Reliable Cache Architectures and Task Scheduling for Multiprocessor Systems

Sugihara, Makoto
Department of Information and Computer Sciences, Toyohashi University of Technology

Ishihara, Tohru
System LSI Research Center, Kyushu University

Murakami, Kazuaki
Department of Informatics, Kyushu University

出版情報：IEICE transactions on Electronics. E91-C (4), pp.410-417, 2008-04-01. The Institute of Electronics, Information and Communication Engineers
バージョン：
権利関係：

PAPER Special Section on Advanced Technologies in Digital LSIs and Memories

Reliable Cache Architectures and Task Scheduling for Multiprocessor Systems*

Makoto SUGIHARA†, **Tohru ISHIHARA**††, ***and*** **Kazuaki MURAKAMI**†††, ***Members***

SUMMARY This paper proposes a task scheduling approach for reliable cache architectures (RCAs) of multiprocessor systems. The RCAs dynamically switch their operation modes for reducing the usage of vulnerable SRAMs under real-time constraints. A mixed integer programming model has been built for minimizing vulnerability under real-time constraints. Experimental results have shown that our task scheduling approach achieved 47.7-99.9% less vulnerability than a conventional one.

key words: *Single Event Upset, SRAM, DRAM, Reliability, Cache Architecture, Task Scheduling*

1. Introduction

A single event upset (SEU) on a memory module often causes a soft error of a computer system. Occurrence of SEUs in SRAM memories is becoming a critical issue as technology continues to shrink [5–7]. Embedding vulnerable SRAM modules into a system-on-a-chip (SOC) deteriorates reliability of the SOC. From a viewpoint of IC design, accurate reliability estimation and design for reliability (DFR) are becoming important in order that one applies reasonable DFR to vulnerable part of an IC.

Several system vulnerability estimation techniques have been proposed [8–11, 14]. A methodology and an algorithm have been proposed for estimating reliability of computer systems at the instruction set simulation (ISS) level [11, 14]. The estimation methodology adopts cycle-accurate simulation to identify which part of a memory is utilized spatially and temporally during executing programs. Identification of spatial and temporal usage of memory modules contributes to accurate estimation of reliability of computer systems. Since the abstraction level of ISS is higher than that of circuit simulation, ISS-based vulnerability estimation is faster than circuit simulation-based vulnerability estimation. ISS-based vulnerability estimation is appropriate to identifying vulnerability parts of an IC product with short development time.

Design for reliability (DFR) is also one of the themes of urgent concern. Coding and parity techniques are popular design techniques for detecting or correcting SEUs in memory modules. These techniques have been well studied and developed. Recently, Elakkumanan *et al*. have proposed a DFR technique for logic circuits, which exploits time redundancy by using scan flip-flops [2]. Their approach updates a pair of flip-flops at different moments for an output signal to duplicate for reliability purpose. Recently, we have proposed reliable cache architectures (RCAs), in which the structural redundancy of set associative cache memories is exploited for controlling reliability and performance of computer systems [12, 13]. The RCAs dynamically change their operation modes from reliability to performance modes or vice versa, for controlling reliability and performance of computer systems. Under the performance mode, cache ways operate same as ordinary set associative cache memories. On the contrary, under the reliability mode, cache memories, which are made of vulnerable SRAM cells, are made reliable by the following approaches: (i) Disable cache ways partly or wholly. (ii) Make two or more original cache ways constitute a single reliable cache way. The former approach disables vulnerable part of memories and eliminates their vulnerability from the total vulnerability of the computer system with some performance degradation. The latter approach decreases vulnerability of a cache way by making it multiplicate. Two or more original cache ways hold the same contents and behave as if they constitute a single reliable cache way. Occurrence of an SEU on an original cache way is detected by comparing the values of duplicated original cache ways. Three or more original cache ways can take majority among them and correct an SEU. Under the reliability mode of the RCA, the virtual size of a cache memory decreases while reliability of the cache memory increases.

The RCAs are capable of changing their operation modes from reliability to performance modes or vice versa. The operation mode of the RCAs must be optimally determined for both high reliability and high performance of computer systems. This paper proposes a task scheduling method which optimally determines time series of operation modes of a computer system so that its reliability is maximized under a real-time constraint. We build a mixed integer programming (MIP) model for the task scheduling problem. The target system of the task scheduling problem is a computer system in which multiple processors run on a non-preemptive real-time operating system (RTOS).

The remainder of this paper is organized as follows: Section 2 reviews a tradeoff between performance and reliability of a computer system. Section 3 reviews reliable cache

Manuscript received August 10, 2007.
Manuscript revised November 12, 2007.

†The author is with the Department of Information and Computer Sciences, Toyohashi University of Technology, Aichi, Japan.
††The author is with the System LSI Research Center, Kyushu Univeristy, Fukuoka, Japan.
†††The author is with the Department of Informatics, Kyushu University, Fukuoka, Japan.

*This paper was presented at the ACM/IEEE DATE Conference, Nice, France, April 2007.

architectures. Section 4 presents a mathematical model to schedule tasks for reliable cache architectures of multiprocessor systems. Section 5 presents experimental results on our task scheduling method. Finally, concluding remarks are provided in Section 6.

2. Performance and Reliability

A soft error rate (SER) is often utilized for measuring and evaluating vulnerability of a memory component. An SER is defined as the number of soft errors which occur during a certain time. All SEUs are regarded as critical to the memory component on measuring its SER. The SER is, however, directly inapplicable to estimating vulnerability of computer systems because computer systems dynamically behave to use memory modules temporally and spatially. Some of SEUs on memory modules make the computer systems faulty and the others not. In the sense, it is pessimistic to directly adopt the SERs for evaluating reliability of computer systems. From the viewpoint of executing a program, it is not an SER, which is the number of faults during a certain time, but the number of faults during a certain task that should be the metric for estimating reliability of computer systems.

In this section, we briefly review relation between performance and reliability of computer systems. We show the number of soft errors during execution of a program on a microprocessor-based system which consists of an ARM processor (ARMv4T, 200MHz), an instruction cache module, a data cache module, and a main memory module. The cache line size and the number of cache-sets are 32-byte and 32, respectively. We adopted the least recently used (LRU) policy [4] as the cache replacement policy. We evaluated reliability of the computer system, which adopted the write-through policy [4]. The cell-upset rates (CUR) of both SRAM and DRAM modules are shown in Table 1. We used the CURs shown by Slayman [15] as the ones of plain SRAMs and DRAMs. According to Baumann, error detection and correction (EDAC) or error correction codes (ECC) protection will provide a significant reduction in failure rates (typically 10k or more times reduction in effective error rates) [1]. We have assumed that introducing an ECC circuit makes reliability of memory modules 10k times higher.

Table 1 Cell-upset rates.

	Cell upset rates			
	[FIT/bit]		[errors/word/cycle]	
	non-ECC	ECC	non-ECC	ECC
SRAM	1.0×10^{-4}	1.0×10^{-8}	4.4×10^{-24}	4.4×10^{-28}
DRAM	1.0×10^{-8}	1.0×10^{-12}	4.4×10^{-28}	4.4×10^{-32}

Figure 1 shows vulnerability and runtime of a computer system versus the number of cache ways. In the figure, acronyms IL1, DL1, IMM, and DMM indicate an instruction L1 cache, a data L1 cache, an instruction main memory, and a data main memory respectively. Note that the

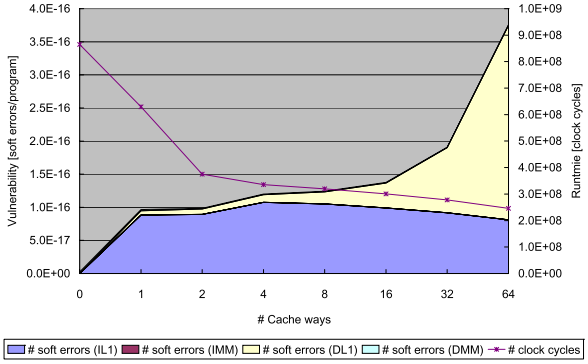


Fig. 1 Vulnerability vs cache size (ECC L1, ECC main memory).

size of a cache way is 1 kB and the cache size is linear to the number of cache ways. This figure shows that increasing the cache size of the computer system decreases its runtime and increases its vulnerability. In this computer system, the vulnerability of the cache memory is dominant in the entire vulnerability of the computer system. The vulnerability of the main memory is too small to see in the figure. The figure explicitly shows that there exists a tradeoff between vulnerability and runtime of a computer system. In other words, it shows that cache memory sizing contributes to adjusting vulnerability and runtime of the computer system. In this paper, we discuss dynamically changing the cache size for reliability and performance of computer system in which SRAMs are more vulnerable than DRAMs.

3. Reliable Cache Architectures

As we discussed in the previous section, decreasing the size of a cache memory contributes to increasing reliability of a computer system, in which the CUR of a cache memory is much higher than that of a main memory, with some performance degradation. In this section, we review reliable cache architectures (RCAs) which dynamically change the cache size, control the vulnerability, and affirmatively accept some performance degradation for increasing the reliability [12, 13]. The reliable cache architectures are useful especially for real-time systems in which the deadline times of tasks are given.

There are basically three approaches for increasing reliability of cache memories.

Naive RCA

In this cache architecture, not all cache ways are necessarily utilized for operations. All or some cache ways are deactivated under a reliability mode for increasing reliability of a computer system while all cache ways are activated under a performance mode as shown in Figure 2. Ideally speaking, every cache way should be capable of changing into reliability and performance modes. From the viewpoint of hardware implementation, some of cache ways may change its operation mode.

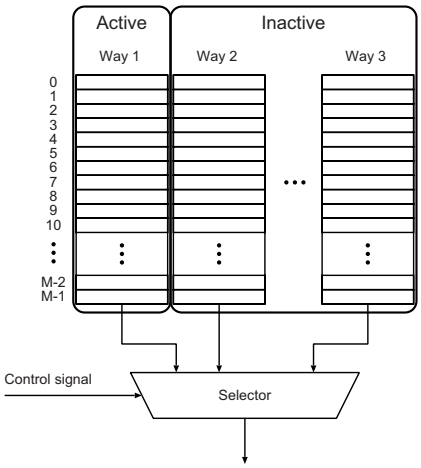


Fig. 2 Naive RCA for N-way set associative cache.

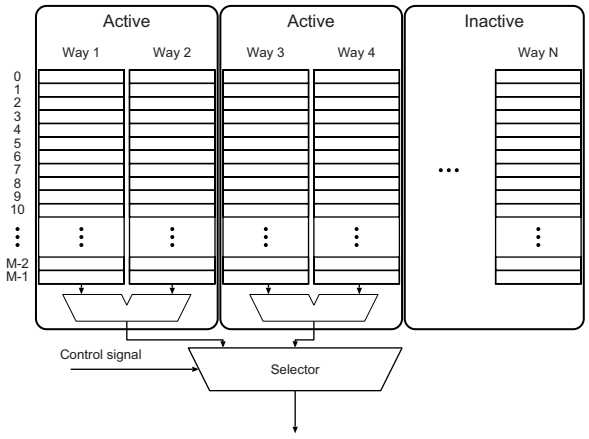


Fig. 3 Detection-oriented RCA for N-way set associative cache.

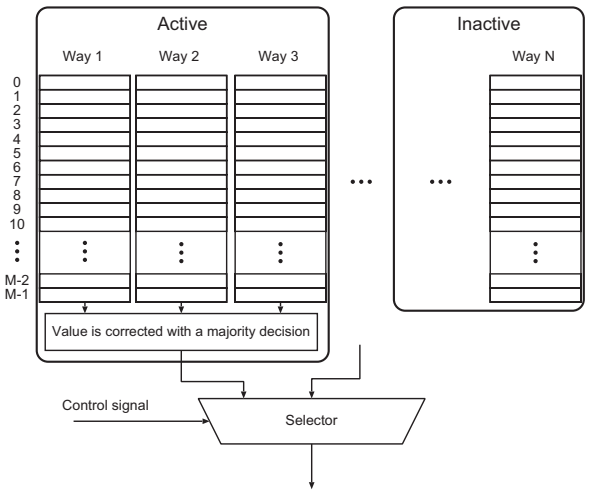


Fig. 4 Correction-oriented RCA for N-way set associative cache.

Detection-Oriented RCA

This cache architecture is error-detection-oriented. In this cache architecture, two cache ways are regarded as a redundant pair to constitute a single reliable cache way. The original cache ways are to hold the same content as each other as shown in Figure 3. If an SEU occurs on one of the original cache ways of the redundant pair, the SEU is promptly detected by comparing the contents of the original cache ways before the CPU manages the SEU.

Correction-Oriented RCA

This cache architecture is error-correction-oriented. In this cache architecture, three or more cache ways are regarded as a redundant set and they retain same content as one another as shown in Figure 4. If an SEU occurs on one of the original cache ways, the SEU is promptly detected and corrected by majority rule. A correct value is made by majority among the corresponding cache ways.

The merits and demerits of the three RCAs are summarized in Table 2. Computer system designers should choose one from the three RCAs according as their products require. It is relatively easy to implement the RCAs of write-through cache systems. Depending on the operation modes, cache ways are chosen for read and write and additional selector and comparator are added in front and back of memory modules. A selector to select a way (ways) for write is added for all RCAs, and a selector, a comparator and selector, and a majority circuit and selector are added for read of naive, detection-oriented, and correction-oriented RCAs respectively. Implementation of RCAs of write-back cache systems is more complex than that of a write-through one. On changing operation modes, dirty data items on a cache should be written out to a lower level of memory hierarchy. There are two ways to write dirty data items out to a lower level of memory hierarchy. One is a software approach and the other is a hardware one. In a software approach, the dirty data items are written out by writing any data item at a hypothetical address corresponding to every data item. In a hardware approach, the extra logic is added which write the dirty data items out to a lower level of cache memory by looking at their dirty flags.

Table 2 Three cache architectures.

	Reliability	Area overhead
Naive	low	low
Detection-oriented	middle	middle
Correction-oriented	high	high

	Performance	Power	Detection	Correction	Cache size
	high	low	no	no	$\leq 100\%$
	middle	high	yes	no	$\leq 50\%$
	low	high	yes	yes	$\leq 33.3\%$

4. Task Scheduling for Reliable Cache Architectures of Multiprocessor Systems

In this section, we discuss reliability maximization by task scheduling for a non-preemptive real-time operating system (RTOS) which runs on a multiprocessor system. We build a mixed integer programming (MIP) model to solve a task scheduling problem for a computer system in which an RCA is utilized. Once an MIP model is obtained for the problem, a problem instance for the MIP model can be solved with an MIP solver. Using an MIP solver conceils its concrete optimization algorithms from users.

Before we build an MIP model, we review an MIP model. An MIP model is generally described as follows [16]:

$$\begin{aligned} \text{Minimize:} \quad & \mathbf{Ax} + \mathbf{By} \\ \text{subject to:} \quad & \mathbf{Cx} + \mathbf{Dy} \leq \mathbf{E}, \text{ such that } \mathbf{x} \geq 0, \mathbf{y} \geq 0, \end{aligned}$$

where $\mathbf{A}$ and $\mathbf{B}$ are cost vectors, $\mathbf{C}$ and $\mathbf{D}$ are constraint matrices, $\mathbf{E}$ is a column vector of constants, $\mathbf{x}$ is a vector of integer variables, and $\mathbf{y}$ is a vector of real variables. Efficient MIP solvers are readily available [17].

We now address a task scheduling problem for a non-preemptive RTOS on a multiprocessor system, on which N_{T} tasks are executed. Preemption causes large deviations between the worst-case execution times (WCET) of tasks that can be statically guaranteed and average-case behavior. Non-preemptivity gives a better predictability on runtime since the worst-case is closer to the average case behavior. Task i, $1 \leq i \leq N_{\mathrm{T}}$, becomes available to start at its arrival time a_i and must finish by its deadline time d_i. The RCAs, shown in Section 3, select their operation mode from "the reliability mode" or "the performance mode." Switching the operation modes contributes to controlling both vulnerability and runtime of a task. We define an *RCA cache configuration*, or simply cache configuration in this paper, as time series of switching the operation modes for finishing a certain task. Suppose we have M_i alternative RCA cache configurations for Task i. Each RCA cache configuration for a task is different from one another with regard to the time at which the operation modes are switched. The time at which the operation modes are switched is determined by choosing an RCA cache configuration for the task. Figure 5 shows vulnerability and performance of a computer system on various RCA cache configurations for executing a task. For evaluating runtime and SEU vulnerability of an RCA-based system, we have assumed that the computer system has a single CPU, L1 cache memory, and main memory and that it operates under its performance and reliability modes. The L1 cache memory behaves as four-way set-associative cache memory under the performance mode while it is disabled under the reliability mode. We have adopted the SEU vulnerability estimation methodology [11, 14] for calculating the SEU vulnerability of the RCA-based computer system. In every RCA cache configuration of the task, the task starts with the reliability mode and then the operation mode is turned into the performance mode before it finishes. In the figure, the horizontal axis is the time at which the operation mode is turned from the reliability mode into the performance one. This figure obviously shows that there is a tradeoff between reliability and performance for executing a task. We denote vulnerability of Cache Configuration j for Task i by $v_{i,j}$. Similarly, we denote runtime of Cache Configuration j for Task i by $l_{i,j}$. Let s_i be a variable for the start time of Task i.

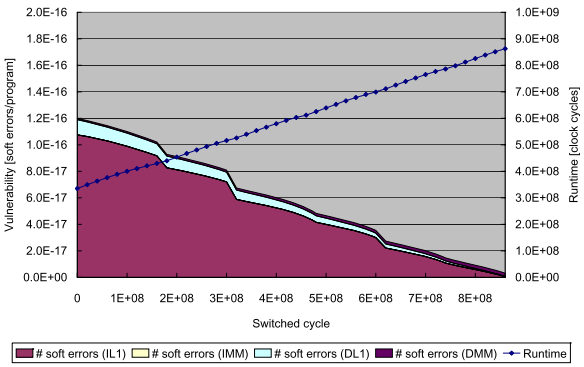


Fig. 5 Vulnerability and performance by switching reliability to performance mode.

The task scheduling problem that we address in this section is to minimize system vulnerability by optimally determining the start times $s_1, s_2, \cdots, s_{N_{\mathrm{T}}}$ and the RCA cache configuration of every task. The problem $\mathcal{P}$ is formally stated as follows.

- $\mathcal{P}$: Given a multiprocessor system, which consists of N_{P} processors, N_{T} tasks, their arrival and deadline times of Task i, a_i and d_i, M_i RCA cache configurations for Task i, and vulnerability $v_{i,k}$ and runtime $l_{i,k}$ of the kth RCA cache configuration of Task i, assign every task to a processor, select an RCA cache configuration for each task, and determine the start time for each task such that (1) all tasks are executable, (2) every task completes by its deadline, and (3) the overall system vulnerability is minimized.

We now develop an MIP model for Problem $\mathcal{P}$. We first formulate a nonlinear model, and then linearize it using standard techniques of linearization [16].

Let $x_{i,j}$, $1 \leq i \leq N_{\mathrm{T}}$, $1 \leq j \leq N_{\mathrm{P}}$, be a binary variable defined as follows:

$$x_{i,j} = \begin{cases} 1 & \text{if Task } i \text{ is assigned to Processor } j, \\ 0 & \text{otherwise.} \end{cases}$$

A task is assigned to a single processor. The following constraint, therefore, is introduced.

$$\sum_j x_{i,j} = 1, 1 \leq i \leq N_{\mathrm{T}}.$$

Let $y_{i,k}$, $1 \leq i \leq N_{\mathrm{T}}$, $1 \leq k \leq M_i$, be a binary

variable defined as follows:

$$y_{i,k} = \begin{cases} 1 & \text{if Cache Configuration } k \text{ is adopted for Task } i, \\ 0 & \text{otherwise.} \end{cases}$$

Vulnerability of a computer system is the sum of vulnerabilities of all tasks. Vulnerability of a task is determined by the RCA cache configuration which is adopted. Vulnerability of the computer system, therefore, is stated as follows.

$$V = \sum_{i,k} v_{i,k} y_{i,k}.$$

A single RCA cache configuration is chosen for each task and therefore the following constraint is introduced.

$$\sum_{k} y_{i,k} = 1, 1 \leq i \leq N_{\mathrm{T}}.$$

Task i starts between its arrival time a_i and its deadline time d_i. The start time s_i is, therefore, bounded as follows.

$$a_i \leq s_i \leq d_i, 1 \leq i \leq N_{\mathrm{T}}.$$

Task i must finish by its deadline time d_i. A constraint on the deadline time of a task is introduced as follows.

$$s_i + \sum_{k} l_{i,k} y_{i,k} \leq d_i, 1 \leq i \leq N_{\mathrm{T}}.$$

Now assume that two tasks $i1$ and $i2$ are assigned to a processor. Two tasks are simultaneously inexecutable on the single processor. The two tasks must be sequentially executed on the single processor. Two tasks $i1$ and $i2$ are inexecutable on the single processor if (i) $s_{i1} < s_{i2} + \sum l_{i2,k} y_{i2,k}$ and $s_{i1} + \sum l_{i1,k} y_{i1,k} > s_{i2}$, or (ii) $s_{i2} < s_{i1} + \sum l_{i1,k} y_{i1,k}$ and $s_{i2} + \sum l_{i2,k} y_{i2,k} > s_{i1}$. The two tasks, inversely, are executable on the processor under the following constraints.

$$\begin{aligned} & x_{i1,j} = x_{i2,j} = 1 \\ & \rightarrow \left\{ \left(s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} \leq s_{i2} \right) \right. \\ & \left. \vee \left(s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} \leq s_{i1} \right) \right\}, \\ & 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}. \end{aligned} \tag{1}$$

The task scheduling problem is now stated as follows.

Minimize the cost function $V = \sum_{i,k} v_{i,k} y_{i,k}$
subject to

1. $\sum_{j} x_{i,j} = 1, 1 \leq i \leq N_{\mathrm{T}}$.
2. $\sum_{k} y_{i,k} = 1, 1 \leq i \leq N_{\mathrm{T}}$.
3. $s_i + \sum_{k} l_{i,k} y_{i,k} \leq d_i, 1 \leq i \leq N_{\mathrm{T}}$.
4. $x_{i1,j} = x_{i2,j} = 1 \rightarrow \{(s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} \leq s_{i2}) \vee (s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} \leq s_{i1})\}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.

Variables

- $x_{i,j}$ is a binary variable, $1 \leq i \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $y_{i,k}$ is a binary variable, $1 \leq i \leq N_{\mathrm{T}}, 1 \leq k \leq M_i$.
- s_i is a real variable, $1 \leq i \leq N_{\mathrm{T}}$.

Bounds

- $a_i \leq s_i \leq d_i, 1 \leq i \leq N_{\mathrm{T}}$.

The above constraint 4) is nonlinear and must be linearized for solving the problem as an MIP model. $A \rightarrow B \vee C$ is the same as $(A \rightarrow B) \vee (A \rightarrow C)$ and therefore the constraint 4) can be written as follows.

$$\begin{aligned} & \left\{ \left(x_{i1,j} = x_{i2,j} = 1 \right) \rightarrow \left(s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} \leq s_{i2} \right) \right\} \\ & \vee \left\{ \left(x_{i1,j} = x_{i2,j} = 1 \right) \rightarrow \left(s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} \leq s_{i1} \right) \right\} \end{aligned}$$

The above constraint is a disjunction of two subconstraints and is nonlinear. The subconstraints themselves are also nonlinear. We first linearize the subconstraints then the disjunction of the two subconstraints. The subconstraints are linearized as follows.

Linearizing $(x_{i1,j} = x_{i2,j} = 1) \rightarrow (s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} \leq s_{i2})$

- $s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} - s_{i2} - M_{i1,i2}(1 - z_{i1,i2,j}) \leq 0$, $1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $M_{i1,i2} = d_{i1} + \max_k l_{i1,k} - a_{i2}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}$.
- $z_{i1,i2,j} = x_{i1,j} \cdot x_{i2,j}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $z_{i1,i2,j} \leq x_{i1,j}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $z_{i1,i2,j} \leq x_{i2,j}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $z_{i1,i2,j} \geq x_{i1,j} + x_{i2,j} - 1, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.

Linearizing $(x_{i1,j} = x_{i2,j} = 1) \rightarrow (s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} \leq s_{i1})$

- $s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} - s_{i1} - M_{i2,i1}(1 - z_{i1,i2,j}) \leq 0$, $1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $M_{i2,i1} = d_{i2} + \max_k l_{i2,k} - a_{i1}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}$.

The disjunction of the two subconstraints are linearized as follows.

Linearizing $\{s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} - s_{i2} - M_{i1,i2}(1 - z_{i1,i2,j}) \leq 0\} \vee \{s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} - s_{i1} - M_{i2,i1}(1 - z_{i1,i2,j}) \leq 0\}$

- $s_{i1} + \sum_{k} l_{i1,k} y_{i1,k} - s_{i2} - M_{i1,i2}(1 - z_{i1,i2,j}) - M_{i1,i2}(1 - \delta_{i1,i2,j,1}) \leq 0$, $1 \leq i1 < i2 \leq N_{\mathrm{T}}$, $1 \leq j \leq N_{\mathrm{P}}$.
- $s_{i2} + \sum_{k} l_{i2,k} y_{i2,k} - s_{i1} - M_{i2,i1}(1 - z_{i1,i2,j}) - M_{i2,i1}(1 - \delta_{i1,i2,j,2}) \leq 0$, $1 \leq i1 < i2 \leq N_{\mathrm{T}}$, $1 \leq j \leq N_{\mathrm{P}}$.
- $\delta_{i1,i2,j,1} + \delta_{i1,i2,j,2} \geq 1, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq$

$j \leq N_{\mathrm{P}}$.

The task scheduling problem is finally stated as follows.

Minimize the cost function $V = \sum_{i,k} v_{i,k} y_{i,k}$
subject to

1. $\sum_j x_{i,j} = 1, 1 \leq i \leq N_{\mathrm{T}}$.
2. $\sum_k y_{i,k} = 1, 1 \leq i \leq N_{\mathrm{T}}$.
3. $s_i + \sum_k l_{i,k} y_{i,k} \leq d_i, 1 \leq i \leq N_{\mathrm{T}}$.
4. $s_{i1} + \sum_k l_{i1,k} y_{i1,k} - s_{i2} - M_{i1,i2}(2 - z_{i1,i2,j} - \delta_{i1,i2,j,1}) \leq 0$,
   $1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
5. $s_{i2} + \sum_k l_{i2,k} y_{i2,k} - s_{i1} - M_{i2,i1}(2 - z_{i1,i2,j} - \delta_{i1,i2,j,2}) \leq 0$,
   $1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
6. $\delta_{i1,i2,j,1} + \delta_{i1,i2,j,2} \geq 1, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
7. $z_{i1,i2,j} \leq x_{i1,j}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
8. $z_{i1,i2,j} \leq x_{i2,j}, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
9. $z_{i1,i2,j} \geq x_{i1,j} + x_{i2,j} - 1, 1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.

Variables

- $x_{i,j}$ is a binary variable, $1 \leq i \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $y_{i,k}$ is a binary variable, $1 \leq i \leq N_{\mathrm{T}}, 1 \leq k \leq M_i$.
- $z_{i1,i2,j}$ is a binary variable, $1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- $\delta_{i1,i2,j,1}$ and $\delta_{i1,i2,j,2}$ are binary variables, $1 \leq i1 < i2 \leq N_{\mathrm{T}}, 1 \leq j \leq N_{\mathrm{P}}$.
- s_i is a real variable, $1 \leq i \leq N_{\mathrm{T}}$.

Bounds

- $a_i \leq s_i \leq d_i, 1 \leq i \leq N_{\mathrm{T}}$.

Solving the above MIP model yields the optimal schedule which achieves the minimal vulnerability under a certain real-time constraint.

5. Experimental Results

5.1 Experimental Setup

In order to observe effect of our task scheduling method which is stated in Section 4, we adopted a computer system, which has two ARM CPU cores (ARMv4T, 200 MHz), for experiment. Each CPU core has a 4 kB four-way set-associative instruction cache, a 4 kB four-way set-associative data cache, and a main memory modules, as shown in Figure 6. We adopted the naive RCA for cache memories, which was presented in Section 3. The performance mode enabled all cache ways while the reliability mode disabled all of them. We assumed that two CPU cores have their own main memory and can independently transfer data from/to their own main memory. The cache line size and the number of cache-sets are 32-byte and 32, respectively. We adopted the LRU policy for cache line replacement. We used the CURs of both SRAM and DRAM modules which are shown in Table 1.

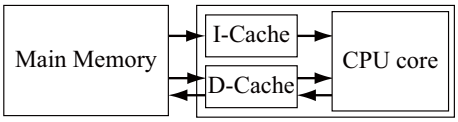


Fig. 6 A target system.

We used nine benchmark programs as shown in Table 3. Three of the benchmark programs are *Compress* version 4.0, *JPEG encoder* version 6b, and *MPEG2 encoder* version 1.2. We also used *basicmath*, *bitcnts*, *qsort*, *susan1*, *susan2*, and *susan3*, which are supplied from MiBench, an embedded benchmark suite [3]. Varying the time at which the operation mode turns from the reliability mode into the performance one, we made up several RCA cache configurations for every task. The numbers of RCA cache configurations are shown in Table 3. Minimal and maximal runtimes, and minimal and maximal vulnerabilities for each task among its RCA cache configurations are also shown in Table 3. RCA cache configurations are different from each other regarding time series of switching the operation modes, concretely speaking, when operation modes switch from reliability mode into performance one in this experiment.

We used the GNU C compiler and debugger to generate address traces. All programs were compiled with "-O3" option. We assumed that the operation mode of a CPU core may change only once from reliability mode to performance one during execution of each task.

We used an ILOG CPLEX 9.1 optimization engine [17] to solve the MIP model shown in Section 4 so that vulnerability of the computer system was minimized. We solved all scheduling problem instances on an AMD Opteron 275 processor which runs at 2.2 GHz.

Figure 7 shows a flow of our experiment. First of all, all traces for benchmark programs have been obtained. Next, we have defined a hypothetical multiprocessor system as we described at the beginning of this section. And then trace-driven ISS has been executed for obtaining runtime and SEU vulnerability of every program. At the same time, we have given hypothetical arrival and deadline times to every program as shown in Table 3. Finally, an MIP problem instance has been solved with a commercial MIP solver. Once all the values of runtime, vulnerability for all RCA cache configuration of every task, and arrival and dead times are obtained, the MIP solver can seek for a solution of optimal schedule to minimize SEU vulnerability as shown in Figure 7.

5.2 Experimental Results

Vulnerability of the computer system was minimized under five scenarios. The scenarios are different from one another regarding both arrival and deadline times. The arrival and

Table 3 Benchmark programs.

		Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9
Program name		Compress	JPEG	MPEG2	basicmath	bitcnts	qsort	susan1	susan2	susan3
# RCA cache configurations		45	43	35	24	19	14	16	18	16
Min runtime [10^6 cycles/task]		335	187	138	898	74	48	23	20	6
Max runtime [10^6 cycles/task]		865	821	679	2,258	346	138	149	32	14
Min vulnerability [10^{-21} errors/task]		7	18	4	111	9	2	3	1	0
Max vulnerability [10^{-21} errors/task]		27,050	36,402	12,579	64,846	7,477	2,769	2,116	1,902	655
Arrival and deadline times, a_i and d_i [10^6 cycles]	Scenario 1	0/ASAP	0/ASAP	0/ASAP	0/ASAP	0/ASAP	0/ASAP	0/ASAP	0/ASAP	0/ASAP
	Scenario 2	0/1,500	0/1,500	0/1,500	0/1,500	1,100/1,500	1,300/1,500	1,300/1,500	1,400/1,500	1,400/1,500
	Scenario 3	0/2,000	0/2,000	0/2,000	0/2,000	1,600/2,000	1,800/2,000	1,800/2,000	1,900/2,000	1,900/2,000
	Scenario 4	0/2,500	0/2,500	0/2,500	0/2,500	2,100/2,500	2,300/2,500	2,300/2,500	2,400/2,500	2,400/2,500
	Scenario 5	0/∞	0/∞	0/∞	0/∞	0/∞	0/∞	0/∞	0/∞	0/∞

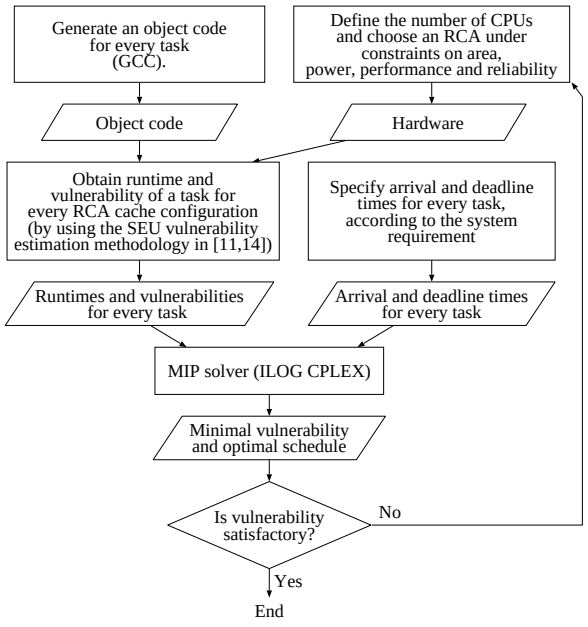


Fig. 7 Experiment flow

deadline times for all tasks are shown in Table 3. In Scenario 1, all tasks become available from the beginning and are required to finish as soon as possible. The vulnerability of Scenario 1 is basically equivalent to that of a non-RCA computer system. In Scenario 5, all tasks become available from the beginning and have no real time constraints. The vulnerability of Scenario 5 is basically equivalent to that of a computer system which has no cache memories.

Table 4 shows vulnerability, runtime of the computer system, and computation time under the five scenarios. Note that the vulnerability of Scenario 1 is basically equivalent to that of a non-RCA computer system. The vulnerability of Scenario 1 is the upper bound of vulnerability and the runtime is the lower bound of runtime. Comparing with the vulnerability value of Scenario 1, we have found that our task scheduling method has achieved 47.7-99.9% less vulnerability as the deadline constraints have made some slack. Runtime of the computer system has become 1.6-3.0 times longer. Note that increase of runtime has no disadvantage as far as task scheduling satisfies deadline constraints, that is real-time ones.

Table 4 Experimental results.

	Vulnerability [10^{-21} errors]	Runtime [10^6 cycles]	Computation time [s]
Scenario 1	6,855,090.3	966.9	0.0
Scenario 2	3,585,254.8	1500.0	351.9
Scenario 3	1,617,569.3	2000.0	145.3
Scenario 4	138,019.2	2500.0	1168.1
Scenario 5	6,813.8	2875.1	0.0

It has taken less than a second to minimize vulnerability under Scenarios 1 and 5 while about two to twenty minutes under Scenarios 2, 3 and 4. The reason why the computation time has become so short under Scenario 1 has been that the RCA cache configuration of the shortest runtime, in which no reliability mode has been used, has been always chosen as an optimal one for every task. Similarly, the reason why the computation time has become so short under Scenario 5 has been that the RCA cache configuration of the longest runtime, in which no performance mode has been used, has been always chosen as an optimal one for every task. In Scenarios 2, 3 and 4, tasks have conflicted with the others and the solution space on which the optimization engine has sought for an optimal solution has become large. However, optimization time less than 20 minutes is practical for statically minimizing vulnerability of a computer system offline. A heuristic task scheduling method should be studied and developed for adaptively minimizing vulnerability online. Variation in runtime of tasks should be taken into account for developing online scheduling.

6. Conclusion

In this paper, we have presented a task scheduling method for reliable cache memories of non-preemptive multiprocessor systems, in which cache memories are capable of changing their operating modes from reliability to performance modes or vice versa, for controlling vulnerability and runtime of the computer systems. We have built an MIP model for minimizing vulnerability of a multiprocessor system under real-time constraints. We have presented several experiments, in which our task scheduling has achieved 47.7-99.9% less vulnerability than a naive task scheduling method for a non-RCA computer system.

Our task scheduling method presented in this paper

has been mainly built for statically minimizing vulnerability of non-preemptive multiprocessor systems offline. Our task scheduling method is effective especially for minimizing vulnerability of a computer system statically before its operation. Runtime of a task varies depending on its input data and probably becomes shorter than its WCET. Shorter execution time of the task makes slack for the other tasks to run under the reliability mode. An adaptive task scheduling method should be examined for exploiting slack for more reliable operation.

Acknowledgments

This work was supported by the Japan Science and Technology Agency (JST), CREST. This work was also supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc.

References

[1] R. B. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on device and materials reliability*, Vol. 5, No. 3, pp. 305-316, September 2005.

[2] P. Elakkumanan, K. Prasad, and R. Sridhar, "Time redundancy based scan flip-flop reuse to reduce SER of combinational logic," *Proc. IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 617-622, March 2006.

[3] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, "MiBench: A Free, commercially representative embedded benchmark suite," *Proc. IEEE Workshop on Workload Characterization*, December 2001.

[4] J. L. Hennessy and D. A. Patterson, "Computer architecture: a quantitative approach," pp. 401–402, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[5] N. Seifert, D. Moyer, N. Leland and R. Hokinson, "Historical trend in alpha-particle induced soft error rates of the Alpha(tm) microprocessor," *Proc. IEEE International Reliability Physics Symposium (IRPS)*, pp. 259–265, April 2001.

[6] N. Seifert, X. Zhu, D. Moyer, R. Mueller, R. Hokinson, N. Leland, M. Shade, and L. Massengill, "Frequency dependence of soft error rates for sub-micron CMOS technologies," in the Technical Digest of *International Electron Devices Meeting (IEDM)*, pp. 14.4.1–14.4.4, December 2001.

[7] T. Karnik, B. Bloechel, K. Soumyanath, V. De and S. Borkar, "Scaling trends of cosmic ray induced soft errors in static latches beyond 0.18μ," *Proc. Symposium on VLSI Circuits*, pp. 61–62, June 2001.

[8] H. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Vulnerability analysis of L2 cache elements to single event upsets," *Proc. Design, Automation and Test in Europe Conference (DATE)*, pp. 1276–1281, March 2006.

[9] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," *Proc. IEEE International Symposium on Computer Architecture (ISCA)*, pp. 532–543, June 2005.

[10] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch: An architecture level tool for modeling and analyzing soft errors," *Proc. IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 496–505, June 2005.

[11] M. Sugihara, T. Ishihara, K. Hashimoto, and M. Muroyama, "A simulation-based soft error estimation methodology for computer systems," *Proc. IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 196–203, March 2006.

[12] M. Sugihara, T. Ishihara, and K. Murakami, "An analysis on a trade-off between reliability and performance and a reliable cache architecture for computer systems," *IEICE Technical Report*, SIP2006-99, pp. 93-98, October 2006.

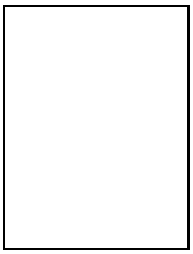
[13] M. Sugihara, T. Ishihara, and K. Murakami, "A task scheduling method for reliable cache architectures," *IEICE Technical Report*, CPSY2006-34, pp. 1-6, November 2006.

[14] M. Sugihara, T. Ishihara, and K. Murakami, "Architectural-level soft-error modelingfor estimating reliability of computer systems," *IEICE Transactions on Electronics*, Vol. E90-C, No. 10, pp. 1983-1991, October 2007.

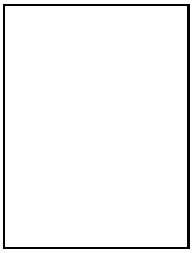
[15] C. W. Slayman, "Cache and memory error detection, correction and reduction techniques for terrestrial servers and workstations," *IEEE T-DMR*, 5(3):397–404, 2005.

[16] H. P. Williams, *Model Building in Mathematical Programming*, John Wiley & Sons, 1999.

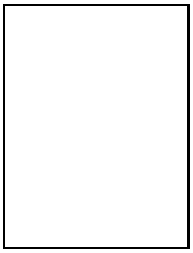
[17] ILOG Inc., CPLEX 9.1 Reference Manual, 2005.



Makoto Sugihara was born in Tokyo, Japan in 1974. He received the B.E., M.E. and Ph.D. degrees in computer science and communication engineering from Kyushu University, Japan in 1996, 1998 and 2001, respectively. He was a researcher at the Fujitsu Laboratories, Japan, in 1998. From 2002 to 2003, He was a visiting researcher at Duke University, NC, USA. From 2003 to 2007, he was a researcher at the ISIT, Fukuoka, Japan, and was also a visiting associate professor at Kyushu University, Fukuoka, Japan. He is currently a lecturer at Toyohashi University of Technology, Aichi, Japan. His research interest includes design and test methodologies as well as CAD algorithms for VLSI. He is a member of the IEEE and the IEEE Computer Society, the SPIE, the IEICE, and the IPSJ.



Tohru Ishihara received his B.S., M.S., and Ph.D. in computer science from Kyushu University in 1995, 1997, and 2000 respectively. From 1997 to 2000, he was a Research Fellow of the Japan Society for the Promotion of Science. For the next 3 years he worked as a Research Associate in VLSI Design and Education Center, the University of Tokyo. From 2003 to 2005, he stayed at Fujitsu Laboratories of America as a research staff of an advanced CAD technology group. In August 2005, he returned to Kyushu University as an associate professor. His research interests include low power SoC design and hardware/software co-design. He is a member of IEEE and IPSJ.



Kazuaki Murakami was born in Kumamoto, Japan in 1960. He received the B.E., M.E. and Ph.D. degrees in computer science and engineering from Kyoto University in 1982, 1984, and 1994, respectively. From 1984 to 1987, he worked for the Fujitsu Limited, where he was a Computer Architect of the mainframe computers. In 1987, he joined the Department of Information Systems of Kyushu University, Japan. He is currently a Professor of the Department of Informatics, and also the Director of the Computing and Communications Center. He is a member of the ACM, the IEEE, the IEEE Computer Society, the IPSJ, and the JSIAM.