Reliability-Aware Instruction Set Customization for ASIPs with Hardened Logic

Unmesh D. Bordoloi[1], Bogdan Tanasa[1], Mehdi B. Tahoori[2],
Petru Eles[1], Zebo Peng[1], Syed Z. Shazli[3], Samarjit Chakraborty[4]
[1]Linköpings Universitet, Sweden [2]Karlsruhe Institute of Technology, Germany
[3]Northeastern University, USA [4]TU Munich, Germany
[1]E-mail:{unmesh.bordoloi, bogdan.tanasa, petru.eles, zebo.peng}@liu.se
[2]E-mail: {mehdi.tahoori@kit.edu}
[3]E-mail: {sshazli@ece.neu.edu} [4]E-mail: {samarjit@tum.de}

***Abstract*—Application-specific instruction-set processors (ASIPs) allow the designer to extend the instruction set of the base processor with selected *custom* instructions to tailor-fit the application. In this paper, with the help of a motivational example, we first demonstrate that different custom instructions are vulnerable to faults with varying probabilities. This shows that by ignoring the vulnerability to faults, traditional methods of instruction set customization can provide no guarantees on the reliability of the system. Apart from such inherent disparity in error vulnerability across custom instructions, each custom instruction can have multiple implementation choices corresponding to varying hardened levels. Hardening reduces the vulnerability to errors but this comes at the overhead of area costs and reduced performance gain. In this paper, we propose a framework to select custom instructions and their respective hardening levels such that reliability is optimized while the performance gain is satisfied and area costs are met as well. Our framework is based on a novel analytical method to compute the overall system reliability based on the probability of failure of individual instructions. Wide range of experiments that were conducted illustrate how our tool navigates the design space to reveal interesting tradeoffs.**

I. Introduction

Application-specific instruction-set processors (ASIPs) allow the user to extend the instruction set of the base processor with custom instructions to tailor fit the application requirements. Typically, frequently executed subgraphs of the program's dataflow graph (DFG) are chosen as custom instructions. Some examples of commercial customizable processors include Lx [13], Xtensa [15], Strech S5 [3] among others. Fig. 1 illustrates the DFG of the *blowfish* application from the MiBench benchmark [16]. Individual nodes (like AND, ADD, XOR, and >>) in the DFG represent base instructions. Custom instructions are shown as shaded areas subsuming multiple base instructions and, thus, consist of a pattern of the base instructions.

Design automation tools and methodologies for customizing the instruction sets of ASIPs have focused on optimizing the design for metrics like performance, power and area. Such tools implicitly assume that the processor functions without any faults. With aggressive scaling of electronic components, however, processors are also susceptible to faults and hence,

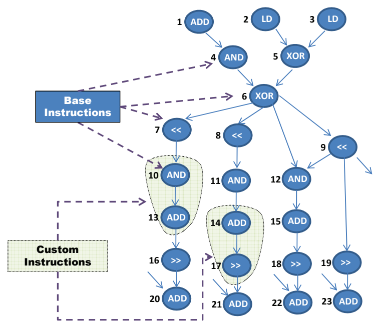


Fig. 1. DFG of the *blowfish* from the MiBench benchmark.

such tools can provide no guarantees on the reliability of the system. Moreover, embedded systems are now being widely deployed in health-care devices, automotive electronic systems and other safety critical devices, thereby making system reliability an important design concern.

In this paper, we propose an instruction set customization framework to optimize system reliability, apart from satisfying the metrics of performance and area. We assume that the processor operates in the presence of *transient* faults. Transient faults appear for a very short duration, cause miscalculations in the logic, data corruption, and then disappear without permanent damage to the circuit. These faults can be caused by radiation, fabrication process, voltage and temperature (PVT) variations and they erroneously change the output of a gate or value of wire from 0 to 1 or from 1 to 0. We focus on transient faults because the rate of transient faults is significantly higher than for other types of faults, making them a primary concern for system reliability [25].

Our contributions: The contributions of our paper are enumerated as follows.

- Our paper demonstrates that there is an inherent disparity in the vulnerability of different custom instructions to transient faults. Towards this, in Section II, we use the *blowfish* application (see Fig. 1) as a motivational example to show that different custom instructions are vulnerable to faults with different probabilities. This establishes that, by ignoring the susceptibility of custom instructions to faults, current tool-chains and methodologies for instruction set customization cannot provide reliability guarantees.

- Apart from the above-mentioned inherent disparity in error vulnerabilities across custom instructions, each custom instruction can have multiple implementation choices corresponding to various hardening levels. Hardening of logic implies reduction in its vulnerability to soft errors. While hardening might mitigate the probability of error of custom instructions, this comes at cost overheads in terms of performance and area. In this paper, we propose to leverage hardened custom instructions to enhance system reliability in a cost-effective fashion.
- We present a framework to select custom instructions and their respective hardened versions such that the design is optimized for reliability while the constraints on performance and area are satisfied. Our framework is based on Constraint Logic Programming (CLP) [2]. CLP allows us to write constraints in a logic programming framework and solves the problem using branch and bound search based on constraint programming. It should be noted that the framework is flexible in the sense that the underlying analytical framework remains unchanged if the design needs to be optimized for a different metric (like performance) and the reliability is instead given as a design constraint. Our framework is based on a novel analytical method (Section IV) that connects the probability of failure of custom instructions at various hardened levels and the overall system reliability. Our analysis takes into account the varying frequency with which each kernel is invoked in an application run in order to accurately compute the contribution of each kernel to the overall system reliability.
- Finally, we also present an efficient heuristic that scales with larger problem instances because the CLP-based branch and bound search does not scale to large problems. This is direct consequence of the fact that the optimization problem tackled in this paper is computationally expensive.

Related work: The previous decade has seen a flurry of research activities in the domain of ASIPs. Lot of research has been devoted to custom instruction *selection* techniques so as to optimize either performance or hardware area [12], [18], [5], [8], [7]. Algorithms to expose the tradeoffs between performance and area were reported in [9]. However, none of these approaches have considered reliability issues because they assume a fault free functionality. This inhibits the applicability of ASIPs to safety critical domains like health-care and automotive electronics.

Note that in this work we also focus on the custom instruction *selection* problem and like the above lines of work, we too assume that a library of custom instruction candidates is given. Such a library of custom instructions may be *enumerated* by extracting frequently occurring computation patterns from the data flow graph of the program. Recent advances in the custom instruction enumeration techniques are reported in [11], [23], [20], [6], [30].

It should be mentioned here that there have been some efforts to enhance system reliability by applying various fault recovery techniques at the instruction-level [10], [26]. However, our work differs significantly from them. First and foremost, these works are neither targeted towards customizable processors nor do they focus on instruction selection. Hence, they address a completely different problem setting. Secondly, they rely on simulation based methods and are unlike the analytical methods proposed in this paper. Finally, they do not consider hardening alternatives or disparities in the error vulnerabilities of instructions to achieve an overall system reliability.

II. Disparity in Error Vulnerabilities

In this section, we will first demonstrate that there is a significant variation in the error vulnerabilities (probabilities of failure) across custom instructions. This underscores the importance of considering reliability explicitly as a design metric when customizing instruction sets for ASIPs. It is known that hardening techniques can improve the vulnerabilities of hardware to errors. Thus, varying levels of hardening would imply varying error vulnerability of instructions and this will be discussed in the later part of this section. We would like to mention that our goal here is not to propose a new technique to compute error probabilities. Rather, we demonstrate the need for a reliability-aware instruction set customization framework.

A. Inherent Disparity

Error vulnerabilities, i.e., probabilities of error of instructions can be computed by methods discussed in [28], [1]. These probabilities typically depend on two factors — (i) the probability that an error occurs, i.e., the value of a bit is incorrectly flipped from 0 to 1 or from 1 to 0 and (ii) the *Error Propagation Probabilities* (EPPs), i.e., the probability that this fault will result in an observable error at the primary output of the circuit (in our case, the output of the instructions). The first, i.e., the probability that an error occurs, depends on the raw error rate [4], [21] and instruction execution time, among other factors. The raw error rate can be expressed in terms of failure in time (FIT). FIT is equal to one failure in a billion hours (10^9) of operation. For instance, the FIT rates for Xilinx SRAM based-based FPGA devices have been reported to be up to 1000 FIT/Mbit for 90-nm technology [19]. The raw error rate depends on the device characteristics and the environment. The overall raw FIT rate for a custom instruction is proportional to the number of gates used for the implementation of that instruction (i.e., the area of that instruction). Since the hardware area required for implementing custom instructions vary significantly, the FIT rate and, in turn, the probability that an error occurs will also vary from one instruction to another.

Independently of the overall FIT rate, the second factor, i.e., EPP also varies across instructions. We will illustrate this

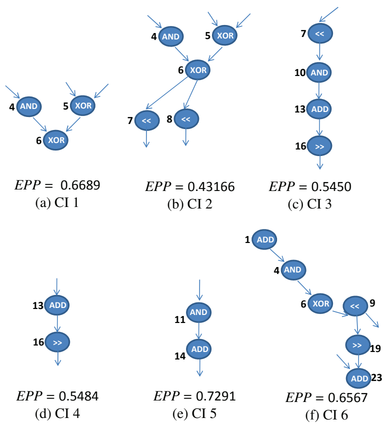


Fig. 2. Error Propagation Probabilities (EPPs) for 6 instruction patterns of *blowfish*.

for the *blowfish* application from the MiBench benchmark (Fig. 1). This result, along with the factors discussed above, establish that the probabilities of failure of instructions vary in an irregular fashion and motivates the need for new techniques for instruction set customization methods like the one proposed in this paper. In the following, we first introduce the notion of EPP, then describe a methodology used to calculate the EPP values, and finally, we discuss the results obtained by calculating the EPP values on a benchmark.

Error Propagation Probability: The EPP of a logic circuit is the probability that an error in the components of the logic block will propagate to a primary output of the block and results in an observable error. As mentioned in Section I, we consider transient faults and, hence, our error model is a logical bit-flip. A bit-flip causes an error if the effect of the bit-flip is observable at the primary output of the circuit. Given a bit-flip occurring on a wire, the EPP for this wire is defined as the ratio between the number of input combinations which resulted in an error and the total number of possible input combinations. EPP computation is based on an all-solution SAT solver (RELSAT [27]). Due to space constraints, we will not discuss the details of EPP computation here. We refer the reader interested in the details on EPP computation to [28].

Results: As mentioned before, custom instructions are essentially a pattern of instructions. We choose 6 instruction patterns from the DFG (Fig. 1) of the *blowfish* application obtained from the MiBench benchmark [16]. We apply the above methodology to compute the EPP of these 6 patterns. The EPPs obtained are shown in Fig. 2. The EPP values can be seen to vary from around 0.43 to 0.73, which clearly establishes that the error probabilities of different custom instructions are different. We already discussed that the raw error rates for different custom instructions are different. Thus, both factors contributing to the probability of failure of instructions have different values, thereby establishing the disparity in error vulnerabilities of custom instructions. Thus, various custom instructions contribute to the overall system reliability in varying magnitudes.

B. *Disparity Based on Hardening Levels*

Above, we discussed the inherent disparities in error probabilities across custom instructions, i.e., we did not factor in the possibility of hardening. Hardened hardware is less vulnerable to transient faults. A variety of techniques for hardening hardware logic and circuits have been proposed in the literature [22], [29], [24], [14]. Most of these techniques rely on selective resizing of gates or transistors in order to enhance the immunity to soft errors. We note, however, that while they reduce the probability of error, all of them incur additional area overhead and suffer performance degradation. Typically, such techniques report multiple hardening levels with decreasing error probabilities but increasing area and performance overheads. Thus, in ASIPs that allow such hardening, each custom instruction maybe implemented in one out of many feasible hardening levels. This implies that, apart from disparities in probability of failure across custom instructions (as discussed in Section II-A), there is another dimension of disparity in error vulnerabilities within each custom instruction based on its hardening levels.

It is important to note that, apart from the disparity in error probabilities, different custom instructions are invoked for different number of times by the application. This is yet another factor that leads to the varying contribution of the custom instructions to the overall system reliability. Hence, the frequency of the custom instructions being invoked must be considered as well by any reliability-aware technique for instruction set customization.

Example: With an illustrative example, we shall now discuss how the interplay of various factors like probability of failure, frequencies and hardening levels of a custom instruction impacts the system reliability. For simplicity of elucidation, let us consider that we have only two custom instruction candidates CI_1 and CI_2 with probability of failures $p_1 = 8 \times 10^{-2}$ and $p_2 = 7 \times 10^{-2}$ respectively. Let us assume that the frequency of execution of each instruction, i.e., the number times the instructions are executed in one run of application are $f_1 = 3$ and $f_2 = 6$, the performance gains achieved are $G_1 = 4$ and $G_2 = 2$ for one invocation of each instance and the area requirements are $A_1 = 4$ and $A_2 = 2$. Let us consider that the area is in number of LUTs and gain is in units of clock cycles.

The total performance gain achieved by selecting CI_1 or CI_2 is the same, i.e., $f_1 \times G_1 = f_2 \times G_2 = 12$. Let us consider that for the design at hand, the area budget is restricted to 5 units. Then, only one of either CI_1 or CI_2 maybe selected because the combined area costs of CI_1 and CI_2 is 6 units. Our goal is to select the one that yields higher reliability. CI_2 has a lower probability of error and seems a promising candidate. However, when we consider the frequency of executions of the instructions, CI_2 might not be the optimal selection. The probability of failure of CI_1, considering all its executions, is $1-(1-p_1)^{f_1}$=0.2213.

On the other hand, overall failure of CI_2 evaluates to be $1-(1-p_2)^{f_2}$=0.3530. Thus, even though $p_1 > p_2$, considering all execution instances, CI_2 has a higher probability of failure than CI_1 in a run of the application. Hence, CI_1 should be selected in this case instead of CI_2.

Above, we did not consider any hardening alternatives. Let us now assume that CI_1 has no possible hardening levels as alternative implementations but CI_2 has one level of hardening alternative, denoted by CI_2^H, with following parameters $p_2^H = 4 \times 10^{-2}$, $A_2^H = 5$ and $G_2^H = 1$. Thus, hardening improves the probability of failure but it comes with increased area costs and degradation in possible performance gains. The overall probability of failure of CI_2 is now computed to be 0.2172, that is less than that of CI_1. Thus, instead of CI_1, selecting CI_2's hardened implementation is optimal from the reliability perspective. However, this comes at the overhead of performance because instead of the performance gain of 12 clock cycles that was possible with CI_1, the performance gain with CI_2^H is only $f_2 \times G_2^H = 6$ clock cycles.

The above example illustrates the intricate relationship, between probability of failure, frequency of execution of the instructions and their hardening levels, that must be captured while computing the reliability. It also illustrates that the tradeoffs between reliability, performance gain and area that must be accounted for during custom instruction selection. Our proposed framework captures the relevant relationships between various parameters and systematically evaluates the tradeoffs between various optimization objectives. We note that in the above example, for simplicity, we assumed that no base instructions contribute to the system reliability. In our proposed framework, we will explicitly take into account the probability of failure of base instructions and their execution frequencies. Note that the probability of failure of base instructions can also be computed in a similar manner as discussed in Section II-A.

III. System Model

Given an application to be run on a customizable processor, we assume that the library of custom instruction candidates for this application is known to us. Let there be N custom instruction candidates denoted by the set $CI = \{C_1, C_2, \ldots, C_N\}$. A custom instruction C_i has n_i instances in the application, and we denote them as $c_{i,1}, c_{i,2}, \ldots, c_{i,n_i}$. The execution frequency $fc_{i,j}$ (i.e., the number of times an instruction is executed in one run of the application) of each instance is also known to us. This is input dependent and can be obtained by profiling the execution trace of the application on a large set of inputs. For a custom instruction C_i let us say that there are r_i versions for implementations considering all possible hardening levels. Without loss of generalization, we assume that the non-hardened version is one of the r_i versions. For each of these levels, let $\{G_{i,1}, G_{i,2}, \ldots, G_{i,r_i}\}$ denote the gain in performance obtained by one instance of the custom instruction C_i. Similarly, let the area overheads be $\{A_{i,1}, A_{i,2}, \ldots, A_{i,r_i}\}$. We note that a problem that does not consider hardening would have $r_i = 1$. Thus, our problem formulation is quite general that allows us to consider both hardening and non-hardening scenarios in a seamless fashion.

Note that the processor core has a set of existing instructions. These instructions can be categorized into two sets — (i) BC is the set of existing instructions that can be covered by a custom instruction instance and (ii) BI is the set of instructions that cannot be covered by any custom instruction instance. For any existing instruction $b_k \in (BC \cup BI)$, the execution frequency of that instruction is given by fb_k^b. In other words, BC is the set of the existing instructions that maybe subsumed by the custom instruction candidates, while BI is the set of instructions that cannot be subsumed by any custom instruction candidate.

We assume that the probabilities of failure of all the custom instruction candidates as well as the existing instructions are known to us. Such probabilities can be obtained using techniques described in Section II. The probability of failure of a custom instruction C_i, when implemented at hth hardening level, is denoted by $pc_{i,h}$, where $C_i \in CI$ and $h \in \{1, 2, \ldots, r_i\}$. The probability of failure of an existing instruction b_k is denoted by pb_k, where $b_k \in (BC \cup BI)$. A higher probability for an instruction implies that this instruction is more vulnerable to transient faults.

Problem statement: In this paper, our goal is to choose a set of custom instructions and their respective hardened levels, such that the reliability of the application is optimized while the desired performance gain is achieved and the area constraints are satisfied. In other words, the goal is to minimize the overall probability of error of the application. Towards this, a Constraint Logic Programming (CLP) [2] based formulation is presented in Section IV. CLP allows users to write constraints in a logic programming framework and solves the problem using branch and bound search. However, this is computationally expensive and hence, we will also present an efficient heuristic. We would like to emphasize here that the underlying analysis remains unchanged if a different optimization objective (like performance gain) is chosen instead of reliability and the desired reliability is specified as a constraint. Note that typically if a custom instruction is selected, all instances of that instruction are run in dedicated hardware to maximize the performance gain. However, custom instructions might have higher probability of error than base instructions. This might be the case, for instance, when the base instructions run on a core that is more hardened than the dedicated hardware for custom instructions. Such a scenario is feasible, for example, when the application is running on a soft-core processor in the FPGA logic, and both base instructions and custom instructions are implemented using FPGA logic. In such a scenario, invoking all instances of custom instructions might lead to very low reliability. Hence, our problem formulation is quite general and allows the flexibility to select the instances of each custom instruction.

IV. Proposed Framework

Our proposed scheme is illustrated in Figure 3. We first propose an analysis that connects the error probabilities of the custom instructions to the overall reliability.

A. Probability Analysis

In this section, our goal is to compute the overall probability of failure for the system. Note that this probability gives us a measure of the unreliability of the system. First, let us consider the custom instructions. The probability of failure of an instance, $c_{i,j}$ of a custom instruction C_i is given by $pc_{i,h}$ for its hth hardening level. Thus, the probability that it executes successfully is $1 - pc_{i,h}$. Given the execution frequency $fc_{i,j}$ of $c_{i,j}$ and considering the hth hardening level, the probability that the instance executes successfully each time it is invoked is thus given by:

$$(1 - pc_{i,h})^{fc_{i,j}} \tag{1}$$

Now we will derive an expression that denotes the probability of failure for the instance $c_{i,j}$ considering all possible hardening levels. Towards this, let there be r_i boolean variables, $x_{i,j,1}, x_{i,j,2}, \ldots, x_{i,j,r_i}$ associated with an instance $c_{i,j}$ where if $x_{i,j,h}$ is 1, it implies that the instance $c_{i,j}$ of the custom instruction C_i has been selected to implemented in the hth hardening level. If $x_{i,j,h}$ is 0, it implies otherwise. This allows us to derive the following expression as the probability of failure for the instance $c_{i,j}$ considering all possible hardening levels.

$$\prod_{h=1}^{r_i} (1 - pc_{i,h})^{x_{i,j,h} \times fc_{i,j}} \tag{2}$$

We note that at most one of the r_i boolean variables might be true for $c_{i,j}$ because at most one hardening level can be chosen for an instance. This constraint will be described formally in next section as part of our CLP formulation (see Equation 12 and Equation 13).

Considering all instances of the custom instruction C_i, that are selected, the probability that they execute without faults is given by:

$$\prod_{j=1}^{n_i} \prod_{h=1}^{r_i} (1 - pc_{i,h})^{x_{i,j,h} \times fc_{i,j}} \tag{3}$$

Now, let us consider all custom instructions. The probability that they execute correctly without any fault is:

$$P_{CI} = \prod_{i=1}^{N} \prod_{j=1}^{n_i} \prod_{h=1}^{r_i} (1 - pc_{i,h})^{x_{i,j,h} \times fc_{i,j}} \tag{4}$$

In Eq. 4, we computed the probability that all the customized instructions execute successfully. Now let us consider the base instructions. From Section III, we recall that $BC = \{B_1, B_2, \ldots, B_M\}$ is the set of existing instructions that can be covered by a custom instruction instance. However, our framework allows the flexibility that all instances of a custom instruction need not run in hardware, and thus, there will be some instances of the instructions in the set BC which will run in the existing processor core. The frequencies with which the base instructions occur are given by $fb_1, fb_2, \ldots, fb_M$. However, because some instances of these base instructions might be covered by the custom instructions, the base instruction will now execute with less frequency. Let this number be given by fb'_k for an instruction B_k. fb'_k can be computed as follows.

$$fb'_k = fb_k - \sum_{i=1}^{|CI_k|} \sum_{j=1}^{n_i} \sum_{h=1}^{r_i} x_{i,j,h} \times fc_{i,j} \tag{5}$$

In the above equation, CI_k is the set of custom instruction candidates that covers the base instruction B_k. Essentially, we subtracted the number of times all the custom instructions have covered the base instruction. Once fb'_k is computed, the probability that these base instructions execute successfully is given by:

$$P_{BC} = \prod_{k=1}^{M} (1 - pb_k)^{fb'_k} \tag{6}$$

Finally, there are base instructions that have no custom instruction candidates and are denoted by the set BI. The probability that instructions in the set BI execute successfully is given by:

$$P_{BI} = \prod_{k=1}^{|BI|} (1 - pb_k)^{fb_k} \tag{7}$$

We note that the above term P_{BI} is constant but we account for it in the optimization framework to obtain accurate results. Thus, the overall probability that all the instructions (customized and base instructions) execute without failures is:

$$GP = P_{CI} \times P_{BC} \times P_{BI} \tag{8}$$

The overall probability of failure for the system, i.e., a measure of unreliability is then given by $1 - GP$.

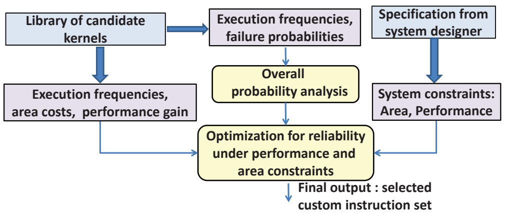

Library of candidate kernels
Execution frequencies, failure probabilities
Specification from system designer
Overall probability analysis
Execution frequencies, area costs, performance gain
System constraints: Area, Performance
Optimization for reliability under performance and area constraints
Final output : selected custom instruction set

Fig. 3. The overall scheme consists of a probability analysis core that is invoked by the optimization framework.

B. CLP-based Optimal Approach

We now formulate the custom instruction selection problem as an optimization problem in CLP, where the optimization objective is reliability, i.e., the minimization of $(1 - GP)$. As an output of the CLP, we obtain the set of selected custom instructions and their respective hardening levels. In the following we describe the CLP constraints.

Performance constraint: The first constraint is that the desired performance gain must be achieved by selecting the custom instructions, i.e.,

$$\sum_{i=1}^{N}\sum_{j=1}^{n_i}\sum_{h=1}^{r_i}(x_{i,j,h} \times G_{i,h} \times fc_{i,j}) \geq G^T \tag{9}$$

Here G^T is the total performance gain that the designer wants to achieve with the use of custom instructions. For instance, G^T may be specified as a certain fraction of G_{max}, where G_{max} is the maximum achievable performance gain using all custom instructions without any area constraints. Note that G_{max} can be easily known once the library of the custom instructions has been enumerated. In certain settings, it is possible that the designer does not have an explicit constraint G^T but rather a range of possible G^T values. In such cases, our framework (both the CLP-based approach and the proposed heuristic) can be invoked iteratively for different G^T values in order to evaluate the tradeoffs between reliability versus performance gain and the designer can then choose a suitable design point from the resulting design space. This will be illustrated in more detail in our experimental results.

Area constraints: Assume that the overall hardware area available for custom instructions is constrained by R. Given a hardening level h, if at least one instance of a custom instruction C_i is chosen to be implemented at that level, then $A_{i,h}$ units of hardware cost is incurred. Let $X_{i,h}$ be a boolean variable which is true if at least one instance of C_i is chosen at h hardened version. Mathematically, we have the following,

$$X_{i,h} = \begin{cases} 1 & \text{if } \sum_{j=1}^{n_i} x_{i,j,h} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Thus, the constraint on area may be given as follows:

$$\sum_{i=1}^{N}\sum_{h=1}^{r_i} X_{i,h} \times A_{i,h} \leq R \tag{11}$$

Hardening constraints: Note that, for each custom instruction, at most one hardened version might be chosen for each custom instruction instance. Thus, out of r_i hardened levels of $c_{i,j}$, at most one may be chosen for implementation. This is ensured by the following constraint:

$$x_{i,j,1} + x_{i,j,2} \ldots + x_{i,j,r_i} \leq 1 \tag{12}$$

All the constraints discussed above and the analysis presented in Section IV-A allow the flexibility to implement different instances of the same custom instruction to be implemented in different hardening levels. If, however, we assume that all instances of a custom instruction are implemented at the same hardening level, then we must have the following additional constraint for each $c_{i,j}$,

$$x_{i,1,h} = x_{i,2,h} \ldots = x_{i,n_i,h}, \forall h \in \{1, 2, \ldots, r_i\} \tag{13}$$

Optimization goal: Our goal is to minimize the overall probability of failure.

$$min\ (1 - GP) \tag{14}$$

Note that this is equivalent to maximizing the overall reliability, i.e., the probability that all the instructions (customized and non-customized) will execute without failures.

Algorithm 1 Heuristic for optimizing reliability

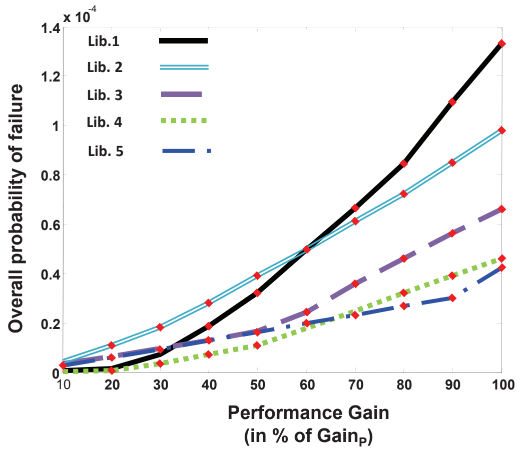
```
Input: A library of custom instruction candidates $\{c_{1,1}, c_{1,2}, \ldots, c_{i,j}, \ldots, c_{N,n_i}\}$, with probability of failures $p_{i,j}$ and area cost $A_i$
 1: $G = 0$; $PROB = \prod_{k=1}^{|BI|\cup|BC|}(1 - pb_{k,h})^{fb_k}$;
 2: $X_i = 0\ \forall 1 \leq i \leq N$; $x_{i,j} = 0\ \forall 1 \leq i \leq N, 1 \leq j \leq n_i$;
 3: for $i \in \{1, 2, \ldots, N\}$ do
 4:   for $j \in \{1, 2, \ldots, n_i\}$ do
 5:     compute $Y_{i,j} \leftarrow G_i \times fc_{i,j}/(A_i \times (1 - (1 - pc_{i,h})^{fc_{i,j}}))$
 6:   end for
 7: end for
 8: sort $Y_{i,j}$ values in descending order and enqueue $Y_{i,j} \in Q$
 9: while $Q \neq \phi$ do
10:   select first element from Q
11:   if $X_i = 1$ and $G < G^T$ then
12:     $x_{i,j} = 1$
13:     $G = G + G_{i,j} \times fc_{i,j}$
14:     $PROB = PROB \times Y_{i,j}/(1 - pb_k)^{fb_k}$ where $b_k \in BC$ and $b_k$ is covered by instance $c_{i,j}$
15:   else if $X_i \neq 1$ and ($R_i \leq R$ or $G < G^T$) then
16:     $X_i = 1$; $x_{i,j} = 1$;
17:     $G = G + G_{i,j} \times fc_{i,j}$
18:     $R = R - A_i$
19:     $PROB = PROB \times Y_{i,j}/(1 - pb_{k,h})^{fb_k}$ where $b_k \in BC$ and $b_k$ is covered by instance $c_{i,j}$
20:     remove K from Q where K is the set of hardened versions of $c_{i,j}$
21:   end if
22:   dequeue instruction $c_{i,j}$ from Q
23: end while
```

C. Heuristic Approach

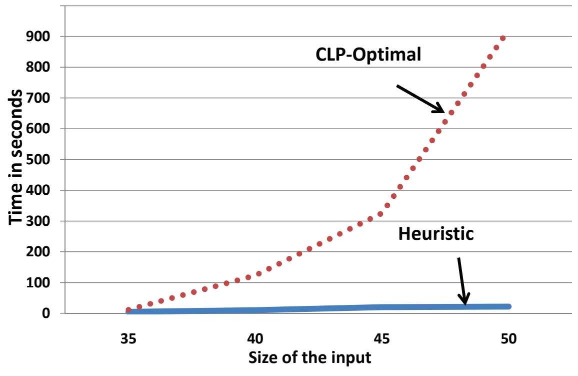
The CLP formulation described above will return optimal solutions but is computationally expensive. In this section, we propose an efficient heuristic (listed in Alg. 1) based on a greedy algorithm. We recall that our objective is to select a subset of the custom instruction instances such that the area and performance gain constraints are satisfied while the overall probability of failure is minimized. Thus, for each custom instruction instance $c_{i,j}$, the following three factors have to be taken into account — (i) the area cost of each custom instruction (A_i), (ii) the performance gain ($G_i \times fc_{i,j}$) obtained by using the custom instruction instance and (iii) the probability of failure ($1 - (1 - pc_{i,h})^{fc_{i,j}}$). We consider each custom instruction instance as a set of r_i items corresponding to its hardened levels. We sort these items (considering all custom instruction instances) based on the ratio $G_i \times fc_{i,j}/(A_i \times (1 - (1 - pc_{i,h})^{fc_{i,j}}))$ in the descending order (lines 3 to 8 of Alg. 1). Then, the items are chosen from this list as long as the sum of area does not exceed the area constraint R or the performance gain constraint G^T is not satisfied (lines 11 and 15). Note that the list of items is also updated at each iteration to ensure that at most one hardened level is chosen for any custom instruction instance. The lines 9 to 22 ensure this and the correct computation of the overall probability of failure. The time complexity of our heuristic is $O(n\ log\ n)$, where $n = \sum_{i=1}^{N} n_i \times r_i$, i.e., the total number of items.

V. Experimental Results

Several experiments were conducted to evaluate our proposed framework. The experiments show how the



(a)



(b)

Fig. 4. (a) Tradeoffs between gain in performance and reliability for 5 different inputs of custom instruction libraries. (b) Running times of the heuristic versus the optimal CLP.

proposed scheme reveals interesting tradeoffs between reliability, performance and area.

Experimental setup: Our experimental framework has been built in C++. The Constraint Logic Programming component has been developed using *Prolog* [2] and is invoked from within our C++ framework. All the experiments were conducted on a Windows 7 machine running a 4-core Xeon(R) 2.67 GHz processor. For our experiments, we generated many synthetic test cases (custom instruction libraries) with varying parameters. The number of custom instruction candidates and their instances were varied between 2 and 10 respectively in each library. The area cost for each hardware version was varied between 20 and 60 LUTs while the performance gain from each instance was varied between 30 and 90 clock cycles. The error probabilities of the custom instructions of our input sets were generated between 10^{-8} and 10^{-6}. We conducted two broad sets of experiments on our test cases — (i) without hardening (Section V-A) and (ii) with hardening (Section V-B). We will report results obtained on an industrial case study in Section V-C.

A. Without Hardening

In our first set of experiments, we did not consider any hardening alternatives for the custom instructions. Thus, for each custom instruction C_i we have $r_i = 1$. The results discussed below illustrate that even when no hardening alternatives are considered, the proposed methodology provides significant benefits.

Tradeoffs: First, we show that our framework can reveal tradeoffs between reliability and performance gain, unlike the techniques that are limited to maximizing performance. Towards this, we implemented a CLP-based framework (CLP-P) to maximize the performance gain under area constraints without considering reliability issues. Then, we followed the steps outlined below for 10 different inputs of custom instruction libraries.

1) For each input library, we recorded the maximum performance gain ($Gain_P$) that can be achieved by CLP-P without considering reliability.
2) For each input, we ran our proposed CLP based optimal framework. In contrast to step (1), the goal here is to maximize reliability with performance gain as a constraint. This constraint was set to different values, varying from $\{100\%, 90\%, 80\%, \ldots, 10\%\}$ of $Gain_P$.

Fig. 4(a) illustrates the overall probability of failure (a measure of unreliability) versus the performance gain, at $\{100\%, 90\%, 80\%, \ldots, 10\%\}$ of $Gain_P$, for 5 of our 10 input sets. If we observe the tradeoffs for the library 1 (solid curve shaded in black), the highest probability of failure occurs when the performance constraint is 100% of $Gain_P$. Note that this is the failure probability when the optimization criterion is performance (CLP-P), without any consideration for reliability. In fact, in all 10 input sets, the reliability was worst at 100% of $Gain_P$ buttressing the fact that a technique for instruction set customization that optimizes simply for performance achieves the worst reliability. On the other hand, our proposed framework can reveal tradeoffs between reliability and performance gain which is important for safety-critical applications. On the average, the results for all the 10 test cases show that by using our technique we can achieve 16% gain in reliability while sacrificing only 10% of the overall gain in performance, i.e., at the 90% of the gain that is achieved by traditional methods. This trend may be observed visually for the 5 curves plotted in Fig. 4(a).

Heuristic: For the 10 input libraries, we also compared the result (overall probability of failure) obtained from our heuristic and the CLP-based implementations. For each input library, we compared these values at 10 points by setting the performance constraint to $\{100\%, 90\%, 80\%, \ldots, 10\%\}$ of $Gain_P$. Note that the heuristic returns a probability that may be larger or equal to the probability returned by the optimal CLP. In our experiments, on the average the heuristic deviated from the optimal CLP-based solution by 13% showing that it performs well with respect to the quality of solutions. To show the scalability of our heuristic, we conducted more experiments by increasing the problem size $\sum_{i=1}^{N} n_i$ up to 50. The running times are shown in Fig. 4(b). The CLP-based formulation does not scale to large problems. On the other hand, the execution times for our heuristic scales quite well with larger problem sizes.

B. With Hardening

In our second set of experiments, we considered that there are three levels of hardening, i.e., for each custom instruction C_i we have $r_i = 3$. With increasing levels of hardening, the input characteristics varied as described in the following. The probability of failure was decreased as the hardening level increased. Thus, for the first level, the range was between 10^{-8} and 10^{-6}, for the second level the range was between 10^{-9} and 10^{-7}, and the range for the most hardened level was between 10^{-10} and 10^{-8}. The performance gain was assumed to degrade atmost 5% at each level due to hardening and the maximum area overhead increase was considered in steps of 10%.

We now illustrate how our tool can navigate the tradeoffs between performance gain and area overheads against the overall probability of failure. Towards this, Figure 5(a) and Figure 5(b) show the results obtained for one custom instruction library. Similar results were obtained with other libraries but we will focus on one for the clarity of exposition. Figure 5(a) shows three different plots, each corresponding to the case that the same level of hardening is allowed for all custom instructions. Thus, $h = 1$ illustrates the tradeoffs between reliability and performance gain when only level one hardening was allowed for all custom instructions. Similarly, $h = 2$ and $h = 3$ show plots when, respectively, only second and only third level of hardening was allowed. For each level, we followed a similar procedure as in Section V-A to conduct the experiments, i.e., we first recorded the maximum performance gain ($Gain_P$) that can be achieved by CLP-P without considering reliability. Then, we set the performance constraint to different values, varying from $\{100\%, 90\%, 80\%, \ldots, 10\%\}$ of $Gain_P$ and we ran our proposed CLP-based framework.

At each of the 10 points, the overall probability of failure decreases from $h = 1$ to $h = 2$ and from $h = 2$ to $h = 3$, as visualized in Figure 5(a). This shows that the higher hardening levels $h = 2$ and $h = 3$ can significantly enhance the overall system reliability. However, we note that for $h = 2$ and $h = 3$ the CLP solver reported no solutions at $\{100\%, 90\%\}$ of $Gain_P$. This is reflected in Figure 5(a), where these two curves are seen to be truncated at 80% of $Gain_P$. This is because as the hardening levels increase, the performance gain that can be achieved decreases. Thus, at $h = 2$ and $h = 3$ it is not possible to meet high performance demands as well as to guarantee high reliability. This result also shows the limitations of utilizing the same hardening level for all custom instructions.

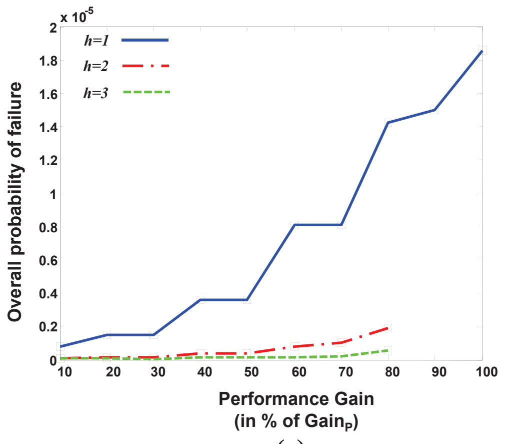
On the other hand, when varying hardening levels are allowed for different custom instructions, it might be possible to strike the right balance between reliability and performance. Figure 5(b) illustrates the reliability versus performance tradeoffs in this case. The curve ($h = X$) with solid markers in Figure 5(b) corresponds to the case when different custom instructions can utilize different hardening levels. Note that this curve ($h = X$) is superimposed on the $h = 3$ curve up to 80% of $Gain_P$. This is explained by the fact that our CLP searches for solutions with highest reliability and $h = 3$ contains such solutions. The interesting case is that, at $h = 3$, the there is no solution beyond 80% of $Gain_P$, but in the general case ($h = X$), valid solutions are found as shown in Figure 5(b). It is noteworthy that at 90% of $Gain_P$, $h = X$ reports a solution that is not reported by $h = 1$, $h = 2$ or $h = 3$. Thus, these solution points offer both high performance as well as high reliability. This example shows how our framework can reveal very interesting trends and find solutions that offer the right balance between reliability and performance.

Apart from the tradeoffs between reliability and performance, design of ASIP processors must also optimize area costs. Area overheads are even more significant when we consider hardening. Figure 6(a) and (b) shows area overhead incurred by $h = 1$, $h = 2$ and $h = 3$. As seen in Figure 6(a) the higher hardening levels have utilized more area to provide the same performance. This is a consequence of the fact that they provide more reliability at increased area overheads. Note however that given an instruction, the hardware area for its more hardened implementations is not strictly increasingly [22], [29], [24], [14]. This explains the lower area overhead for $h = 3$ in Figure 6(a) at around 30% of performance gain. Also, we note from Figure 6(a) that high performance requirements lead to high area costs as more instructions are customized.

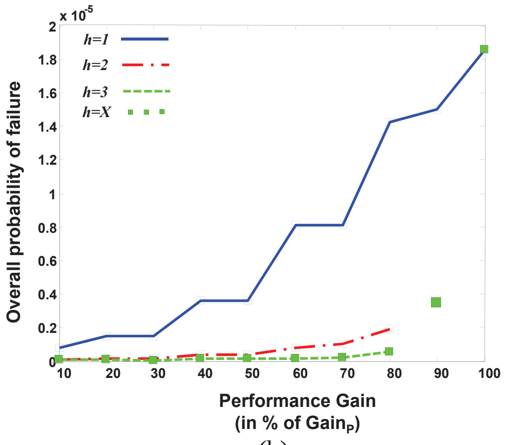
Figure 6(b) plots the tradeoffs between area and overall probability of failure for each of the 10 design points for $h = 1$, $h = 2$ and $h = 3$. For the almost same area overhead (155 LUTs) $h = 3$ provides significantly higher reliability than $h = 1$, but we know from Figure 6(a) that at this area overhead, $h = 3$ provided only 80% of the total performance while $h = 1$ guaranteed maximum performance. This illustrates the importance of navigating the tradeoffs between area, performance gain and reliability in a systematic manner, as performed by our tool. We would like to mention that our heuristic showed significant speedups as discussed before.

C. Case Study

As a case study, we studied *gsm-encoder* from the mobile application domain. We profiled the application using the LLVM suite [17]. The values of area costs of the custom instructions were obtained using Xilinx ISE WebPack considering the device XC5VLX50 Virtex-5. We considered 5 custom

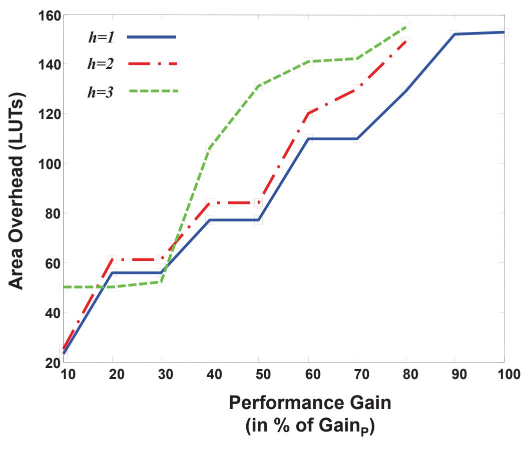


(a)

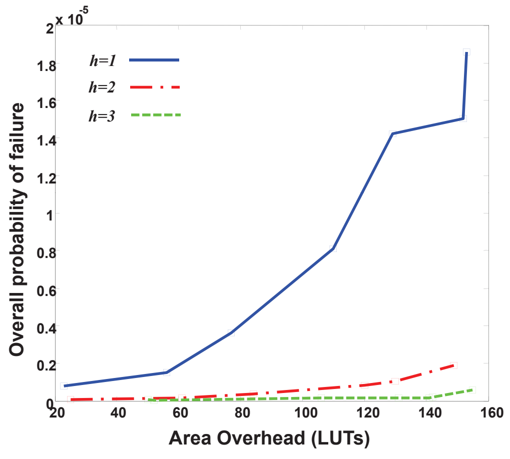


(b)

Fig. 5. The tradeoffs between performance gain and the overall probability of failure when (a) the same hardening levels is allowed for all custom instructions and when (b) varying hardening levels are allowed for different custom instructions.



(a)



(b)

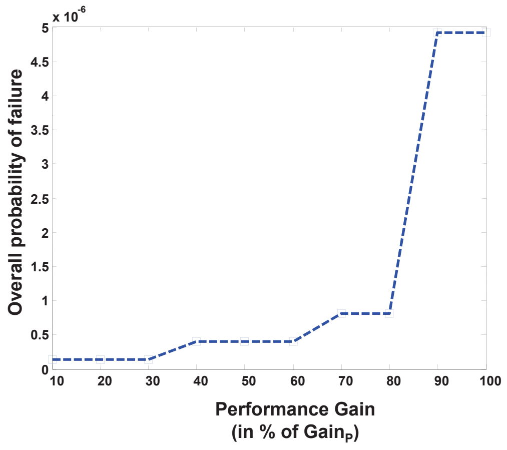
Fig. 6. The area overhead incurred by the different levels of hardening (a) plotted against performance gain and (b) plotted against the overall probability of failure.

instruction candidates $C_1, \ldots, C_5$. The area and performance gain are given in pairs (A_i, G_i) for $C_1, \ldots, C_5$ in the following set $\{(71,30),(119,50),(27,10),(73,24),(119,55)\}$. The area is in terms of LUTs and performance gain is in terms of clock cycles. For these custom instructions we first computed the EPP values using the methodology in Section II. For $C_1, \ldots, C_5$, the EPP values were, respectively, $\{0.0014152, 0.002626, 0.04107, 0.004139, 0.008329\}$. Note that there is significant variation in the EPP values and this supports our discussion in Section II about the inherent disparity in error vulnerabilities across custom instructions. In fact, EPP of C_3 is an order of magnitude different from the rest. We conducted the experiments considering no hardening (i.e., $r_i = 1$) and that the probabilities of failure of the custom instructions are directly proportional to the EPP values.

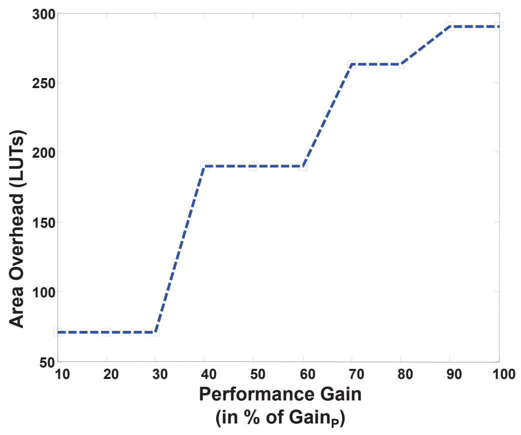
The results are shown in Figure 7(a) and (b). Figure 7(a) depicts the plot of overall probability failure versus the performance gain. In this figure, the effect of EPP variation is clearly reflected. C_3 is custom instruction with highest EPP value and it was not chosen by the tool for the design points $\{80\%, \ldots, 10\%\}$ of $Gain_P$. However, without selecting C_3, it was not possible to meet the high performance at $\{90\%, 100\%\}$ of $Gain_P$. When C_3 was selected to increase the performance, it significantly contributed to high unreliability as reflected by the spike in Figure 7(a). Figure 7(b) shows the increasing area overheads with increasing performance gains, as is expected.

VI. Conclusion

We proposed a instruction set customization technique for *reliable* designs in the presence of transient faults. Our analysis takes into account the variability in error tolerance across custom instructions and hardening levels in order to optimize reliability. Our framework can be also be used to optimize for performance or for evaluating tradeoffs between performance and reliability as illustrated in our experimental results. It will be interesting to utilize other techniques



(a)



(b)

Fig. 7. The tradeoffs between reliability, performance gain and area overhead for the case study.

like triple-modular redundancy as well to provide reliability guarantees for ultra-reliable systems. It is also worthwhile to explore possibilities of hardware resource sharing to effectively utilize area constraint. Note that such techniques are orthogonal to the approach presented in this paper and may build upon the foundations introduced here.

Acknowledgements: This work was partially supported by the Swedish Foundation for Strategic Research. It was supported in parts by the German Research Foundation (DFG) as part of the priority program ”Dependable Embedded Systems” (SPP 1500 -spp1500.itec.kit.edu).

References

[1] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. Goofi: Generic object-oriented fault injection tool. In *DSN*, 2001.

[2] K. R. Apt and M. G. Thiran. *Constraint Logic Programming using* ECL^iPS^e. Cambridge University Press, 2007.

[3] J. M. Arnold. S5: The architecture and development flow of a software configurable processor. In *FPT*, 2005.

[4] H. Asadi and M. B. Tahoori. Analytical techniques for soft error rate modeling and mitigation of FPGA-based designs. *IEEE Trans. Very Large Scale Integr. Syst.*, 15(12):1320–1331, 2007.

[5] K. Atasu, R. G. Dimond, O. Mencer, W. Luk, C. Özturan, and G. Dündar. Optimizing instruction-set extensible processors under data bandwidth constraints. In *DATE*, 2007.

[6] K. Atasu, O. Mencer, W. Luk, C. Ozturan, and G. Dundar. Fast custom instruction identification by convex subgraph enumeration. In *CODES+ISSS*, 2008.

[7] L. Bauer, M. Shafique, S. Kramer, and J. Henkel. Rispp: Rotating instruction set processing platform. In *DAC*, 2007.

[8] P. Bonzini and L. Pozzi. Recurrence-aware instruction set selection for extensible embedded processors. *IEEE Trans. Very Large Scale Integr. Syst.*, 16, 2008.

[9] U. D. Bordoloi, H. P. Huynh, S. Chakraborty, and T. Mitra. Evaluating design trade-offs in customizable processors. In *DAC*, 2009.

[10] D. Borodin, B. H. Juurlink, S. Hamdioui, and S. Vassiliadis. Instruction-level fault tolerance configurability. *Journal of Signal Processing Systems*, 57(1), 2009.

[11] N. Cheung, S. Parameswaran, and J. Henkel. INSIDE: INstruction Selection/Identification & Design Exploration for extensible processors. In *ICCAD*, 2002.

[12] N. Clark, H. Zhong, and S. Mahlke. Processor acceleration through automated instruction set customization. In *MICRO*, 2003.

[13] P. Faraboschi et al. Lx: A technology platform for customizable VLIW embedded processing. In *ISCA*, 2000.

[14] R. Garg, N. Jayakumar, S. P. Khatri, and G. Choi. A design approach for radiation-hard digital electronics. In *DAC*, 2006.

[15] R. E. Gonzalez. Xtensa: A configurable and extensible processor. *Micro*, 20(2), 2000.

[16] M. R. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE Annual Workshop on Workload Characterization*, 2001.

[17] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization*, 2004.

[18] J. Lee, K. Choi, and N. Dutt. Efficient instruction encoding for automatic instruction set design of configurable ASIPs. In *ICCAD*, 2002.

[19] A. Lesea, S. Drimer, J.J. Fabula, C. Carmichael, and P. Alfke. The Rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs. *IEEE Trans. Device and Materials Reliability*, 5(3):317–328, 2005.

[20] T. Li, Z. Sun, W. Jigang, and X. Lu. Fast enumeration of maximal valid subgraphs for custom-instruction identification. In *CASES*, 2009.

[21] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO*, 2003.

[22] I. Polian, J. P. Hayes, S. M. Reddy, and B. Becker. Modeling and mitigating transient errors in logic circuits. *IEEE Trans. on Dependable and Secure Computing*, 2010.

[23] L. Pozzi, K. Atasu, and P. Ienne. Exact and approximate algorithms for the extension of embedded processor instruction sets. *IEEE TCAD*, 25(7), July 2006.

[24] R. R. Rao, V. Joshi, D. Blaauw, and D. Sylvester. Circuit optimization techniques to mitigate the effects of soft errors in combinational logic. *ACM Trans. Des. Autom. Electron. Syst.*, 15, 2009.

[25] R.C.Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

[26] G. A. Reis, J. Chang, and D. I. August. Automatic instruction-level software-only recovery. *IEEE Micro*, 27(1):36–47, 2007.

[27] Relsat 2.1,. www.bayardo.org/resources.html.

[28] S. Z. Shazli and M. B. Tahoori. Obtaining microprocessor vulnerability factor using formal methods. In *Int'l Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008.

[29] W. Sheng, L. Xiao, and Z. Mao. Soft error optimization of standard cell circuits based on gate sizing and multi-objective genetic algorithm. In *DAC*, 2009.

[30] A. K. Verma, P. Brisk, and P. Ienne. Fast, nearly optimal ISE identification with I/O serialization through maximal clique enumeration. *IEEE Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 29:341–354, 2010.