution, bit i might be part of a 1-bit wide MBU (or single bit upset - SBU), a 2-bit wide MBU, a 3-bit wide MBU etc. For example, given that bit i has two neighboring bits, a 2-bit wide MBU might affect the pair $\{i-1, i\}$ or the pair $\{i, i+1\}$. For the presented formulation, we assume that MBUs affect bits horizontally; vertical MBUs are dealt with by observing that each memory word is protected by different parity bits.

The MBU fault model defines how the MBUs manifest to the memory array. For instance, an MBU distribution, taken from [8] for 65nm memories, is [1: 0.45, 2: 0.18, 3: 0.10, 4: 0.27]. This distribution indicates that with 0.45 probability, the MBU will affect one bit, with 0.18 probability it will affect two bits, etc. In case of a 2-bit wide MBU, a fault that includes bit i should be analyzed seperately in case $i-1$ and i bits are affected, and in case i and $i+1$ are affected. This distiction is essential, because the vulnerability factor changes according to whether $i-1$ or $i+1$ is included in a parity tree.

In this study, we make the assumption that the two cases have equal probability. Given the MBU distribution described earlier, we assume that the probability of a 2-bit wide MBU affecting bits $i-1$ and i is 0.09 (0.18/2), and the probability of an MBU affecting bits i, $i+1$ is also 0.09. Therefore, in case of a j-wide MBU, the probabilities are distributed equally among all cases, as shown in Fig. 2. Equation 2 now becomes:

$$\sum_{i=1}^{k} V_i \sum_{j=1}^{d} \frac{P_j}{j} \sum_{m=1}^{j} (j\text{-wide MBU affects bit } i) \tag{3}$$

In the simple case of one parity tree ($r = 1$), a j-wide MBU will affect bit i if an even number of these j bits are protected by parity. For example, in case bits $i-1$, i, and

$i+1$ are affected by an MBU, the error will be detected if $S_{i-1} + S_i + S_{i+1}$ is an odd number, implying that 1 or 3 of these bits are protected by parity. In case 0 or 2 bits are protected, the error will be masked, and will affect the user with probability V_i * $P_3/3$.

Therefore, in case the sum of S of the j affected bits is even, the corresponding case should be set to 1, otherwise it should be set to 0. Equation 3 now becomes:

$$\sum_{i=1}^{k} V_i \sum_{j=1}^{d} \frac{P_j}{j} \sum_{m=1}^{j} (1 - ((\sum_{n=1}^{j} S_{i+m+n-j-1}) \mod 2)) \quad (4)$$

Note that the inclusion of the *mod* operator converts the problem to non-linear. In the next section we present the transformations to linearize it.

Equation 4 assumes that only one parity tree is used. In order to account for t multiple trees, Equation 4 is extended to the following equation:

$$\sum_{i=1}^{k} V_i \sum_{j=1}^{d} \frac{P_j}{j} \sum_{m=1}^{j} \prod_{r=1}^{t} (1 - ((\sum_{n=1}^{j} S_{i+m+n-j-1,r}) \mod 2)) \quad (5)$$

The expression $(1 - ((\sum_{n=1}^{j} S_{i+m+n-j-1,r}) \mod 2)$ is 0 when, in case of a j-wide MBU, there is at least one case (out of the j cases) that the error will be detected by parity. Therefore, the addition of the product operator ensures that the contribution of the particular case to the total vulnerability will be 0 if there is at least one parity tree that detects the corresponding fault. For instance, if only parity tree 2 out of 3 trees detects the tested case, the product will be $1*0*1 = 0$, implying that the error is detected by the current configuration of $S_{i,r}$ and will not contribute to the total MWVF.

Equation 5 is the cost function that we want to minimize. Since we indicate that b out of the k bits will be added to the parity trees, the following constraint is added:

$$\sum_{i=1}^{k} \sum_{j=1}^{t} S_{i,j} \leq b \quad (6)$$

Note that this constraint does not preclude the inclusion of a bit to multiple trees.

C. Converting cost function to linear

The inclusion of the *mod* and the *product* operators make this optimization problem non-linear. In this section, we introduce two transformations to convert it back to linear.

In order to remove the *mod* operator, the expression $((\sum_{n=1}^{j} S_{i+m+n-j-1,r}) \mod 2)$ is re-written as $(y_{i,m,n,j,r})$, and the following constraints are added:

$$\sum_{n=1}^{j} S_{i-j+m+n-1,r} = 2x_{i,j,m,r} + y_{i,j,m,r} \quad (7)$$

$$y_{i,j,m,r} \in \{0,1\}, x_{i,j,m,r} \in \{0,1,...,\lfloor j/2 \rfloor\}$$

The variables $x_{i,j,m,r}$, $y_{i,j,m,r}$ are added to the solver. Expression 7 implies that $y_{i,j,m,r}$ will be 0 when $\sum_{n=1}^{j} S_{i-j+m+n-1,r}$ is an even number, otherwise it will be 1. This effectively replaces the mod operator, and our cost function now becomes:

$$\sum_{i=1}^{k} V_i \sum_{j=1}^{d} \frac{P_j}{j} \sum_{m=1}^{j} \prod_{r=1}^{t} (1 - y_{i,m,n,j,r}) \quad (8)$$

The final step of converting the cost function to linear is the removal of the product operator. Similarly to the previous operation, we replace $\prod_{r=1}^{t}(1-y_{i,m,n,j,r})$ with $w_{i,j,m}$, adding the following constraints:

$$\sum_{r=1}^{t} (1 - y_{i,j,m,r}) = t * w_{i,j,m} + z_{i,j,m} \quad (9)$$

$$z_{i,j,m} \in \{0,1,...,t-1\}, w_{i,j,m} \in \{0,1\}$$

Since the term $z_{i,j,m}$ is a positive number smaller than t, $w_{i,j,m}$ will be 1 *iff* $\sum_{r=1}^{t}(1-y_{i,j,m,r}) = t$. The latter implies that all y terms are 0, thus there is no parity tree detecting the corresponding fault. If at least one tree detects the fault, its y term will be 1 and w will be forced to 0.

Therefore, the final optimization function which we feed to the ILP solver is the following:

$$\sum_{i=1}^{k} V_i \sum_{j=1}^{d} \frac{P_j}{j} \sum_{m=1}^{j} w_{i,j,m} \quad (10)$$

given constraints (6, 7, 9). We used GNU MathProg [17] to model the ILP, and SCIP (Solving Constraint Integer Problems) [18] to obtain the solution to the optimization problem.

IV. Results

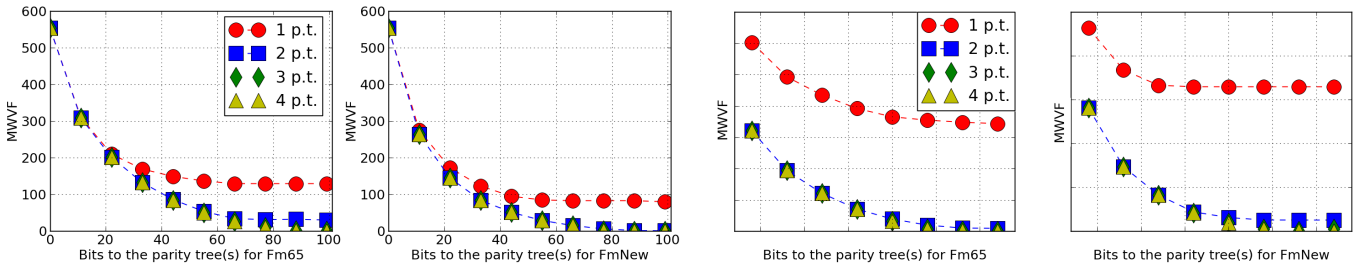
In this section, we discuss the results of the proposed parity optimization method on two different in-core memory arrays: The Alpha 21264 219-bit instruction queue, and an in-core memory array part of the P6 36-entry reservation station. Due to a non-disclosure agreement with Intel Corporation, we do not present implementation details or vulnerability factors of this in-core array. However, this is not required for this study, as the main focus is the *relative vulnerability reduction* achieved by selecting the optimal percentage and distribution of bits to include in the parity trees. The AVF numbers used for the two in-core memory arrays are obtained through the method described by the authors of [19].

Two different fault models, representing the probability of an N-Bit Upset (BU), are used in this study:

- `Fm65`: 45% 1BU, 18% 2BUs, 10% 3BUs, 27% 4BUs (taken from [8] for 65nm memories)
- `FmNew`: 20% 1BU, 16% 2BUs, 16% 3BUs, 16% 4BUs, 16% 5BUs, 16% 6BUs

A. MWVF reduction for various configurations

Optimal selection of parity bits: Figs. 3a and 3b present the MWVF reduction obtained by adding a increasing number of bits to the parity trees, for the Alpha instruction queue and the P6 reservation station respectively, for each of the two fault models. Zero bits indicates that no parity is added. The axis



(a) Alpha 21264 instruction queue

(b) Intel P6 Reservation Station structure

Fig. 3. MWVF reduction for different configuration of parity trees

values are omitted for the P6, as vulnerability estimates for the Intel microprocessor cannot be disclosed. However, it is clear from the graphs of both designs, that a careful construction of parity trees can lead to a significant vulnerability reduction. For example, adding 77 out of 219 bits of the Alpha array to the parity tree reduces vulnerability by 93%. This allows the designer the select the optimal number and distribution of bits to the parity trees in order to meet cost and reliability goals.

We note that, in case the desired fault coverage for the Alpha in-core memory array is 100%, a configuration of 99 bits split among two parity trees offers complete immunity to faults.

Effect of adding parity trees: As a rich set of MBUs is introduced using the `Fm65` and `FmNew` fault models, it is clear on Figs. 3a and 3b that for both microprocessors one parity tree has limited potential for efficient MBU protection. This difference is more apparent in the P6 in-core memory array, where inclusion of a second parity tree leads to an immediate additional MWVF reduction of 50%, even for a very small numbers of bits added to the parity trees. Furthermore, the overall MWVF reduction achieved by using only one parity tree saturates after a certain point. For the Alpha instruction queue, addition of more than 66 bits to the single parity tree does not decrease the vulnerability of the structure; however, 44 bits splitted into two parity trees offers better protection to MBUs than 66 or more bits in a single parity tree (80 MWVF vs. 140 MWVF). This key observation highlights the necessity of formulating the parity problem as an ILP, as the optimal selection and distribution of bits to parity trees is not intuitive.

Another key observation, concerning the number of parity trees, is that adding more than 2 parity trees does not offer significant MWVF reduction, even in the presence of 6-bit wide MBUs (`FmNew` model). Since three and four parity trees significantly increase area overhead, this observation allows us to limit the number of parity trees to two. Evidently, this holds true for the selected distribution of faults. As the distribution of MBUs may change dramatically in future nodes, the ILP will be able to handle and identify the need for more than two parity trees.

B. Parity overhead

Table I presents the area and delay overheads of protecting the Alpha 21264 instruction queue for a different number of bits added to the parity trees. The instruction queue was synthesized using Synopsys Design Compiler.

Area overhead: Using Table I, we can select the most desired parity tree configuration to protect against MBUs. As expected, the area overhead increases linearly as more bits are added to the parity tree, which in turn increases the number of XOR gates required. However, adding a second parity tree adds very small area overhead, as only a flip-flop is added per word.

Delay overhead: Similarly, in terms of delay overhead, increasing the depth of the parity tree increases the time required to calculate the word parity. However, adding two parity trees has a considerable advantage, as the depth of the XOR trees decreases and the delay overhead is significantly reduced.

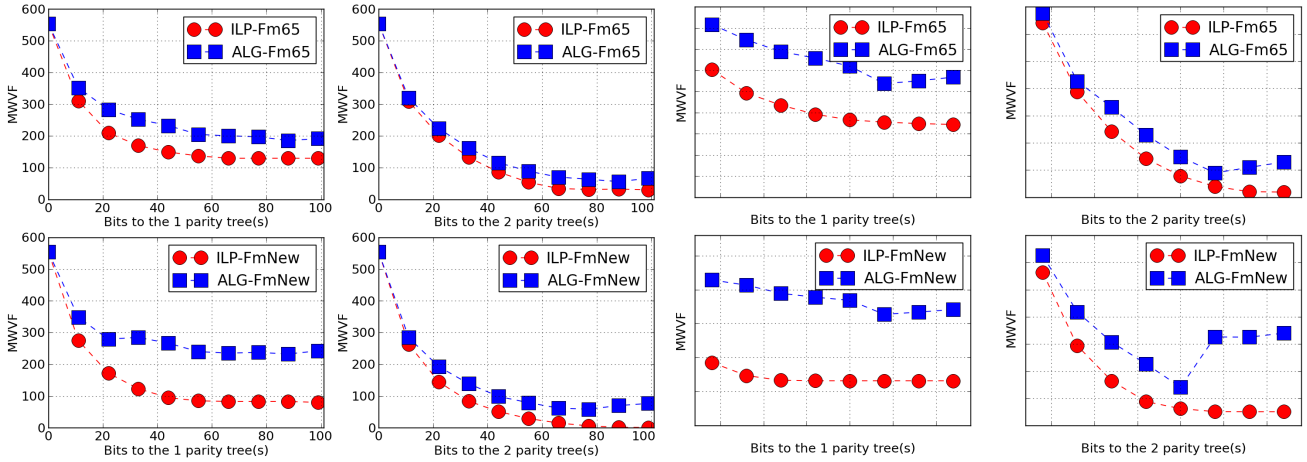
Therefore, since in the previous section we identified 2 parity trees as sufficient, possible candidates for the most cost-effective resiliency enhancement should be selected among the solutions involving 33, 44, 55 or 66 bits and 2 parity trees (shown in boldface in Table I), which offer a MWVF reduction of 67%, 78%, 87% and 92% respectively.

C. Comparison to simple algorithm

In this section, we discuss the quality of the solution obtained by the ILP solver, as compared to the simple algorithm described in Section II. Figs. 4a and 4b present the MWVF reduction achieved by the solutions obtained by the solver (ILP-) and the simple algorithm (ALG-), for the `Fm65` and `FmNew` fault models, and for the Alpha and P6 in-core memory arrays, respectively.

As expected, the solution obtained by the ILP solver is always better than that of the simple algorithm for all configurations of parity trees. Moreover, the simple algorithm yields very poor results when selecting the subset of bits to add to 1 or 2 parity trees, for all structures and fault models. For example, the average MWVF reduction obtained by the ILP for one parity tree in the Alpha array is 83%, much higher than the 50% achieved by the algorithm. This is attributed to the effect presented in Section II, where exclusion of a bit from the trees can lead to better protection from MBUs.

Furthermore, the simple algorithm exhibits an interesting artifact when more than 30% of parity bits are included in 1 or 2 parity bits; adding more bits *increases* the vulnerability of the in-core memory arrays. Indeed, adding more parity bits to one tree increases the density of protected bits; thus, blindly adding them to the tree increases the probability of error masking, as more errors resulting in an even number of bit-flips are introduced.



(a) Alpha 21264 intruction queue

(b) Intel P6 Reservation Station structure

Fig. 4. MWVF reduction of ILP solution compared to simple heuristic algorithm

TABLE I
OVERHEAD FOR DIFFERENT PARITY SCHEMES FOR THE ALPHA 21264 INSTRUCTION QUEUE

% of protected bits	1 Parity tree			2 Parity trees		
	Logic overhead	Delay overhead	MWVF reduction	Logic overhead	Delay overhead	MWVF reduction
5%	0.02%	2.27%	34.29%	0.13%	1.70%	42.41%
10%	0.20%	6.70%	52.16%	0.31%	4.38%	53.06%
15%	1.85%	8.15%	65.88%	**1.96%**	**5.36%**	**67.32%**
20%	3.99%	8.40%	76.53%	**4.10%**	**5.30%**	**78.33%**
25%	5.53%	8.85%	84.83%	**5.65%**	**5.60%**	**87.00%**
30%	6.79%	9.02%	89.72%	**6.91%**	**5.99%**	**92.23%**
35%	8.01%	9.05%	93.14%	8.12%	5.84%	94.40%
40%	8.76%	9.08%	95.48%	8.87%	5.96%	98.73%
45%	10.66%	9.22%	96.57%	10.76%	6.01%	100%

V. CONCLUSION

Recent radiation-induced experiments in contemporary technology nodes reveal a significant increase in multiple bit upsets, highlighting the need for revisiting vulnerability analysis and developing novel methods for protecting modern microprocessor in-core memory arrays against MBUs. To this end, we propose AVF-driven parity selection as an efficient method for detecting single and multi- bit upsets, and we introduce an ILP formulation of the parity tree construction optimization problem. Experimentation with several multi-bit fault distributions injected into in-core memory arrays of the Alpha 21264 and the Intel P6 instruction schedulers elucidates that optimal single tree parity selection can achieve great vulnerability reduction, even when only a small number of bits are added to the parity trees. Furthermore, the effective exploration of the solution space allowed by the ILP formulation revealed that the presence of 2 parity trees offers a vulnerability reduction of more than 50% over a single parity tree. Finally, the solutions obtained by the ILP solver are significantly better than simple, intuitive heuristics, highlighting the usefulness of the ILP formulation.

REFERENCES

[1] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2742–2750, 1996.

[2] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[3] N. Seifert et al., "Radiation-induced soft error rates of advanced CMOS bulk devices," in *IEEE International Reliability Physics Symposium Proceedings*, 2006, pp. 217–225.

[4] R.A. Reed et al., "Heavy ion and proton-induced single event multiple upset," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2224–2229, 1997.

[5] R. Koga, S.D. Pinkerton, T.J. Lie, and K.B. Crawford, "Single-word multiple-bit upsets in static random access devices," *IEEE Transactions on Nuclear Science*, vol. 40, no. 6, pp. 1941–1946, 1993.

[6] A.D. Tipton et al., "Multiple-bit upset in 130 nm CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3259–3264.

[7] Y. Tosaka et al., "Comprehensive study of soft errors in advanced CMOS circuits with 90/130 nm technology," in *IEEE International Electron Devices Meeting*, 2004, pp. 941–944.

[8] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, and F. Ruckerbauer, "Investigation of increased multi-bit failure rate due to neutron induced SEU in advanced embedded SRAMs," in *IEEE Symposium on VLSI Circuits*, 2007, pp. 80–81.

[9] J. Abella, R. Canal, and A. Gonzalez, "Power- and complexity-aware issue queue designs," *IEEE Micro*, vol. 23, no. 5, pp. 50–58, 2003.

[10] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[11] A. Buyuktosunoglu et al., "Tradeoffs in power-efficient issue queue design," in *International Symposium on Low Power Electronics and Design*, 2002, pp. 184–189.

[12] C.W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397–404, 2005.

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

[14] S. Almukhaizim, P. Drineas, and Y. Makris, "Entropy-driven parity-tree selection for low-overhead concurrent error detection in finite state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 8, pp. 1547–1554, 2006.

[15] N.A. Touba and E.J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 7, pp. 783–789, 1997.

[16] S.S. Mukherjee et al., "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *IEEE International Symposium on Microarchitecture*, 2003, pp. 29–40.

[17] A. Makhorin, "Modeling language gnu mathprog," *Relatório Técnico, Moscow Aviation Institute*, 2000.

[18] T. Achterberg, "SCIP: Solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[19] M. Maniatakos et al., "Global signal vulnerability (GSV) analysis for selective state element hardening in modern microprocessors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1361–1370, 2012.