Reducing Soft Errors through Operand Width Aware Policies

Oguz Ergin, Osman S. Unsal, *Member*, *IEEE*, Xavier Vera, and Antonio González

Abstract—Soft errors are an important challenge in contemporary microprocessors. Particle hits on the components of a processor are expected to create an increasing number of transient errors with each new microprocessor generation. In this paper, we propose simple mechanisms that effectively reduce the vulnerability to soft errors in a processor. Our designs are generally motivated by the fact that many of the produced and consumed values in the processors are narrow and their upper order bits are meaningless. Soft errors caused by any particle strike to these higher order bits can be avoided by simply identifying these narrow values. Alternatively, soft errors can be detected or corrected on the narrow values by replicating the vulnerable portion of the value inside the storage space provided for the upper order bits of these operands. As a faster but less fault tolerant alternative to ECC and parity, we offer a variety of schemes that make use of narrow values and analyze their efficiency in reducing soft error vulnerability of different data-holding components of a processor. On average, techniques that make use of the narrowness of the values can provide 49 percent error detection, 45 percent error correction, or 27 percent error avoidance coverage for single bit upsets in the first level data cache across all Spec2K. In other structures such as the immediate field of the issue queue, an average error detection rate of 64 percent is achieved.

Index Terms—Memory structures-reliability, testing and fault tolerance, soft errors, narrow values.

1 Introduction

Alpha particles released by radioactive impurities and neutrons coming from outer space are known to cause transient errors in contemporary microprocessors [5], [33]. "Single bit upsets" may arise when these particles hit intermediate capacitive nodes of processor storage components such as SRAM bitcells and latches. Since these transient errors occur due to an incorrect charge or discharge of an intermediate capacitive node, they do not cause permanent failure in the hardware and, hence, are termed "soft errors" in the literature.

There are four major factors that affect soft error rate (SER) of a processor: amount of charge that hits the storage element, capacitance value of the node that is hit, die area, and supply voltage. While the charge carried by a high-energy particle tends to be constant, node capacitance and supply voltage decrease for each new manufacturing process, making the circuit components of a processor more prone to soft errors [30]. On the other hand, for a given microprocessor design, scaling manufacturing technology reduces the area of the processor and, consequently, reduces the probability of a high-energy particle to hit the die. As a matter of fact, SER of a processor tends to be constant with technology and voltage scaling if the design of a processor does not change. However, as new microprocessor designs make use of additional/bigger hardware structures and are likely to occupy more area [32], SER of a microprocessor is expected to increase and become a major challenge in the microprocessor design.

1.1 Definitions and Motivations

Architectural vulnerability factor (AVF) of a processor component is defined as the probability that a particle strike at any place in the component will result in an erroneous behavior in the executed program. For example, if a particle strike to an unallocated entry in a component does not cause an error in the processor, then the AVF of an unallocated entry is 0 percent.

Mukherjee et al. [30] defined architecturally correct execution (ACE) bits as the bits that are vulnerable to particle strikes. A particle hit on these ACE bits results in a visible error in the final program outcome. Similarly, a bit that does not hold any required information for ACE and hence is not vulnerable to soft errors is defined as an unACE bit. AVF of a component is equal to the percentage of ACE bits inside the corresponding component. Hence, AVF of a component can be calculated with the following equation:

$$AVF = \frac{Average\ number\ of\ ACE\ bits\ resident\ in\ a\ hardware\ structure\ in\ a\ cycle}{Total\ number\ of\ bits\ in\ the\ hardware\ structure}.$$

Our design is generally motivated by the fact that many of the produced and consumed values in a processor are narrow where a narrow value is defined as a value that holds consecutive zeros or ones in its upper order bits [7], [12], [26], [27]. These values can be represented in a simple compressed manner by just ignoring their upper order bits. Soft errors caused by any particle strike to these unnecessarily stored higher order bits can be avoided by simply identifying these narrow operands and converting their upper order bits from ACE to unACE. In this paper, we

- *O. Ergin is with the Department of Computer Engineering, TOBB University of Economics and Technology, Sogutozu Cad. No. 43 Sogutozu, Ankara 06560, Turkey. E-mail: oergin@etu.edu.tr.*
- *O.S. Unsal is with the Barcelona Supercomputing Center, Edificio Nexus II-Jordi Girona 29, 08034 Barcelona, Spain. E-mail: osman.unsal@bsc.es.*
- *X. Vera and A. González are with the Intel Barcelona Research Center, Edificio Nexus II-Jordi Girona 29, 08034 Barcelona, Spain. E-mail: {xavier.vera, antonio.gonzalez}@intel.com.*

Manuscript received 24 Apr. 2007; revised 16 Oct. 2007; accepted 11 Dec. 2007; published online 12 Mar. 2008.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2007-04-0056. Digital Object Identifier no. 10.1109/TDSC.2008.18.

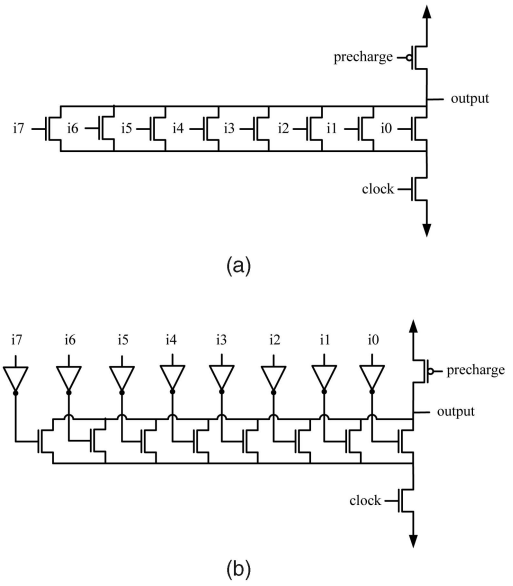


Fig. 1. Consecutive (a) zero and (b) one detection circuits.

exploit narrow values to establish a fault tolerance scheme that has lower latency and lower error coverage when compared to well-known Error Correcting Code (ECC) and parity schemes.

1.2 Outline

This paper proposes several techniques that leverage narrow operands to improve soft error tolerance of processors' data-holding components. With our first technique, by identifying narrow operands and zero partitions (consecutive zeros), some portion of the narrow data becomes invulnerable to particle strikes and the total number of soft errors that affect the final program output is reduced. As a second scheme, we improve our first technique to detect and correct the particle hits that occur on the unprotected part of the narrow value by replicating the significant part of the narrow value into the storage space devoted to store the upper order bits. We further improve our technique by using storage space allocated for data partitions that hold zero values as a repository for replicated data, and later, we use these replicated copies of the data to detect particle hits on the stored value.

The rest of this paper is organized as follows: We present our techniques for using narrow-width operands for improving soft error tolerance in Section 2. A soft error avoidance and correction scheme for narrow operands based on value replication is proposed in Section 3. Our simulation methodology is described in Section 4 followed by our experimental results. We review previous work in Section 5 and summarize some concluding remarks in Section 6.

2 Identifying Narrow Values for Improving Soft Error Tolerance

Many researchers observed that a large percentage of the generated and consumed values in a processor are narrow. The narrowness of the values was previously used for performance improvement [7], [12], [18], [27], energy efficiency [8], [11], [20], [45], and complexity reduction [40] in superscalar microprocessors. In this paper, we propose a new way of exploiting narrow values by identifying them throughout the processor for reducing soft error vulnerability and replicating them inside the conventional storage space for error detection and correction.

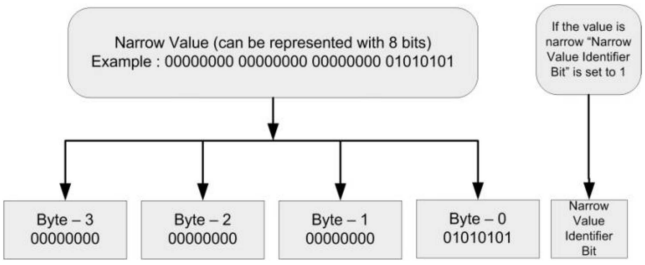


Fig. 2. Example of narrow value identification.

In order to make use of the width variations in produced and consumed values and identify a stored narrow operand, our proposed architecture uses an additional bit called *Narrow Value Identifier Bit (NVIB)* for each data storage entry in the value holding components. This bit is set whenever a narrow value is written into the storage space. By using this bit, it is possible to identify the unneeded portion of the stored value and these bits, which are identified as "unneeded," are converted to unACE bits. Consequently, correctness of the stored narrow value is not endangered by a bit flip caused by a particle strike if this particle strike occurs at the upper order bits. When a value is read out from the storage element, if the narrow value indicator bit is set, upper order bits are not read and the stored narrow value is sign extended to data path width before it is ready to be used. This sign extension can be accomplished by using a simple multiplexer.

Narrow values are identified by leading zero (or leading one) detectors just before writing the value to the specific data component. Fig. 1 shows the circuit diagrams for 8-bit leading zero and one detectors, respectively, which employ dynamic logic for faster operation and larger fan in.

Fig. 2 shows an example of the narrow value identification process where a narrow value is defined as a value that can be represented with only 8 bits. The *NVIB* is checked whenever a value is read from the data storage. If *NVIB* is set, Byte-0 is simply sign extended to 32 bits and any particle strikes to Bytes-1, -2, and -3 become ineffective.

There are obvious trade-offs in defining the length of a narrow operand. If a narrow operand is defined to have too few bits, then the percentage of narrow operands decreases, but the benefits of identifying the narrow values for vulnerability reduction increases since more bits are transformed into unACE bits. If a narrow operand is defined to include large number of bits, the percentage of narrow values increases but the number of protected bits in each narrow value decreases and hence the benefits also decrease. Therefore, there is an optimum point for defining the number of bits in a narrow operand where the percentage of narrow operands and the number of unACE bits are optimized for best vulnerability reduction. Choice

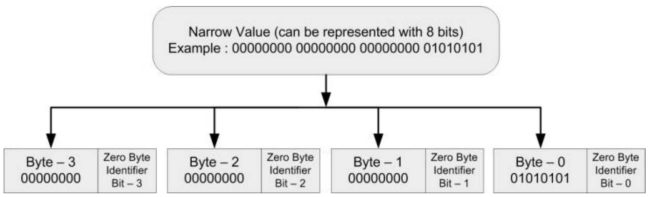


Fig. 3. Example of zero byte identification.

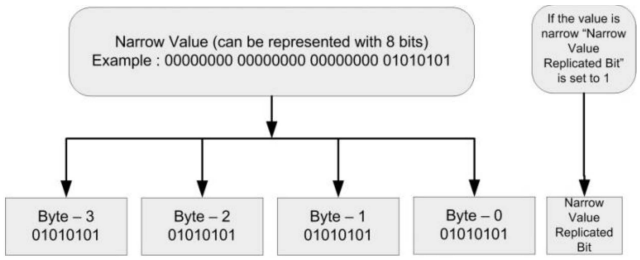


Fig. 4. Example of narrow value replication.

of number of bits to define the size of narrow values depends on the applications that are run.

It should be noted that the *NVIB* is itself unACE when it is indicating that a stored value is narrow. If this bit is flipped when it is indicating a narrow operand, the value is not endangered but the narrow value protection is nullified and the contents of the upper order bits become vulnerable to particle attacks. On the other hand, *NVIB* is an ACE bit when the storage space is holding a wide value since the contents of the upper order bits will be lost if a particle strike occurs on it. Therefore, we call *NVIB* a "half-ACE" bit, meaning that its vulnerability status depends on the contents of the value stored in the storage area.

Although *NVIB* is half-ACE, it is still partially vulnerable to particle strikes and, hence, increases the vulnerability of the structure it is protecting. Therefore, the vulnerability reduction achieved by adding these bits must justify the slight increase in soft error vulnerability.

2.1 Zero Partition Identification for Soft Error

A variation of narrow value identification can be used to increase the chances of reducing soft error vulnerability in a processor by identifying zero partitions instead of identifying the whole narrow values. Identifying zero bytes was first proposed by Villa et al. [45] for reducing data cache energy by avoiding the reading and writing of zero bytes. Instead of avoiding the read and write of zero bytes for energy efficiency, we propose identifying such zero partitions for soft error vulnerability reduction.

Fig. 3 shows an example of zero partition encoding process where a partition is defined to be a byte. By inserting 1 bit per byte, each all-zero-containing-byte can be identified and be immunized to particle strikes. When the zero byte identifier bit is found out to be set while reading the data, the value is not read and instead a zero byte is provided.

As it is the case with narrow value identification bits, zero partition identifier bits are also "half-ACE" since a particle strike on these bits does not jeopardize correct program execution when they indicate a zero byte. Therefore, they also increase the soft error vulnerability of the component where they are added if they are not protected.

3 Replicating Narrow Values for Soft Error Detection and Recovery

Even though narrow value identification decreases the vulnerability in the data-holding components of a processor, errors can still occur on the unprotected part of the narrow operand. Narrow value replication can be used for soft error detection and recovery since multiple copies of a narrow value can fit into the allocated storage space. In case of a particle hit on the entry, this particle hit can be detected by comparing stored copies with each other. Similarly, soft errors can be corrected by recovering the correct value from one of the uncorrupted copies without signaling an error or creating an exception if there are enough number of correct replicated copies.

3.1 Narrow Value Replication for Soft Error Detection and Correction

In the implementation of narrow value replication, our previously proposed *NVIB* is replaced with a *Narrow Value Replicated Bit (NVRB)*, which indicates that the stored value is narrow and the narrow value is replicated inside the storage space. Upon obtaining a value from the storage element, if *NVRB* of the value is set, replicated values are compared with each other for detecting or correcting a potential error. By using this bit, it is possible to detect multiple particle hits to the value or correct at least a single particle hit and recover from the error, provided that there are enough copies of the value inside the storage space.

In order to make the required comparisons by the narrow value replication scheme, a set of comparators is required to work in parallel. The number of comparators depends on the recovery scheme; however, if the full protection is needed for N copies of a value, $N \times (N-1)/2$ comparators are needed to cover all cases. For example, if there are four replicas of a value inside a storage space, $4 \times 3/2 = 6$ comparators are needed. These comparators can be designed using dynamic logic for faster operation, or dissipate-on-match comparators proposed in [30] can be used for power efficiency.

Fig. 4 shows an example of the narrow value replication process where a narrow value is defined as a value that can be represented with only 8 bits. If a value is identified as narrow, *NVRB* is set while the value is being replicated inside the storage element (four copies of the Byte-0 are written to 32-bit storage area). *NVRB* is checked whenever a value is read from the data storage. If this bit is set, Byte-0 is simply sign extended to 32 bits after comparing all of the replicated copies with each other and making sure that all copies indicate the same value. If some of the comparisons result in mismatch, simple voting is used to decide which value to use where the highest number of identical copies inside the storage space wins.

Note that although the replicated value is protected from particle attacks, *NVRB* is itself not protected and is an ACE bit. A particle strike on this bit will endanger the correctness of the stored value at all times. Therefore, unlike previously proposed *NVIB*, the vulnerability increase introduced to the

TABLE 1
Actions Corresponding to Specific Number of Particle Strikes

I	J	K	L	ACTION
0	0	0	E_L	Corrected
0	0	E_K	E_L	If (K ≠ L) → Corrected
				Else → Detected
0	E_J	E_K	E_L	If (J = K = L) or (2 of [J, K, L] are equal) → Miscorrected
				Else → Detected
E_I	E_J	E_K	E_L	If (I = J = K = L) or (3 of [I, J, K,L] are equal)
				→ Miscorrected
				Else → Detected

I, J, K, L are different versions of the value and E_I, E_J, E_K, E_L, are the number of bitflips that occur on these replicated versions (all E are greater than zero).

corresponding component by *NVRB* is not conditional. This bit can either be left unprotected to avoid increased complexity, at the expense of the soft error vulnerability increase, or it can be replicated like the narrow value. Later in Section 4.3, we show that replicating this bit is not a cost-effective solution. If the *NVRB* is not replicated, in some cases errors on this bit may still be recognized with additional hardware since the replicated copies will differ from each other significantly when the content of *NVRB* flips from 0 to 1. It may also be possible to detect an error if the indicator bit flips from 1 to 0 since replicated partitions will differ very little (if they differ at all) from each other.

Similar to narrow value identification for soft error vulnerability reduction, there are tradeoffs in narrow value replication since the number of copies replicated inside the storage space is bound by the definition of the size of the narrow operands. There is an optimum point for defining the number of bits in a narrow operand where the percentage of narrow operands and the number of replicated copies inside the storage elements are optimized for best level of error tolerance.

In a 32-bit storage space, if the narrow operand is defined as the values that can be represented with 16 bits, it is possible to fit two copies of the values inside the storage space. Assuming that there are no other means of error detection, having two copies of the same value is enough to detect a single bit upset but is not enough to recover from this error.

Similarly, it is possible to fit four 8-bit values inside 32-bit storage space for more protection from soft errors. Table 1 summarizes all possible cases for different number of bit flips on the same value, where I, J, K and L denote the different copies of the replicated value and $E_{I,J,K,L}$ shows the number of bit flips on the corresponding copy. For example, on the third row where I holds an uncorrupted copy and all the other copies (J, K, and L) are corrupted, the action depends on the contents of the corrupted copies. If at least two of the corrupted copies are identical, incorrect value will win the voting (two identical corrupted versus one different corrupted and one uncorrupted copy). As it can be seen from the table, our narrow value replication technique with a narrow value size of 8 (four copies of the value inside) can surely correct one particle hit on the storage and can at worst detect two particle hits if the dual bit flip cannot be corrected. Although a rare event, it is also possible to have a miscorrection if most copies are hit at the same bit position and simple voting favors the faulty copies. This situation is extremely improbable, and therefore, we do not provide any solutions in this paper for such occasions.

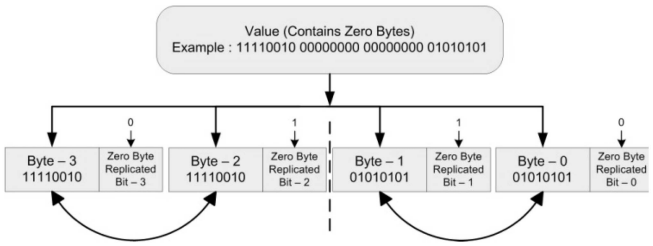


Fig. 5. Example of using zero bytes as replication repositories.

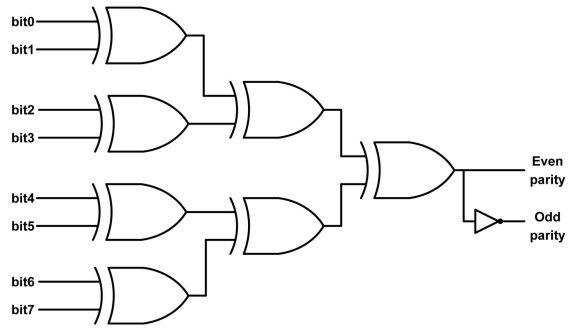
In order to implement our soft error recovery mechanism, a number of comparators are needed to compare all versions of values with each other in addition to the *NVRB* (and its replicated copies) per entry of the data storage element. In our example, where a narrow value is defined as 8 bits wide, there are four copies of the same value and six comparators are needed.

3.2 Replicating Data into Zero Partitions for Soft Error Detection and Correction

A variation of narrow value replication can be used for error detection by using the storage space allocated for zero partitions as a repository for replicating nonzero data. We propose to augment each byte inside data storage space to include a bit called "Zero-Byte, holding Replicated data" (*ZBR*). When a nonzero byte is replicated inside the storage space of a zero byte, this bit is set to indicate that the actual byte is zero and it now contains the value of another byte for error detection purposes. Similar to *NVRB*, *ZBR* is also an ACE bit at all times and the same tradeoffs exist in terms of soft error vulnerability reduction and invested hardware.

Different heuristics can be applied to leverage zero portions of the values for soft error detection (and possibly recovery) in order to simplify the replication and error detection logic. The number of copies generated for a nonzero portion determines the level of protection, as it is the case for narrow value replication. As the number of copies generated and the number of different places a data partition can be replicated increases, the complexity of the design increases together with the level of protection obtained.

As an example to demonstrate the effectiveness of our technique, we present a simple heuristic to detect soft errors by taking a single copy of each byte and copying each byte only in one (or optionally two) different place(s). Fig. 5 shows an example of the heuristic where the 32-bit value is divided into two parts and within each part each byte is allowed to be replicated to the other byte. Replicating bytes only within 16-bit partitions simplifies the design since, when the *ZBR* bit of a byte is set, it is known that the replicated byte is the adjacent byte within the 16-bit grouping. In Fig. 5, the value to be stored contains 2 zero bytes (Byte-1 and Byte-2). Using our heuristic, Byte-0 is replicated to Byte-1 and Byte-3 is replicated to Byte-2. When the value is accessed from the storage space, the error detection logic will detect that Byte-1 and Byte-2 holds replicated data and will compare Byte-3 with Byte-2 and



INPUT	EVEN PARITY	ODD PARITY
01010101	0	1
11010101	1	0

$$Even\ parity = bit0 \oplus bit1 \oplus bit2 \oplus bit3 \oplus bit4 \oplus bit5 \oplus bit6 \oplus bit7$$

Fig. 6. Eight-bit parity generator circuit and parity generation example.

Byte-1 with Byte-0. If there is a mismatch, the processor will signal the detected error. If an error is not detected, the contents of Byte-2 and Byte-1 will be discarded and replaced with zero bytes.

An optimization is possible to extend the protection to cover the cases when Byte-3 and Byte-2 are zero bytes and Byte-1 and Byte-0 hold valid data or vice versa (this case is similar to 16-bit narrow value replication where two uppermost bytes are zero). Our heuristic can be extended to detect this case, and lower order 2 bytes can be copied to the upper order 2 bytes for error detection purposes. When the error detection logic detects that both *ZBR* bits within a 16-bit group are set, it understands that the stored bytes are copied from the other 16-bit group and compares the corresponding bytes with each other for error detection (Byte-0 with Byte-2 and Byte-1 with Byte-3).

Although the optimized heuristic of replicating data into zero bytes has larger soft error detection coverage, a better alternative would be to combine zero byte identification for vulnerability reduction and data replication for detection. When both of the bytes in a byte pair are zero, both *ZBR* bits can be set and their zero byte status can be identified without any data replication inside them. Upon reading the data, whenever error detection logic detects that both bits are set, it just replaces any data stored inside the bytes with zeros. This way, zero bytes are protected from particle attacks while nonzero bytes can be replicated into a zero byte for error detection if the byte next to them is zero.

3.3 Parity Protection and Error Correcting Codes (ECCs)

ECCs and parity are widely known and used techniques in current processors to protect memory structures against soft errors. While parity can only detect single bit errors, ECC can be used to correct single bit errors and detect double bit errors [also known as Single Error Correction, Double Error Detection (SECDED)]. Parity protection allows the detection of odd number of errors by counting the number of 1s inside a value and storing this information in an additional bit. Counting the number of 1s inside a value is accomplished by XORing each and every bit together, as shown in Fig. 6. Since XOR gates are slow compared to other basic gates and using dynamic logic is not helpful due to the nature of exclusive-OR operation that requires independent bit-pair operation, latency of parity generator circuits is typically high compared to simple logic circuits such as NAND and NOR. There have been some efforts to speed up parity generators that are used for large operands by using four-input XOR gates [23]. However, these circuits still need multiple levels of logic.

Detecting an error by using the parity bit also requires the same logic. Error detector logic XORs all of the bits including the parity bit since the parity bit indicates the number of 1s inside the value including itself. If the outcome of this XOR operation differs from the value of the parity bit, an error is signaled.

ECCs are used when the error tolerance requirements mandate correcting errors on the stored data or detecting more number of errors than that can be caught by using a parity bit. The most commonly used code is the Hamming code. Hamming code accomplishes error detection and correction by using multiple parity bits for each stored data. The value of each parity bit is calculated by using a different group of bits inside the value. Therefore, to implement the hamming code, multiple parity generator circuits have to be used in parallel. For an 8-bit value $(\mathbf{v_1v_2v_3v_4v_5v_6v_7v_8})$, four parity bits $(\mathbf{p_1p_2p_3p_4})$ are placed at the digits indicated by the power of two. An example is shown below:

Input value $(v_1v_2v_3v_4v_5v_6v_7v_8)$	Value with ECC $(p_1p_2v_1p_3v_2v_3v_4p_4v_5v_6v_7v_8)$
01010101	**000**1101**0**0101

When the data is read from the storage space, all of the parity bits are recalculated and verified. If an error occurs on the stored data, some of the parity bits are calculated to different values and the parity check fails. The location of the error can be found by summing the location of the parity bits that are found to be 1. Afterward, correction logic flips the bit indicated by the outcome of the summation.

Both ECC and parity require encoding and decoding logics that consist of multiple levels of exclusive-OR gates. Since each and every value written into a protected storage space has to be encoded, the latency of this process is unavoidable. Therefore, using ECC and parity bit is not desirable for structures that are on the critical path due to the delay overhead introduced by these techniques. Generally, ECC is only used on memory and not on data path components (such as the register files) because of the large encoding/decoding latency overhead. Likewise, power dissipation is another concern for highly utilized structures of a processor since more data will be encoded and decoded.

In memory structures, parity is generally implemented as a single bit per 32-bit word or 1 bit per byte, while ECC has a four-bit overhead per byte (50 percent more area) as stated by Phelan [33]. ECC can also be used for the whole 32-bit word with a 7-bit overhead. As a general rule, the more the coverage, the higher the delay and complexity of the encoding and decoding process. For both ECC and parity schemes, every added information (parity) bit introduces an encoder and a decoder circuit like the one shown in Fig. 6 (total two). Therefore, to protect 1 byte, ECC needs eight parity generator circuits, whereas parity scheme uses two. On the other hand, narrow value identification logic is only used once for the whole generated result.

TABLE 2
Comparison of Using Narrow Values with ECC and Parity

	PARITY	ECC	EXPLOITING NARROW VALUES
Error Coverage	Can detect odd number of errors	Corrects 1 and detects 2 errors	May correct and detect multiple bit errors (or avoid the error by overwriting it through sign extension)
Values Protected	All values	All values	Only narrow values
Encoding / Decoding Complexity	Multiple levels of XOR gates	Multiple levels of XOR gates + correction logic	Zero/one detectors, sign extenders, comparators (all can be dynamic logic)
Information Overhead	1 bit per byte or 1 bit per stored word	4 bits for a byte 7 bits for a 32 bit word 8 bits for a 64 bit word	1 bit for each stored value

3.4 Exploiting Narrow Values versus ECC and Parity

ECC and parity are effective methods to detect or correct soft errors in memory structures. However, they do not come for free; complex encoding and decoding circuits that consist of multiple levels of XOR gates have to be employed before the value is written to and after the value is read from the storage space. Therefore, they may not be suitable for structures whose access latency is on the critical path of a processor.

We propose the narrow value identification and replication as a flexible and simple alternative to error detection/correction schemes that use ECC and parity bit. With narrow value identification, it is possible to detect/correct multiple bit errors, but this coverage applies to only narrow values stored in the storage space. If a full coverage is needed, narrow value identification can be used in conjunction with the other error detection/correction schemes to improve error tolerance. Exploiting narrow values offer a wide variety of choices for error detection, correction, or avoidance at the expense of simple low latency encoding (leading zero or one detectors) and decoding (sign extension or comparison). Table 2 summarizes the differences of exploiting narrow values when compared with ECC and parity.

In order to compare the circuit level overhead of exploiting narrow values with that of ECC and parity, we measured the delays and energy dissipations of the parity generator and the narrow value identifier logics through SPICE simulations by using the 90-nm BSIM model 4.5.0 [8]. We used the pass transistor XOR gate design described in [50] to implement the parity generator logic (this XOR gate turned out to be the fastest one among other XOR gate designs in our simulations). We measured the worst case delay of 25-input narrow value detectors in Fig. 1, which are used to detect 8-bit narrow values in a 32-bit data path, as 187 ps. This delay consists of the precharge time (106 ps) and the evaluation time (81 ps) and occurs when only one pull-down path is enabled. The circuit dissipates 311 fJ whenever it does not indicate a narrow value and its energy dissipation is negligible when a narrow value is identified (since the precharged output node is not discharged on narrow value identification). If the narrow values are defined to contain fewer bits, the energy dissipation and the delay of the identifier circuit increase. The delay of a 32-bit identifier circuit is 218 ps with 367-fJ energy dissipation.

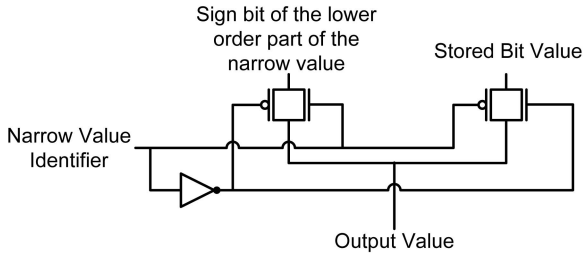


Fig. 7. One-bit multiplexer circuit that is used for sign extension.

Parity generator logic is composed of multiple levels of XOR gates, which make it slow. Our circuit simulations show that the delay of an 8-bit parity generator (shown in Fig. 6) is 698 ps. Unlike narrow value identification, parity generator circuits are both used for encoding and decoding. Therefore, this delay penalty is paid when reading and writing a value, whereas for narrow value identification, delay penalty is paid only when the result is generated. Energy dissipation of a parity generator circuit amounts to 136 fJ. This is lower than that of the narrow value identifier circuit since dynamic logic is used for identifying narrow values. However, it should be noted that parity generator circuit dissipates energy at all times, but the narrow value identifier only dissipates energy when the incoming value is not narrow. Moreover, multiple parity generator circuits have to be employed for generating multiple parity bits for ECC, which makes encoding and decoding more energy consuming.

Narrow value replication mandates the use of comparators to check if the copies of the same value are the same and free of errors. For this purpose, traditional pull-down comparators can be used, as described in [30]. Energy dissipation of an 8-bit traditional comparator circuit at 90-nm technology is 72 fJ and its delay is 84 ps. For sign extension and routing of the correct value to the read busses, transmission gates are used, as shown in Fig. 7. Delays of these gates are generally low; our simulations show that the delay of this circuit is around 30 ps. When compared to encoding and decoding logic of ECC and parity, exploiting narrow values has less circuit level overhead, both in terms of delay and energy dissipation.

4 RESULTS AND DISCUSSIONS

In this section, we present the experimental results to demonstrate the effectiveness of our proposed techniques. Several elements in a processor can benefit from identifying narrow operands for soft error vulnerability reduction. In general, any component that is used to store a data value is eligible to use this technique. We evaluate the benefits of our techniques on writeback latches, integer register file, the immediate field in the issue queue, and the data cache.

4.1 Simulation Methodology

Results provided in this section were collected from an in-house IA-32 trace-driven simulator simulating a processor

TABLE 3
Configuration of the Simulated Architecture

PARAMETER	CONFIGURATION
Machine width	3-wide fetch, 6-wide issue, 3 wide commit
Window size	64 entry issue queue, 64 entry load/store queue, 128–entry ROB
Function Units and Latency (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 2 FP Add (2), 1FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
Trace Cache	32 Kuops, 4 way
L0 D–cache	16 KB, 2–way set–associative, 64 byte line, 1 cycle hit time
L1 Cache unified	2 MB , 8–way set–associative, 128 byte line, 12 cycles hit time
BTB	4K entry, 2–way set–associative
Branch Predictor	Combined with 1K entry Gshare, 8 bit global history, 4K entry bimodal, 1K entry selector
Memory	256 bit wide, 350 cycles first part, 1 cycles interpart
TLB	32 entry (I) – 2-way set-associative, 128 entry (D) – 16-way set associative, 12 cycles miss latency

similar to Intel Pentium 4 [19]. The processor parameters used in our experiments are representative of a recent microarchitecture and are summarized in Table 3. Our workload consists of 26 traces generated from Spec2K integer and floating point benchmarks. To skip the initialization section, we divide each benchmark into 10 equal slices and start executing from the fourth slice. Each trace is composed of 100 million instructions.

In order to report the effectiveness of our narrow value and zero partition identification techniques, vulnerability reduction is used as a metric. Since the absolute soft error vulnerability of a processor component depends on its size and the characteristics of the running application, absolute vulnerability numbers may change with different processor configurations and may be misleading. However, vulnerability reduction percentage of a component is independent from the size of the corresponding component and, hence, is used here as a metric to report the benefits of our techniques along with a soft gain factor $\breve{g}$.

4.2 A "Soft Gain" Metric: $\breve{g}$

Benefits achieved by using narrow and zero value identification have a tradeoff between the area overhead of the identification bits and the reduction in vulnerability of the corresponding component. While too much area overhead is not desirable due to increased complexity, power dissipation, and latency of the structure, large amounts of vulnerability decrease can be achieved by investing minimum amount of hardware resources. In order to evaluate the benefits of the proposed soft error vulnerability reduction schemes, we introduce a "soft gain" metric[1] $\breve{g}$ that takes these concerns into account. We define $\breve{g}$ as

$$\breve{g} = \frac{Decrease\ in\ soft\ error\ vulnerability}{Increase\ in\ storage\ area}.$$

1. We chose the Turkish letter $\breve{g}$ (called "soft g") to indicate soft gain metric.

If the decrease in vulnerability is high enough to justify the invested area (which also indirectly indicates the delay, power, and complexity overhead), gain factor $\breve{g}$ is high and the design is justified. Invested area can be roughly estimated by the following formula:

$$Increase\ in\ area = \frac{Number\ of\ added\ bits}{Total\ number\ of\ bits\ in\ the\ component}.$$

By definition, the smallest value $\breve{g}$ can get is zero, which happens when the vulnerability of the structure does not decrease at all, assuming that there is a technique applied for vulnerability reduction [$\breve{g}$ can be negative only in two circumstances; either a decrease in area makes the component less vulnerable (good case but not realistic) or an increase in area results in an increase in vulnerability (normal case but not a scheme for error tolerance)]. Although there is no upper bound on $\breve{g}$, a practical high value would be 100 (when 100 percent vulnerability reduction is achieved with 1 percent area overhead). For the evaluation of our schemes in the results section, we will consult this metric to compare the different vulnerability reductions achieved with different levels of hardware overhead.

4.3 Evaluation of Narrow Value Identification for Reducing Soft Error Vulnerability

We now present the results for different storage components of a processor. We also discuss the different tradeoffs between soft error reduction and area overhead.

4.3.1 Integer Register File

The integer register file is the major component that holds temporary values inside the CPU and all of its bits are eligible for narrow value identification. For this study, we did not consider the floating point register file since its structure is more complex and stored values are less likely to be narrow. Integer registers hold 32-bit values in IA-32 and can be in four states: free, allocated but not written back, written back but last use has not yet occurred, and dead (last use occurred and the value is kept to reconstruct precise state). Nonallocated registers and registers that are allocated but do not hold any valid data are not vulnerable to particle strikes. Register values are vulnerable to soft errors between the time the result is written back and the time last use of the result occurs. The vulnerability of the register file is reduced as a whole by identifying narrow operands since the number of vulnerable bits between writeback and last use of the value is reduced.

Fig. 8 shows the soft error vulnerability reduction in the integer register file by identifying narrow operands with and without replicating the *NVIBs*. Numbers are presented for different sizes of narrow values across all Spec2K integer and floating point benchmarks. For each defined length of the narrow operand, the leftmost three bars show the integer, floating point, and overall averages for vulnerability reduction with *NVIBs* not replicated. Similarly, the rightmost three bars show the vulnerability reduction with replicated *NVIBs* for integer, floating point, and all benchmarks, respectively, on average. As the figure reveals, there is an optimum point for the defined length narrow operands; it is 12 bits for integer benchmarks and 10 bits

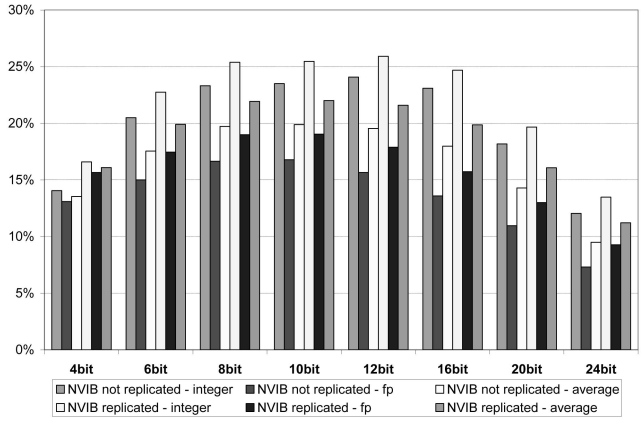


Fig. 8. Vulnerability reduction in the integer register file.

for floating point benchmarks. On average, defining narrow operands as "the operands that can be represented with 10 bits" provides the highest soft error vulnerability reduction in the integer register file.

In order to evaluate the best configuration based on soft error reduction and area overhead, we used the $\breve{g}$ metric introduced in Section 4.2. Table 4 presents the results of our studies. If the *NVIBs* are protected through replication (two additional copies of the *NVIB* are stored for correcting single bit upsets), soft error vulnerability reduction numbers reported in the leftmost three bars for each value width in Fig. 8 are increased by around 2 percent on average. However, the area overhead is tripled (from 1 bit to 3 bits) and $\breve{g}$ decreases significantly, as reported in Table 4 (from 6.36 to 2.35 for 10-bit narrow values). This drop in $\breve{g}$ is expected since we are able to achieve a soft error vulnerability reduction of more than 19.9 percent with the use of a single identification bit while we can improve this reduction by only 2 percent with an investment of two more bits. As $\breve{g}$ drops significantly, protecting *NVIBs* through replication for further improving the soft error vulnerability of the integer register file is not justified. Therefore, defining a narrow operand as 10 bits wide without replicating the identification bits is the best choice.

4.3.2 Immediate Field of the Issue Queue

Immediate field inside the issue queue is 32 bits wide and constitutes around 30 percent of the bits stored inside the issue queue (for a queue that does not hold data values). For an immediate field to be vulnerable to soft errors, the issue queue entry has to be occupied by an instruction that has an immediate literal and this instruction should not be flushed from the pipeline due to branch mispredictions or exceptions.

TABLE 5
Average $\breve{g}$ for Narrow Value Identification in the Immediate Field of the Issue Queue across All Spec2K

	4 BIT	6 BIT	8 BIT	10 BIT	12 BIT	16 BIT	20 BIT	24 BIT
NVIB not replicated	9.25	12.21	13.92	13.81	12.72	10.49	8.06	5.58
NVIB replicated	3.72	4.61	5.11	5.07	4.72	4.02	3.25	2.47

Table 5 shows the change of $\breve{g}$ in an issue queue with 64 entries for various value width definitions. Unlike the integer register file, optimum length of the narrow value definition occurs at 8 bits on the average. Overall vulnerability reduction in the immediate field of the issue queue is around 43.5 percent for 8 bits.

4.3.3 Data-Holding Latches (Writeback Latches)

Narrow value identification can also be used on latches that hold data values throughout the pipeline. Latches at the output of the function units are a good example since the execution stage is the place of actual value creation in the processor. Although all of the results generated by the function units are written to the register file, soft error vulnerability factors of these latches and the corresponding vulnerability reduction due to narrow value identification vary from integer register file's numbers. This is mostly because of the transient values that are consumed only through bypass network and are not read from the register file at all. In addition, the total vulnerable time a value spends in register file is variable since it depends on the time of the consumer's issue, whereas a stored value's lifetime inside a latch usually does not exceed a single cycle.

Table 6 shows the $\breve{g}$ values achieved in the writeback latches by identifying narrow operands. Results are similar to the results from the integer register file and immediate field in the issue queue; optimum point for the length of narrow values for the highest vulnerability reduction is achieved at 8 bits for integer benchmarks and 10 bits for floating point benchmarks. On the average, the highest achieved vulnerability reduction is 24.4 percent and it is achieved with a narrow value length of 10 bits.

4.3.4 Data Cache

The largest single component that holds data values in a processor is the first-level (level-0) data cache. The data part of the data cache contributes to more than 90 percent of its

TABLE 4
Average $\breve{g}$ for Narrow Value Identification in the Integer Register File across All Spec2K

	4 BIT	6 BIT	8 BIT	10 BIT	12 BIT	16 BIT	20 BIT	24 BIT
NVIB not replicated	4.33	5.61	6.31	6.36	6.25	5.75	4.57	3.04
NVIB replicated	1.72	2.12	2.34	2.35	2.30	2.12	1.71	1.20

TABLE 6
Average $\breve{g}$ for Narrow Value Identification in Writeback Latches across All Spec2K

	4 BIT	6 BIT	8 BIT	10 BIT	12 BIT	16 BIT	20 BIT	24 BIT
NVIB not replicated	5.87	6.95	7.75	7.81	7.62	6.84	5.57	4.00
NVIB replicated	2.69	3.02	3.25	3.26	3.19	2.94	2.53	2.03

TABLE 7
Average $\breve{g}$ for Narrow Value Identification in Level-0 Data Cache across All Spec2K

	4 BIT	6 BIT	8 BIT	10 BIT	12 BIT	16 BIT	20 BIT	24 BIT
NVIB not replicated	9.26	9.68	10.36	10.03	9.41	8.17	6.66	4.78
NVIB replicated	3.12	3.25	3.47	3.36	3.16	2.74	2.24	1.61

soft error vulnerability [2]. The data cache used for this experiment is a two-way set associative cache with a size of 16 Kbits and a line size of 64 bytes. Soft error vulnerability of the data cache depends on the utilization of each word contained within a line. Each word inside a line is vulnerable to particle strikes between the time data is inserted to the data cache and the time the data was last accessed. Table 7 shows the $\breve{g}$ values achieved by using narrow value identification. Results from our experiments show that, on the average, the highest achieved vulnerability reduction is 32.4 percent with 8-bit narrow values and this results in a $\breve{g}$ of 10.36.

4.4 Evaluation of Zero Partition Identification for Reducing Soft Error Vulnerability

Zero encoding can improve the vulnerability reduction achieved by narrow value identification if the consecutive zeros in a value reside in different places. In addition, if a generated value consists of all zeros, zero encoding can protect the whole value while the narrow identification cannot. However, the number of identification bits significantly increases especially for the smaller partition sizes, which results in a large area overhead. Therefore, unlike the area overhead of narrow value identification (which is a single bit) the area overhead of zero identification is the primary limitation as it increases with the decreasing number of bits inside the zero partitions.

Fig. 9 shows the vulnerability reduction achieved in the integer register file with the use of zero identification for Spec2K benchmarks. Zero identification results in a significant amount of vulnerability reduction (larger than narrow value identification), and this reduction becomes larger as the partition size decreases. However, this reduction comes at the expense of a large area overhead especially if the *ZIBs* are replicated. For example, for 2-bit zero identification, it is possible to achieve 56.2 percent vulnerability reduction at the expense of 150 percent area increase, which is not a very desirable solution when compared to 100 percent vulnerability reduction with 200 percent area increase by having three copies of each and every bit. Fig. 9 also shows that the vulnerability reduction saturates with smaller partition sizes as the vulnerability overhead of the added identification bits start to limit the benefits.

Results for other processor components are similar and also show that identifying zero partitions achieves high vulnerability reduction but is a less cost-effective solution compared to narrow value identification (i.e., vulnerability reduction achieved per inserted bit is lower). Table 8 summarizes the benefits for different components and

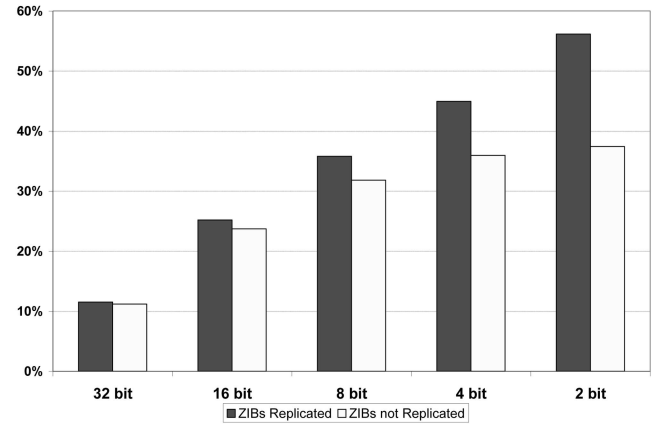


Fig. 9. Vulnerability reduction in the integer register file by using zero encoding.

shows the $\breve{g}$ for zero identification; as it is evident from this table and Table 5, although zero partition identification can achieve higher benefits (in terms of soft error vulnerability reduction), it is less cost effective when compared to narrow value identification. However, zero partition identification is more efficient when compared to having three copies of the same bit $(\breve{g} = 0.5)$ except for 2-bit partitioning with replicated *ZIBs*.

4.5 Evaluation of Narrow Value Replication for Detecting and Correcting Soft Errors

Narrow value identification converts the higher order unneeded bits of a narrow value to unACE bits while the actual ACE bits of the narrow value are still vulnerable to particle hits. It is possible to protect these ACE bits by replicating the ACE part into the storage allocated for the unACE bits. Narrow bit replication provides a larger extent of soft error tolerance at the expense of some hardware overhead.

Fig. 10 shows the percentage of single bit errors across all Spec2K benchmarks that can be avoided with different amount of hardware overhead. The leftmost bar for each benchmark indicates the percentage of avoided errors in the integer register file by simply identifying the narrow operands, where a narrow operand is defined to include 10 bits (which is shown to be the optimum length of a narrow value in Table 5). On the average across all benchmarks, 22 percent of the errors can be avoided by simply identifying the values that can be represented with 10 bits and later sign extending them when they are sought from the register file. The middle bar in Fig. 10 shows the percentage of corrected errors by replicating 10-bit narrow operands inside the 32-bit storage space. On the average, 32 percent of the single bit upsets can be corrected at the expense of three 10-bit comparators that compare the stored copies with each other and a 10-bit 3×1 multiplexer to select the correct value. Control logic that checks the outputs of the comparators can detect or correct a high number of particle strikes (theoretically up to 29 bit flips; eight on the first copy, seven on the second, third, and fourth copies, where each copy indicate a different value) unless the voting outcome at the outputs of the comparators indicates a miscorrection.

TABLE 8
Vulnerability Reduction (with Nonreplicated *ZIBs*) and $\breve{g}$ for Different Processor Components

	Decrease in Soft Error Vulnerability					$\breve{g}$				
	32 bit	16 bit	8 bit	4 bit	2 bit	32 bit	16 bit	8 bit	4 bit	2 bit
Integer Register File	11.2%	23.7%	31.8%	36.0%	37.4%	3.6	3.8	2.5	1.4	0.7
Write-back Latches	12.5%	24.6%	31.4%	35.2%	36.8%	4.0	3.9	2.5	1.4	0.7
Immediate Field of IQ	14.4%	32.4%	38.9%	45.7%	44.6%	4.6	5.2	3.1	1.8	0.9
Level-0 Data Cache	27.1%	38.0%	45.7%	47.1%	45.7%	8.7	6.1	3.7	1.9	0.9

Higher percentage of the soft errors can be avoided by using narrow value replication for only detection and relying on other means of error recovery for correct execution (such as branch misprediction recovery mechanism). In order to detect a single bit upset, having two copies of a narrow value is enough (which means 16-bit narrow value definition with 32-bit storage space). The rightmost bar in Fig. 10 indicates the percentage of errors that can be detected by identifying 16-bit narrow operands and replicating them inside the 32-bit storage space. On the average, 40 percent of the soft errors can be detected by using a single 16-bit comparator that signals the soft error in case the two copies do not match and implementing an error recovery mechanism inside the processor, which restores the precise state.

Table 9 shows the benefits of narrow value replication in data-holding components of the processor on average across all Spec2K benchmarks. As the table reveals, benefits of narrow value replication is consistent in different processor components.

4.6 Evaluation of Replicating Data into Zero Partitions

Data partitions that contain consecutive zeros can be used for data replication as a variation of the proposed narrow value replication scheme. Zero identification is more beneficial than narrow value identification if the consecutive zero blocks are distributed inside the stored data.

Table 10 shows the data distribution cases when a zero encoding is done at the byte level for the integer register file. Each zero in the first column indicates a zero byte and each 1 indicates a nonzero byte. Spec2K benchmark results show that zero bytes in the values are not very distributed and reside mostly on the most significant bits of the values.

Fig. 11 shows the percentage of reduction in the number of observed soft errors inside the integer register file by using three heuristics to replicate data inside the storage space allocated for zero bytes. The first bar in Fig. 11 shows the number of soft errors that can be detected using simple replication where a byte is replicated only to its neighbor if the neighboring byte is a zero byte. On the average across all benchmarks, around 21 percent of the errors can be detected by using simple replication.

Soft error detection coverage can be extended by relaxing the constraint of copying a byte only to its neighboring byte. When both bytes in a pair are zero bytes and the other pair is composed of nonzero bytes, extended replication allows nonzero bytes to be replicated inside zero bytes. This extension results in a more than 30 percent single bit upset detection rate on the average.

A better improvement over the simple replication scheme is to use the zero byte identification bits to reduce

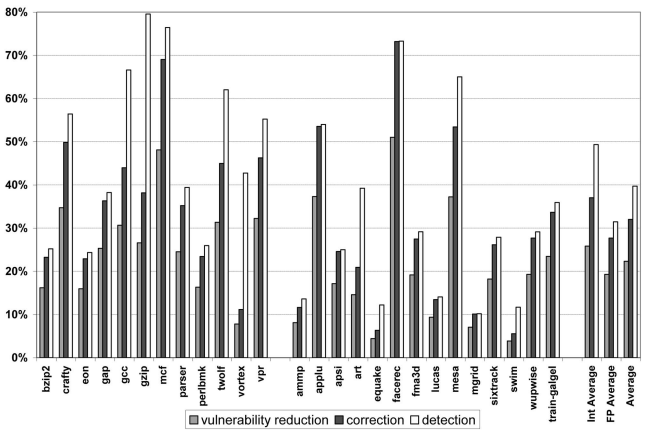


Fig. 10. Soft error reduction in the integer register file by using narrow value replication.

TABLE 9
Benefits of Narrow Value Replication in Various Data-Holding Components

	Vulnerability Reduction (10 bits)	Correction (10 bits)	Detection (16 bits)
Data Cache	31.3% ($\breve{g}$ = 10.03)	45.0% ($\breve{g}$ = 14.40)	49.6% ($\breve{g}$ = 15.87)
Immediate Field in the IQ	43.2% ($\breve{g}$ = 13.81)	61.9% ($\breve{g}$ = 19.81)	63.6% ($\breve{g}$ = 20.35)
Writeback Latches	24.2% ($\breve{g}$ = 7.81)	35.0% ($\breve{g}$ = 11.20)	41.5% ($\breve{g}$ = 13.28)
Integer Register File	19.9% ($\breve{g}$ = 6.36)	32.0% ($\breve{g}$ = 10.24)	39.7% ($\breve{g}$ = 12.70)

TABLE 10
Statistics for Different Zero Byte Holding Data Variations inside the Integer Register File

CASE	INT AVERAGE	FLOAT AVERAGE	AVERAGE
0_0_0_0	10.4%	12.5%	11.5%
0_0_0_1	20.7%	12.4%	16.3%
0_0_1_0	0.8%	0.4%	0.6%
0_0_1_1	13.9%	3.4%	8.2%
0_1_0_0	0.5%	0.1%	0.3%
0_1_0_1	0.1%	0.1%	0.1%
0_1_1_0	0.2%	0.5%	0.4%
0_1_1_1	2.6%	4.9%	3.8%
1_0_0_0	0.3%	0.1%	0.2%
1_0_0_1	0.0%	0.0%	0.0%
1_0_1_0	0.0%	0.1%	0.1%
1_0_1_1	0.0%	0.1%	0.1%
1_1_0_0	0.4%	3.0%	1.8%
1_1_0_1	0.2%	0.2%	0.2%
1_1_1_0	3.6%	34.0%	19.9%
1_1_1_1	46.3%	28.2%	36.6%

the soft error vulnerability by giving hints to the error detection logic and simply supplying zeros to the zero-identified bytes. As seen from the last bar in Fig. 11, an additional 25 percent of the errors can be avoided over simple replication, which is capable of detecting 21 percent of the errors. Together with zero identification, simple replication can tolerate 46 percent of the soft errors that occur on the integer register file.

Table 11 shows the average error coverage numbers for our error detection mechanisms in various data-holding components of a processor for all Spec2K benchmarks. Soft error detection/avoidance coverage is as high as 62 percent in the data cache for our hybrid scheme.

4.7 Selecting the Right Scheme

Until this point, we discussed many schemes that leverage narrow values, also providing coverage numbers along with the corresponding $\breve{g}$ values. Soft error requirements and hardware overhead budget of the target processor should be taken into account to make the best design choice. If a designer wants to improve the error tolerance by investing minimum hardware, just identifying the narrow values with a single bit may provide enough coverage. If the error avoidance and detection is the major concern, simple byte replication can be used along with zero byte identification to cover most partitions of the data values.

As seen in Tables 9 and 11, the choice of the correct scheme also depends on the target component. While simple byte replication with zero identification provides the highest error coverage in level-0 data cache, narrow value replication is a better choice for the immediate field (due to higher number of consecutive ones in the upper order bits). On the other hand, $\breve{g}$ provides information on the efficiency of the design choice for the designer that wants the most from invested hardware. For different components and workloads, different schemes may become more attractive choices.

5 Related Work

Several works have tried to identify the effects of soft errors on the processor pipeline both at architectural level [47] and at gate level [12]. Effects of power saving techniques on SERs are also studied in [24].

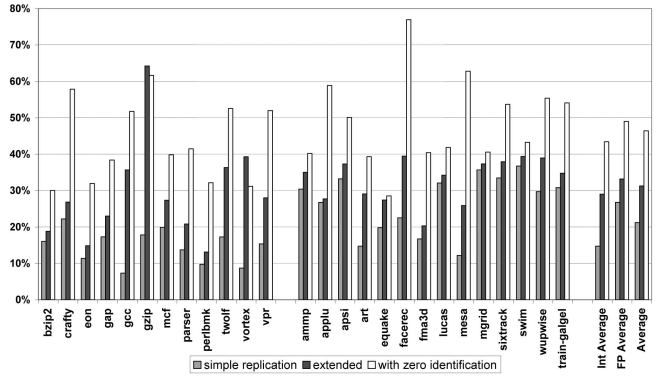


Fig. 11. Results for data replication inside zero bytes.

Reducing the architectural soft error vulnerability of the microprocessors is a developing area of research. Recently, soft error AVFs of some components in a processor were defined and evaluated by Mukherjee et al. [30]. Likewise, AVFs of address-based structures were computed in [6], and on-chip memory vulnerability was analyzed in [1] and [2]. A selective error recovery mechanism was proposed in [48], where a π bit is used to identify possible errors in each instruction and only for the instructions that are needed for ACE signals an error before they leave the pipeline.

Redundancy is a widely used technique to recover from transient faults in a processor [1]. Replicating register values into unused registers to recover from transient faults and soft errors was proposed in [27], where if ECC signals an error, the correct value is taken from the uncorrupted register that holds the copy. Replicating active words into passive words in caches was proposed in [49]. Recently, Reis et al. proposed using hardware-software hybrid schemes, which achieves fault tolerance by replicating instructions at compiler level and using hardware fault detectors that make use of this redundancy [38].

Multithreading is used for error detection and recovery [30], [36], [39], [44]. The general idea is to use multithreading to run two copies of the same thread and, after execution, check the outcome of the instructions to detect the errors and recover from them if it is possible. This approach causes some degradation in performance since each instruction is actually

TABLE 11
Benefits of Byte Replication in Various Data-Holding Components

	SIMPLE REPLICATION	EXTENDED REPLICATION	SIMPLE REPLICATION WITH ZERO IDENTIFICATION
Level-0 Data Cache	22.1% ($\breve{g}$ = 1.77)	30.6% ($\breve{g}$ = 2.45)	62.4% ($\breve{g}$ = 4.99)
Immediate Field in the IQ	18.8% ($\breve{g}$ = 1.51)	21.6% ($\breve{g}$ = 1.73)	53.2% ($\breve{g}$ = 4.26)
Writeback Latches	18.4% ($\breve{g}$ = 1.47)	26.6% ($\breve{g}$ = 2.13)	44.6% ($\breve{g}$ = 3.56)
Integer Register File	21.2% ($\breve{g}$ = 1.70)	31.3% ($\breve{g}$ = 2.50)	46.4% ($\breve{g}$ = 3.71)

run twice. Several works attempted to reduce this performance penalty by utilizing processor resources better or by using idle resources of the processors for error checking [14], [22], [35], [42]. Chip multiprocessors (CMPs) were also used for error detection and correction [16]. Symptom-based soft error detection was proposed in [46].

Narrow operands were leveraged in many different ways in the literature. Packing multiple narrow values into wide function units was proposed in [7] in order to improve performance. The same study was extended with the use of a width predictor in [30], and a similar idea was evaluated for a VLIW machine in [31]. In [26], Lipasti et al. proposed inserting a narrow value into the rename table entry and releasing the corresponding register early. Packing multiple narrow values inside a single register was proposed in [12] for high performance. A similar idea was implemented with a multibanked register file concept in [20]. A clustered architecture that makes use of narrow values to improve processor performance was proposed by Gonzalez et al. [18], [43].

The presence of narrow values inside the processors was widely used for achieving energy efficiency. In [45], Villa et al. have observed that many bytes in the cache contain only zero bits and proposed avoiding reading and writing of these bytes inside the data cache. Significance compression was proposed in [8] in order to encode significant zeros for power reduction in scalar pipelines. ISA extensions with operand-width-specifying opcodes were proposed in [10] for energy efficiency. Power consumption of a value predictor was reduced by exploiting narrow operands in [40]. Energy dissipation and delay of the register file was reduced with a content aware design in [17].

Recently, narrowness of the values was used to detect soft errors in the register files in [20]. Hu et al. propose duplicating narrow values inside the registers and detect soft errors on the wide values through simple parity check. A scheme using a combination of parity (for error detection) and value duplication (for error recovery) is used to mitigate soft errors on the narrow values (defined as 32 bits). Our work differs from this work in several aspects: First, our schemes do not require parity to function for the narrow values. We do not only focus on value duplication but explore other ideas such as triplicating the narrow value to recover from soft errors. We propose a metric to measure the additional hardware investment versus error coverage that enables us to explore the design space for using narrow values for soft error tolerance. We also cover the issue queue, data cache, and writeback latches, as well as the register file.

6 Concluding Remarks

Soft errors caused by particle hits are expected to be a major problem in the near future with increasing chip area and reduced feature sizes. In this paper, we have proposed microarchitectural techniques that make use of consecutive zeros and ones inside the stored values in order to improve soft error tolerance of the data storage components in a microprocessor. We have proposed soft error detection and recovery mechanisms, which protect the stored values by replicating parts of the operands into the already available storage space. None of our schemes result in IPC degradation.

Exploiting narrow values for error tolerance turned out to be a cost-effective solution although it provides protection only when the stored value is narrow. Our first technique of just identifying narrow values with a single bit results in a soft error vulnerability reduction of 22 percent in the integer register file, 43 percent in the immediate field of the issue queue, 24 percent in the writeback latches, and 32 percent in the data array of the data cache. In order to protect the vulnerable part of the narrow value, we proposed replicating these values into the provided storage space, which lead to up to 63 percent error detection or 61 percent error correction for single bit upsets that occur in the immediate field of the issue queue.

Our last technique that replicates nonzero data partitions inside the storage space of zero partitions can detect 22 percent and 31 percent of the single bit errors with our simple and extended schemes, respectively, in level-0 data cache. Our hybrid scheme that combines replication and zero identification together avoids 40 percent and detects 22 percent of the errors, which sums up to a total of 62 percent error reduction.

Simulation results show that there is no "ultimate scheme" that provides the best results in all components by making use of the narrow operands. The best solution for a given component depends on the needs of the processor designer, while the idea of exploiting narrow operands for soft error tolerance offers a range of solutions with tradeoffs between hardware overhead, type of error tolerance (avoidance, detection, or correction), and error coverage.

Acknowledgments

This work was done while Oguz Ergin and Osman Unsal were with the Intel Barcelona Research Center.

References

[1] G-H. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Balancing Performance and Reliability in the Cache Hierarchy," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS),* 2005.

[2] H. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Reducing Cache Susceptibility to Soft Errors," *IEEE Trans. Dependable and Secure Computing,* vol. 3, no. 4, Oct.-Dec. 2006.

[3] H. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Reliability Tradeoffs in Design of Cache Memories," *Proc. Workshop Architectural Reliability (WAR),* 2005.

[4] T.M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 1999.

[5] R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Design and Test of Computers,* 2005.

[6] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S.S. Mukherjee, and R. Rangan, "Computing Architectural Vulnerability Factors for Address-Based Structures," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2005.

[7] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* 1999.

[8] *BSIM 4.5.0 Manual,* http://www-device.eecs.berkeley.edu/~bsim3/bsim4.html, 2008.

[9] R. Canal, A. González, and J.E. Smith, "Very Low Power Pipelines using Significance Compression," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 2000.

[10] R. Canal, A. González, and J.E. Smith, "Software-Controlled Operand Gating," *Proc. Int'l Symp. Code Generation and Optimization (CGO),* 2004.
[11] Y. Cao and H. Yasuura, "A System-Level Energy Minimization Approach Using Datapath Width Optimization," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED),* 2001.
[12] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky, "Assessing SEU Vulnerability via Circuit-Level Timing Analysis," *Proc. Workshop Architectural Reliability (WAR),* 2005.
[13] O. Ergin, D. Balkan, D. Ponomarev, and K. Ghose, "Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 2004.
[14] O. Ergin, O. Unsal, X. Vera, and A. González, "Exploiting Narrow Values for Soft Error Tolerance," *IEEE Computer Architecture Letters (CAL '06),* vol. 5, 2006.
[15] M.A. Gomaa and T.N. Vijaykumar, "Opportunistic Transient-Fault Detection," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2005.
[16] M.A. Gomaa, C. Scarbrough, T.N. Vijaykumar, and I. Pomeranz, "Transient-Fault Recovery for Chip Multiprocessors," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2003.
[17] R. Gonzalez, A. Cristal, D. Ortega, A. Veidenbaum, and M. Valero, "A Content Aware Register File Organization," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2004.
[18] R. Gonzalez, A. Cristal, M. Pericas, M. Valero, and A. Veidenbaum, "An Asymmetric Clustered Processor Based on Value Content," *Proc. Ann. Int'l Conf. Supercomputing (ICS),* 2005.
[19] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium 4 Processor," *Intel Technology J.,* vol. Q1, 2001.
[20] J. Hu, S. Wang, and S.G. Ziavras, "In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2006.
[21] M. Kondo and H. Nakamura, "A Small, Fast and Low-Power Register File by Bit-Partitioning," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* 2005.
[22] S. Kumar and A. Aggarwal, "Optimum Resource Allocation for Concurrent Error Detection Techniques in High Performance Processors," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* 2006.
[23] S. Kumar, S.L. Kuo, and C.Y. Yip, *Fast Parity Generator Using Complement Pass-Transistor Logic,* US Patent 5608741.
[24] L. Li, V.S. Degalahal, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin, "Soft Error and Energy Consumption Interactions: A Data Cache Perspective," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED),* 2004.
[25] X. Li, S.V. Adve, P. Bose, and J.A. Rivers, "SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2005.
[26] M. Lipasti, B.R. Mestan, and E. Gunadi, "Physical Register Inlining," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2004.
[27] G. Loh, "Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 2002.
[28] G. Memik, M.T. Kandemir, and O. Ozturk, "Increasing Register File Immunity to Transient Errors," *Proc. Design, Automation and Test in Europe (DATE),* 2005.
[29] 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," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 2003.
[30] S.S. Mukherjee, M. Kontz, and S.K. Reinhardt, "Detailed Design and Evaluation of Redundant Multithreading Alternatives," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2002.
[31] T. Nakra, B.R. Childers, and M.L. Soffa, "Width Sensitive Scheduling for Resource Constrained VLIW Processors," *Proc. Workshop Feedback Directed and Dynamic Optimizations (FDDO),* 2001.
[32] D. Pham et al., "The Design and Implementation of a First-Generation Cell Processor," *Proc. Int'l Solid-State Circuits Conf. (ISSCC),* 2005.
[33] R. Phelan, *Addressing Soft Errors in ARM Core-Based Designs,* White Paper, ARM, Dec. 2003.
[34] D. Ponomarev, G. Kucuk, O. Ergin, and K. Ghose, "Energy Efficient Comparators for Superscalar Datapaths," *IEEE Trans. Computers,* vol. 53, no. 7, pp. 892-904, July 2004.
[35] M.K. Qureshi, O. Mutlu, and Y.N. Patt, "Microarchitecture-Based Introspection: A Technique for Transient Fault Tolerance in Microprocessors," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2005.
[36] S.K. Reinhardt and S.S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2000.
[37] G.A. Reis, J. Chang, N. Vachharajani, S.S. Mukherjee, R. Rangan, and D.I August, "Design and Evaluation of Hybrid Fault-Detection Systems," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2005.
[38] G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D.I August, "SWIFT: Software Implemented Fault Tolerance," *Proc. Int'l Symp. Code Generation and Optimization (CGO),* 2005.
[39] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors," *Proc. 29th IEEE Int'l Symp. Fault-Tolerant Computing (FTCS '99),* pp. 84-91, June 1999.
[40] T. Sato and I. Arita, "Table Size Reduction for Data Value Predictors by Exploiting Narrow Width Values," *Proc. Ann. Int'l Conf. Supercomputing (ICS),* 2000.
[41] Semiconductors Industry Assoc. (SIA), Int'l Technology Roadmap for Semiconductors 2005, http://www.itrs.net/Links/2005ITRS/Home2005.htm, 2008.
[42] K.C. Smolens, J. Kim, J.C. Hoe, and B. Falsafi, "Efficient Resource Sharing in Concurrent Error Detecting Superscalar Microarchitectures," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 2004.
[43] O. Unsal, O. Ergin, X. Vera, and A. Gonzalez, "Empowering a Helper Cluster through Data Width Aware Instruction Steering Policies," *Proc. 20th Int'l Parallel and Distributed Processing Symp. (IPDPS '06),* Apr. 2006.
[44] T.N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-Fault Recovery Using Simultaneous Multithreading," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2002.
[45] L. Villa, M. Zhang, and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction," *Proc. Ann. Int'l Symp. Microarchitecture (MICRO),* 2000.
[46] N. Wang and S.J. Patel, "ReStore: Symptom Based Soft Error Detection in Microprocessors," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2005.
[47] N. Wang, J. Quek, T.M. Rafacz, and S.J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2004.
[48] C. Weaver, J. Emer, S.S. Mukherjee, and S.K. Reinhardt, "Techniques to Reduce the Soft Errors Rate in a High-Performance Microprocessor," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2004.
[49] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam, "ICR: In-Cache Replication for Enhancing Data Cache Reliability," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2003.
[50] R. Zimmermann and W. Fichtner, "Low-Power Logic Styles: CMOS versus Pass-Transistor Logic," *IEEE J. Solid-State Circuits,* vol. 32, no. 7, 1997.

Oguz Ergin received the BS degree in electrical and electronics engineering from the Middle East Technical University, Ankara, Turkey, in 2000 and the MS and PhD degrees in computer science from the State University of New York, Binghamton, in 2003 and 2005, respectively. He was a senior research scientist at the Intel Barcelona Research Center for 15 months during 2004 and 2005. He is currently an assistant professor in the Department of Computer Engineering, TOBB University of Economics and Technology, Ankara. His current research interests include VLSI design, energy-efficient and high-performance computer architectures, and dependable/reliable systems.

Osman S. Unsal received the BS degree in electrical and computer engineering from Istanbul Technical University, Istanbul, in 1987, the MS degree in electrical and computer engineering from Brown University, Providence, Rhode Islands, in 1991, and the PhD degree in electrical and computer engineering from the University of Massachusetts, Amherst, in 2002. He is currently a senior researcher at the Barcelona Supercomputing Center. His current research interests include computer architecture, reliability, and ensuring programmer productivity. He is a member of the IEEE, the ACM, the Sigma Xi, and the Phi Kappa Phi.

Xavier Vera received the MS degree in computer science from the Universitat Politècnica de Catalunya, Barcelona, in 2000 and the PhD degree from Mälardalens Högskola, Västerås, Sweden, in 2004. Since 2004, he has been with the Intel Barcelona Research Center, where he is participating in research in the area of reliable and variations-aware microarchitectures.

Antonio González received the MS and PhD degrees from the Universitat Politècnica de Catalunya (UPC), Barcelona. He is the founding director of the Intel Barcelona Research Center, which started in 2002, whose research focuses on new microarchitecture paradigms and code generation techniques for future microprocessors. Prior to this, he joined the faculty of the Computer Architecture Department, UPC, in 1986 and became a full professor in 2002. He has published more than 250 papers, has given more than 80 invited talks, has filed more than 40 patents, and has advised 13 PhD thesis in the areas of computer architecture and compilers. He is an associate editor of the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Architecture and Code Optimization*, and *Journal of Embedded Computing*. He has served on more than 100 program committees for international symposia in the field of computer architecture, including the International Symposium on Computer Architecture (ISCA), the Annual International Symposium on Microarchitecture (MICRO), the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), the International Symposium on High-Performance Computer Architecture (HPCA), the International Conference on Parallel Architectures and Compilation Techniques (PACT), the Annual International Conference on Supercomputing (ICS), the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), and the IEEE International Parallel and Distributed Processing Symposium (IPDPS). He has been a program (co-) chair for ICS 2003, ISPASS 2003, MICRO 2004, and HPCA 2008, among other symposia.