

A Simulation-Based Soft Error Estimation Methodology for Computer Systems

Sugihara, Makoto
ISIT

Ishihara, Tohru
Kyushu Univ.

Hashimoto, Koji
Fukuoka Univ.

Muroyama, Masanori
Kyushu Univ.

<https://hdl.handle.net/2324/6794487>

出版情報 : Proc. of International Symposium on Quality Electronic Design, pp.196-203, 2006-03.
International Symposium on Computer Quality Electronic Design

バージョン :

権利関係 :



A Simulation-Based Soft Error Estimation Methodology for Computer Systems

Makoto Sugihara^{*1}, Tohru Ishihara^{*2}, Koji Hashimoto^{*3}, Masanori Muroyama^{*2}

^{*1} ISIT, 2-1-22 Momochihama, Sawara-ku, Fukuoka 814-0001, Japan

^{*2} Kyushu Univ., 3-8-33 Momochihama, Sawara-ku, Fukuoka 814-0001, Japan

^{*3} Fukuoka Univ., 8-19-1 Nanakuma, Jonan-ku, Fukuoka 814-0180, Japan

email: sugihara@isit.or.jp

Abstract

This paper proposes a simulation-based soft error estimation methodology for computer systems. Accumulating soft error rates (SERs) of all memories in a computer system results in pessimistic soft error estimation. This is because memory cells are used spatially and temporally and not all soft errors in them make the computer system faulty. Our soft-error estimation methodology considers the locations and the timings of soft errors occurring at every level of memory hierarchy and estimates the soft errors of the whole system using instruction-set simulation. Our experiment demonstrates that the reliability of computer systems depends on not only SERs of memories but also the behavior of software running on the systems.

Keywords: Soft Error, Reliability, Estimation, Computer Systems, Instruction-Set Simulation

1 Introduction

A hard error is caused by a permanent physical defect while a soft error usually refers to a transient defect which is caused by thermal neutrons, high-energy neutrons, or α particles. May first discovered that α particles emitted from radioactive substances caused soft errors in DRAM modules [3]. The feature size of integrated circuits has reached nano-scale and the nano-scale transistors become more soft-error sensitive. Soft error estimation and highly-reliable design become of utmost concern in mission-critical systems as well as consumer products.

Recently, several techniques for estimating soft errors have been proposed. In [1, 5, 11], soft error simulation with fault injection is discussed. In [1, 4], a soft error estimation technique is discussed for microprocessors. Soft error simulation in logic circuits are also being studied and developed [7–10]. The structure of memory modules is so regular and monotonous that it is comparatively easy to estimate the soft errors because they can be calculated with the SERs which are obtained by field or accelerated tests. These SERs of memory modules become pessimistic when they are embedded into computer systems. More specifically, every soft error occurred in memory modules is regarded as a critical error when memory modules are under field or accelerated tests. This implicitly assumes that every soft error on memory cells of a module make a computer system faulty. Since memory modules are used spatially and temporally in computer systems, every soft error on the memory modules does not necessarily make the computer system faulty. Therefore, the soft errors in an entire computer system should be estimated in a different way from the way used for memory modules.

Accurate soft error estimation of an entire computer sys-

tem is one of the themes of urgent concern. This paper discusses an accurate soft error estimation technique which can be adopted early in development of computer systems. It is necessary to consider the dynamic behavior of computer systems for accurately estimating the reliability of them. The duration during which a memory module or a logic circuit is being used depends on the behavior of a computer system programmed with software and affects its reliability. It is hard to estimate whether or not a soft errors in memory is critical to the computer system without knowing its behavior. The complicated structure of the hierarchical memory modules which are utilized in computer systems makes the soft error estimation of them harder. In such a computer system, it is hard to judge which part of memories is active to operate the computer system. Our soft error estimation technique computes the number of soft errors on all memory modules of the computer system by cycle-accurate instruction-set simulation (ISS) with taking their dynamic behavior into account. To our best knowledge, this is the first study to model the dynamic behavior of computer systems for estimating the accurate number of soft errors during the execution of a program.

The remainder of this paper is organized as follows: In Section 2, we review a model to estimate the number of soft errors on a word item in memory hierarchy. In Section 3, an soft error estimation model for instruction memory is discussed. The soft error estimation model for instruction memory is simple because instruction memory is basically read-only. In Section 4, a soft error estimation model for the number of soft errors in data memory is discussed. The estimation model for data memory is rather complicate because data memory is readable as well as writable. In Section 5, a simulation-based soft-error estimation algorithm for computer systems is discussed. Some experimental results are shown in Section 6. Section 7 concludes this paper with a summary.

2 Soft Errors on a Data Item

Unlike memory components, the soft error rates (SER) of computer systems vary every moment because the computer systems use memory modules spatially and temporally. Since only active part of the memory modules affects the reliability of computer systems, it is essential to identify the active part of memory modules for accurately estimating the number of soft errors occurring in the computer systems. A universal soft error index other than an SER is necessary to estimate reliability of computer systems because an SER is a reliability index suitable for components of regular and monotonous structure like memory modules but not for computer systems. In this paper, the number of

soft errors to run a program is adopted as a soft error index for computer systems.

In computer systems, a word item is a basic element for computation in CPUs. A word item is an instruction item in a instruction memory while that is a data item in a data memory. A collective of word items are required to be processed in order to run a program. We consider the reliability to process all word items as the reliability of a computer system. The total number of soft errors which are expected to occur on all the word items is regarded as the number of soft errors of the computer system. This section discusses an estimation model for the number of soft errors on a word item.

A CPU-centric computer system typically has the hierarchical structure of memory modules which include a register file, cache memory modules, and main memory modules. The computer system at which we target has N_{mem} levels of memory modules, $M_1, M_2, \dots, M_{N_{\text{mem}}}$ in order of accessibility from/to the CPU. In the hierarchical memory system, instruction items are generally processed as follows.

1. Instruction items are generated by a compiler and loaded into a main memory. The birth time of an instruction item is the time when the instruction item is loaded into the main memory, from the view point of a program execution.
2. When an instruction item is required by the CPU, the CPU fetches the instruction item from the memory module closest to the CPU. The instruction item is duplicated into all levels of memory modules which reside between the CPU and the master memory module.

Note that instruction items are basically read-only. Duplication of instruction items are unidirectionally made from a low level of a memory module to a high level of a memory module.

Data items in data memory are processed as follows.

1. Some data items are given as initial values of a program when the program is generated with a compiler. The birth time of such a data item is the time when the program is loaded into a main memory. The other data items are generated during execution of the program by the CPU. The birth time of the data item which is made on-line is the time when the data item is made and saved to the register file.
2. When a data item is required by a CPU, the CPU fetches the data item from the memory module closest to the CPU. If write allocate policy is adopted, the data item is duplicated at all levels of memory modules which reside between the CPU and the master memory module, otherwise it is not duplicated at the interjacent memory modules.

Note that data items are readable as well as writable. This means that data items can be copied from a high level of a memory module to a low level of a memory module, and vice versa. In CPU-centric computer systems, data items are utilized as constituent elements. The data items vary

in lifetime and the numbers of soft errors on the data items vary from data item to data item.

Let an SER on a word item on Memory Module M_i be SER_{M_i} . When a word item d is retained during Time $time$ in Memory Module M_i , the number of soft errors, $error_{M_i}(d)$, which is expected to occur on the word item is described as follows:

$$error_{M_i}(d) = SER_{M_i} \cdot time. \quad (1)$$

Word item d is required to be retained during Time $valid_time_{M_i}(d)$ in Memory Module M_i to transfer to the CPU. The number of soft errors, $error_{\text{system}}(d)$, which occur from the birth time to the time when the CPU fetches is given as

$$error_{\text{system}}(d) = \sum_i SER_{M_i} \cdot valid_time_{M_i}(d), \quad (2)$$

where $valid_time_{M_i}(d)$ is necessary and minimal time to transfer the word item from the master memory module to the CPU, and depends on the memory architecture. This kind of retention time is exactly obtained with cycle-accurate simulation of the computer system.

3 Soft Errors in Instruction Memory

Each instruction item has its own lifetime while a program runs. The lifetime of each instruction item is different from one another and is not necessarily equal to the execution time of a program. Generally speaking, the birth time of instruction items is the time when they are loaded into main memory, from the view point of a program execution. It is necessary to identify which part of retention time of an instruction item in a memory module affects reliability of a computer system.

Now let us break down into the number of soft errors on an instruction item before the total number of soft errors in instruction memory is discussed. The i th instruction fetch of a CPU for an instruction item which resides at Address a is shown by $if(a, i)$. $if(a, 0)$ denotes the time when the instruction is loaded into the main memory. An example of some instruction fetches is shown in Figure 1. In this figure, the boxes show that the copies of the instruction item resides in the corresponding memory modules. The labels on the boxes show when the copies of the instruction items are born. In this example, the instruction item is fetched three times to the CPU.

On the first instruction fetch for the instruction item, a copy of the instruction item exists in neither the L1 nor the L2 cache memories. The instruction item resides only on the main memory. It is required that the instruction item is transferred from the main memory to the CPU. On transferring the instruction item to the CPU, its copies are made in the L1 and the L2 cache memory modules. In this example, we assume that some latency is necessary to transfer the instruction item from the main memory module to the L1 cache, the L2 cache, and the register file. When the instruction item in a source memory module is fetched to the CPU, any soft errors which occur after completing transferring the instruction item have no influence on the instruction

fetch. In the figure, the boxes with slanting lines are found to be the retention times whose soft errors make Instruction Fetch $if(a, 1)$ faulty. The soft errors during any other retention times are unknown to make the computer system faulty.

On the second instruction fetch for the instruction item, the instruction item resides only in the main memory, same as on the first instruction fetch. The instruction item is fetched from the main memory to the CPU, same as on the first instruction fetch. The dotted boxes are found to be the retention times whose soft errors make Instruction Fetch $if(a, 2)$ faulty. The soft errors of any other retention times are unknown to make the computer system faulty on Instruction Fetch $if(a, 2)$. Note that soft errors on the box with slanting lines in the main memory are already treated on Instruction Fetch $if(a, 1)$ and is not treated on Instruction Fetch $if(a, 2)$ in order to avoid counting soft errors duplicately.

On the third instruction fetch for the instruction item, the highest level of memory module which retains the instruction item is the L1 cache memory. Soft errors on the gray boxes are treated as what make Instruction Fetch $if(a, 3)$ faulty. The soft errors on any other boxes are not counted for Instruction Fetch $if(a, 3)$.

Now assume that a program is executed in a computer system. Given an input data to a program, let an instruction fetch sequence be $i_1, i_2, \dots, i_{N_{\text{inst}}}$ to run the program. And let necessary and minimal retention time for Instruction Fetch i_i to be on Memory Module M_j be $valid_time_{M_j}(i_i)$. The number of soft errors on Instruction Fetch i_i , $error(i_i)$, is given as follows.

$$error(i_i) = \sum_j error_rate_{M_j} \cdot valid_time_{M_j}(i_i). \quad (3)$$

The total number of soft errors in the computer system is shown as follows:

$$error(\mathbf{i}) = \sum_i error(i_i) \quad (4)$$

$$= \sum_i \sum_j error_rate_{M_j} \cdot valid_time_{M_j}(i_i), \quad (5)$$

where $\mathbf{i} = (i_1, i_2, \dots, i_{N_{\text{inst}}})$. Given the program of the computer system, $valid_time_{M_j}(i_i)$ can be exactly obtained by performing cycle-accurate simulation of the computer system.

4 Soft Errors in Data Memory

Data memory is readable as well as writable. It is more complex than instruction memory because word items are bidirectionally transferred between a high level of memory and a low level of memory. Some data items are given as an input to a program and the other data items are born during the program execution. Some data items are used and the others are unused even if they reside in memory modules. The soft errors during some retention time of a data item is influential in a computer system. The soft errors during the other retention time is not influential even if the data item is used by the CPU. A data item has valid or invalid part of time. It is quite important to identify valid or invalid retention time of a data item in order to accurately estimate the number of soft errors of a computer system. In this paper, valid retention time is sought out by using the following rules.

- A data item which is generated on compilation is born when it is loaded into main memory.
- A data item as input to a computer system is born when it is inputted to the computer system.
- A data item is born when the CPU issues a store instruction for the data item.
- A data item is valid at least until the time when the CPU loads the data item and uses it in its operation.
- A data item which a user specifies as a valid one is valid even if the CPU does not issue a load instruction for the data item.

The bidirectional copies between high-level and low-level memory modules must be taken into account in data memories because the data memories are readable as well as writable. According to [2], there are two basic options on cache hit when writing to the cache:

- *Write through:* the information is written to both the block in the cache and to the block in the lower-level memory.

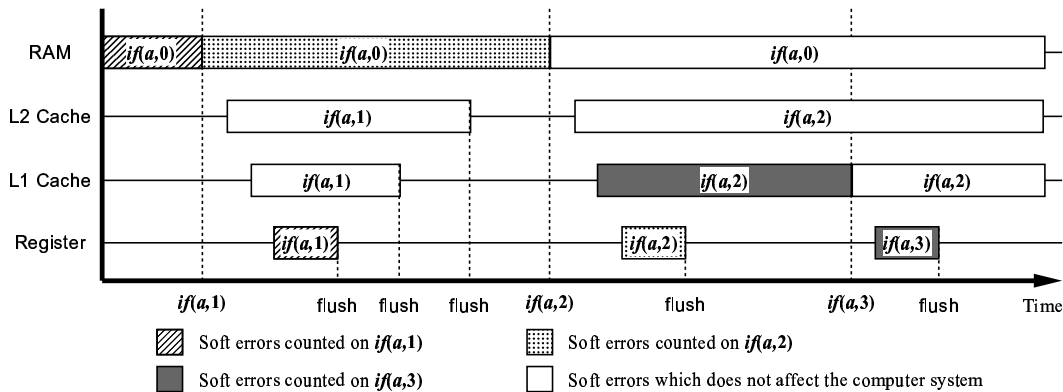


Figure 1. Soft errors which affect a computer system in instruction memory.

- *Write back*: the information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

The write policies affects the estimation for the number of soft errors and should be taken into account.

4.1 Soft Error Model in a Write-Back System

A soft-error estimation model in write-back systems is discussed in this section. Let the i th store operation of a CPU at Address a be $s(a, i)$ and the j th load operation at Address a be $l(a, j)$. An example of the behavior of a write-back system is shown in Figure 2. Each box in the figure shows the existence of the data item in memory modules. The labels on the boxes show when the data items are born. In the example, two store operations and two load operations are executed. First, a store operation is executed and only the L1 cache is updated with the data item. The L2 cache or main memory is not updated with the store operation. A load operation on the data item which resides at Address a follows. The data item resides in the L1 cache memory and is transferred from the L1 cache to the CPU. The soft errors on the boxes with slanting lines are found to be influential in reliability of the computer system by the issue of Load $l(a, 1)$. The other boxes with Label $s(a, 1)$ is unknown to be influential in the reliability. Next, the data item in the L1 cache goes out to the L2 cache by the other data item. The L2 cache memory becomes the highest level of memory which retains the data item. Next, a load operation $l(a, 2)$ is issued and the data item is transferred from the L2 cache memory to the CPU. With the load operation

$l(a, 2)$, the soft errors on the dotted boxes are found to be influential in reliability of the computer system. Soft errors on the white boxes labeled as $s(a, 2)$ are not counted on Load $l(a, 2)$.

4.2 Soft Error Model in a Write-Through System

A soft-error estimation model in write-through systems is discussed in this section. An example of the behavior of a write-through system is shown in Figure 3. First, a store operation at Address a is issued. Write-through architecture makes multiple copies of the data item in the cache memories and the main memory. Next, a load operation follows. The CPU fetches the data item from the L1 cache and soft errors on the boxes with slanting lines are found to be influential in reliability of the computer system. Next, a store operation $s(a, 2)$ comes. The previous data item at Address a is overridden and the white boxes labeled as $s(a, 1)$ are no longer influential in the reliability of the computer system. Next, the data item in the L1 cache is replaced with the other data item. The L2 cache becomes the highest level of memory which has the data item of Address a . Next, a load operation $l(a, 2)$ follows and the data item is transferred from the L2 cache to the CPU. With Load $l(a, 2)$, soft errors on the dotted boxes are found to be influential in the reliability of the computer system.

5 Simulation-Based Soft Error Estimation

As discussed in the previous sections, the retention time of a word item in memory modules needs to be obtained so that the number of soft errors in a computer system is es-

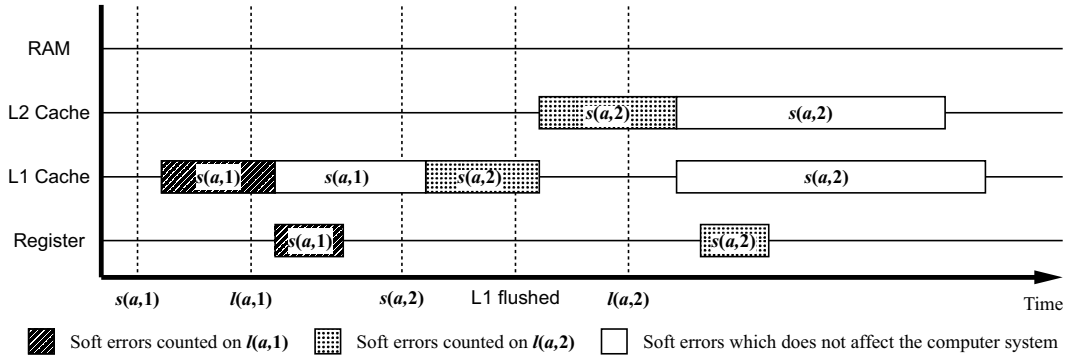


Figure 2. Soft errors which affect a computer system with write-back policy.

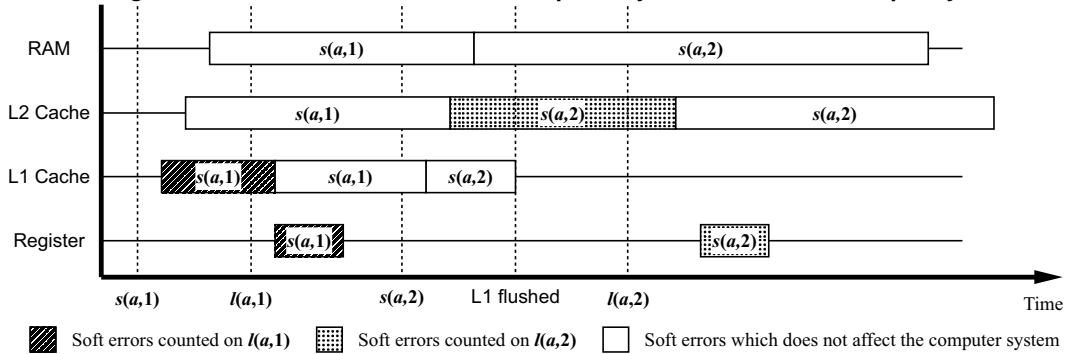


Figure 3. Soft errors which affect a computer system with write-through policy.

timated. A cycle-accurate Instruction-Set Simulation (ISS) is executed in order to obtain the retention time of a word item. A simplified algorithm to estimate the number of soft errors for a computer system to finish a program is shown in Figure 4. The input to the algorithm is an instruction sequence, and the output from the algorithm is the accurate number of soft errors, $error_{system}$, which occur during program execution.

First, several variables are initialized. $error_{system}$ is initialized with 0. The birth times of all data items are initialized with the time the program starts. A for-loop sentence follows. A cycle-accurate ISS is executed in the for-loop. An iteration loop corresponds to an execution of an instruction. The number of soft errors is counted for every instruction item and is added to $error_{system}$. If $error_{system}$ is updated, the birth time of the corresponding word item is updated with the present time. Some computation is done if the present instruction is a store or a load operation. If the instruction is a load operation, the number of soft errors on the data item which is found to be influential in the reliability of the computer system are added to $error_{system}$. A load operation updates the birth time of the data item with the present time. If the instruction is a store operation, the birth time of all word items changed and is updated with

Soft Error Estimation Algorithm

Procedure EstimateSoftError

Input Instruction sequence given by trace.

Output $error_{system}$: errors for the system.

begin

$error_{system}$ is initialized with 0.

 Birth time of every word item is initialized with the beginning time.

for all instructions **do**

 // Computation for soft errors

 // in instruction memory

 Add the number of soft errors which are found to be influential with the present instruction to $error_{system}$.

 Update the birth time on the instruction item with the present time.

 // Computation for soft errors in data memory

if the current instruction is a load **then**

 Add the number of soft errors which are found to be influential with the load operation to $error_{system}$.

 Update the birth time on the data item with the present time.

else if the current instruction is a store **then**

 Update the birth times on all changed data items with the present time.

endif

endfor

return $error_{system}$

end

Figure 4. A soft error estimation algorithm.

the present time. After the above procedure is applied to all instructions, $error_{system}$ is outputted as the number of soft errors which occur during a program execution.

6 Experiments and Results

6.1 Experimental Setup

We targeted a microprocessor-based system consisting of an ARM processor (ARMv4T, 200MHz), an instruction cache module, and a data cache module, and a main memory module as shown in Figure 5. The cache line size and the number of cache-sets are 32-byte and 32, respectively. We adopted the Least Recently Used (LRU) policy as the cache replacement policy. We evaluated reliability of computer systems with the two write policies, write-through and write-back ones. The cell-upset rates of both SRAM and DRAM modules are shown in Table 1. We used the cell-upset rates shown in [6] as the cell-upset rates of non-ECC SRAMs and non-ECC DRAMs.

Table 1. Cell-upset rates for experiment.

	Cell update 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}

We used three benchmark programs: *Compress* version 4.0 (Compress), *JPEG encoder* version 6b (JPEG), and *MPEG2 encoder* version 1.2 (MPEG2). We used the GNU C compiler and debugger to generate address traces. We chose to execute 100 million instructions in each benchmark program. This allowed the simulations to finish in a reasonable amount of time. All programs were compiled with “-O3” option. Table 2 shows the code size in words, active code size and active data size for each benchmark program. The active code and data sizes represent the number of different instruction and data addresses which are accessed during the execution of 100 million instructions, respectively.

Table 2. Specification of benchmark programs.

	Code size	Active code size	Active data size
	S_{code} [words]	AS_{code} [words]	AS_{data} [words]
Compress	10,716	1,874	140,198
JPEG	30,867	6,129	33,105
MPEG2	33,850	7,853	258,072

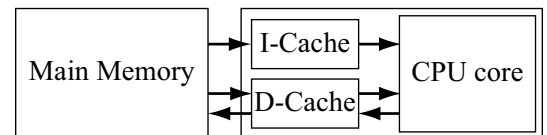


Figure 5. A target system.

6.2 Experimental Results

Figure 6 shows the results of our soft error estimation method. Four different memory configurations were considered as follows:

1. non-ECC L1 cache memory and non-ECC main memory,
2. non-ECC L1 cache memory and ECC main memory,
3. ECC L1 cache memory and non-ECC main memory,
4. and ECC L1 cache memory and ECC main memory.

The vertical axis represents the number of soft errors occurred during the execution of 100 million instructions. The horizontal axis represents the number of ways in a data cache. The other cache parameters, i.e., the line size and the number of lines in a cache way, are unchanged. The size of the data cache is, therefore, linear to the number of cache ways in this experiment. The cache sizes corresponding to the values shown on the horizontal axis are 1kB, 2kB, 4kB, 8kB, 16kB, 32kB, and 64kB, respectively.

The number of soft errors which occurred during a program execution depends on the reliability design of the memory hierarchy. When the cell-upset rate of SRAMs was higher than that of DRAMs, the soft errors on cache memories became dominant in the whole soft errors of the computer systems. The number of soft errors in a computer system, therefore, increased as the size of cache memories increased. When the cell-upset rate of SRAMs was lower than that of DRAMs, the soft errors on main memories became dominant in the system soft errors in contrast. The number of soft errors in a computer system, therefore, increased as the size of cache memories decreased because the larger size of cache memories reduced runtime to finish a program. Table 3 shows the number of CPU cycles to finish executing the 100 million instructions of each program.

Table 3. The number of CPU cycles [M cycles/100 M instructions].

		The number of ways in cache memory						
		1	2	4	8	16	32	64
Compress	WT	968	523	422	405	390	371	348
	WB	1,058	471	325	303	286	267	243
JPEG	WT	548	455	364	260	247	245	244
	WB	474	336	237	129	110	104	101
MPEG2	WT	497	179	168	168	167	167	167
	WB	446	124	110	110	110	110	110

Table 4 shows the results of more naive approaches which calculated the number of soft errors using the following equations.

$$SE_1 = \{S_{\text{cache}} \cdot SER_S + (S_{\text{code}} + AS_{\text{data}}) \cdot SER_D\} \cdot N_{\text{cycle}} \quad (6)$$

$$SE_2 = \{S_{\text{cache}} \cdot SER_S + (AS_{\text{code}} + AS_{\text{data}}) \cdot SER_D\} \cdot N_{\text{cycle}} \quad (7)$$

where S_{cache} , S_{code} , AS_{code} , AS_{data} , N_{cycle} , SER_S , SER_D denote the code size, the active code size, the active data

Table 4. # of soft errors [10^{-9} /100 M instructions].

		The number of cache ways						
		1	2	4	8	16	32	64
Compress	WT	M1	2267	2417	3869	7394	14216	27068
		M2	2263	2415	3867	7393	14214	27067
		M3	776	852	1248	1458	1541	1724
	WB	M1	2478	2175	2976	5530	10423	19461
		M2	2474	2173	2975	5529	10439	19460
		M3	999	881	1101	1372	1722	2484
JPEG	WT	M1	1262	2083	3324	4735	9013	17867
		M2	1255	2078	3320	4732	9010	17864
		M3	384	670	1355	2209	3417	4801
	WB	M1	1092	1540	2160	2355	4024	7593
		M2	1087	1536	2157	2354	4023	7592
		M3	369	558	941	1147	1664	2323
MPEG2	WT	M1	1197	838	1550	3167	6310	12217
		M2	1191	836	1548	3069	6118	12215
		M3	561	453	613	705	718	754
	WB	M1	1073	578	1019	2016	4016	8017
		M2	1067	577	1018	2015	4015	8016
		M3	494	321	410	474	492	534

size, the cache size, the number of CPU cycles, the SER for SRAM per word per cycle and the SER for DRAM per word per cycle, respectively. M1 and M2 appeared in Table 4 correspond to the calculations using Equations (6) and (7), respectively. Our method corresponds to M3. It is obvious that the naive approaches resulted in large overestimation of soft errors. This indicates that accumulating soft error rates of every memory modules in a system resulted in pessimistic estimation.

7 Conclusion

This paper discussed the simulation-based soft error estimation technique which sought the accurate number of soft errors for a computer system to finish running a program. Unlike the SERs of memory components, it is not important to seek for the static SER of a computer system in a conventional way, that is the simple summation of the SERs of memory modules. An important point to emphasize is that seeking for the number of soft errors to run a program is essential for accurate soft-error estimation for computer systems. We estimated the accurate number of soft errors of the computer systems based on ARM V4T architecture. The experimental results showed the following facts.

- There was a great difference between the number of soft errors derived with our technique and that derived from the naive summations of the static SERs of memory modules. The behavior of computer systems must be taken into account for accurate reliability estimation.
- The SER of a computer system virtually increases with a larger cache memory adopted because the SER is calculated by summing up the SERs of memory modules utilized in the system. It was, however, found that the number of soft errors to finish a program was reduced with larger cache memories in the computer system which had an ECC L1 cache and a non-ECC main

memory. This is because the soft errors in cache memories were negligible and the retention time of data items in the main memory was reduced by the performance improvement.

There are several issues with regard to the reliability of computer systems. Future work includes

- reliable design which determines the memory hierarchy of computer systems with our reliability estimation technique,
- design methodology which can treat a tradeoff between reliability and power consumption.

References

- [1] V. Degalahal et al., "SESEE: soft error simulation and estimation engine," *Proc. MAPLD International Conference*, 2004.
- [2] J. L. Hennessy et al., "Computer architecture: a quantitative approach," pp. 401–402, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
- [3] T. C. May et al., "Alpha-particle-induced soft errors in dynamic memories," *IEEE Transactions on Electron Devices*, vol. 26, pp. 2–7, 1979.
- [4] S. S. Mukherjee et al., "The soft error problem: an architectural perspective," *Proc. IEEE International Symposium on HPCA*, pp.243-247, 2005.
- [5] M. Rebaudengo et al., "An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor," *Proc. DATE*, pp.10602-10607, 2003.
- [6] C. W. Slayman, "Cache and memory error detection, correction and reduction techniques for terrestrial servers and workstations," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397-404, September 2005.
- [7] Y. Tosaka et al., "Neutron-induced soft error simulator and its accurate predictions," *Proc. IEEE International Conference on SISPAD*, pp. 253–256, 1997.
- [8] Y. Tosaka et al., "Simulation technologies for cosmic ray neutron-induced soft errors: models and simulation systems," *IEEE Transactions on Nuclear Science*, vol. 46, pp. 774-780, 1999.
- [9] Y. Tosaka et al., "Comprehensive study of soft errors in advanced CMOS circuits with 90/130nm technology," *IEEE IEDM*, pp. 941–948, 2004.
- [10] Y. Tosaka et al., "Comprehensive soft error simulator NISES II," *Proc. IEEE International Conference on SISPAD*, pp. 219–226, 2004.
- [11] N. J. Wang et al., "Characterizing the effects of transient faults on a high-performance processor pipeline," *Proc. IEEE International Conference on Dependable Systems and Networks*, pp.61-70, 2004.

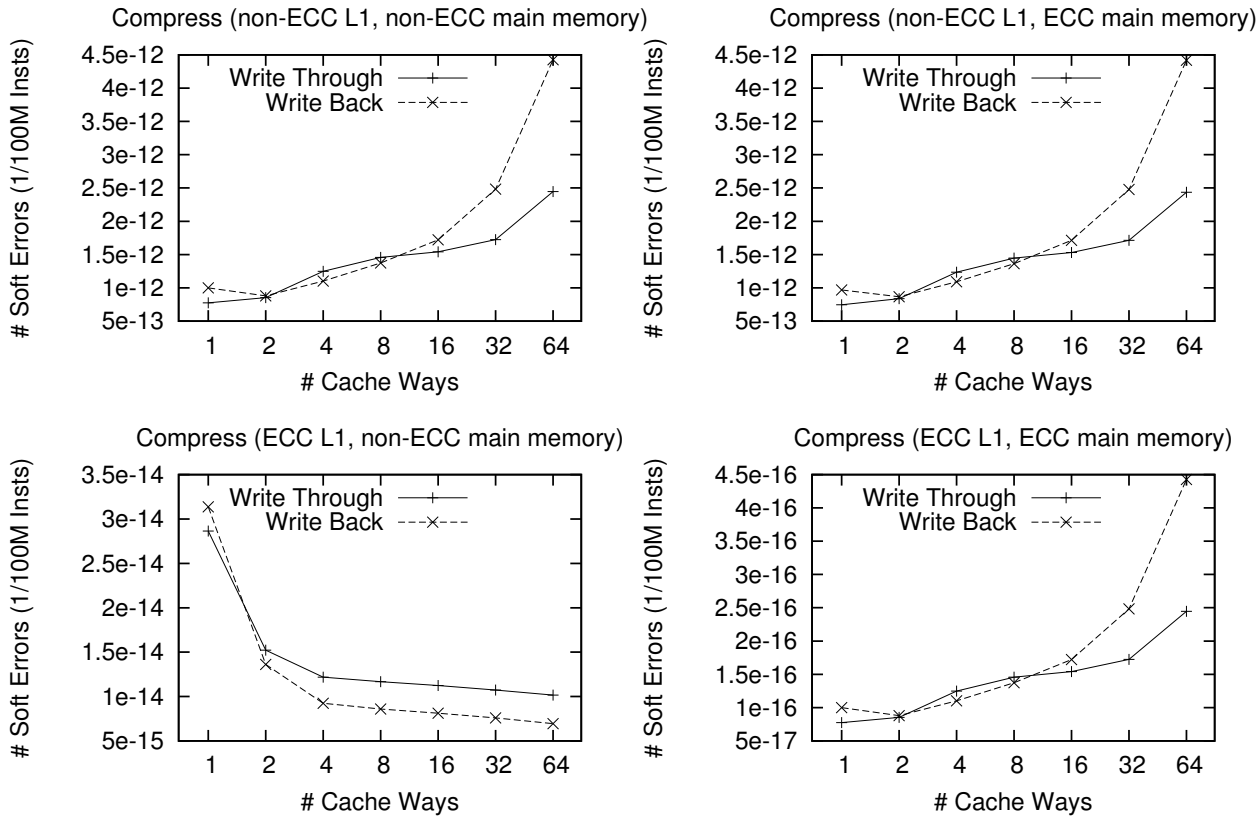


Figure 6. The number of soft errors.

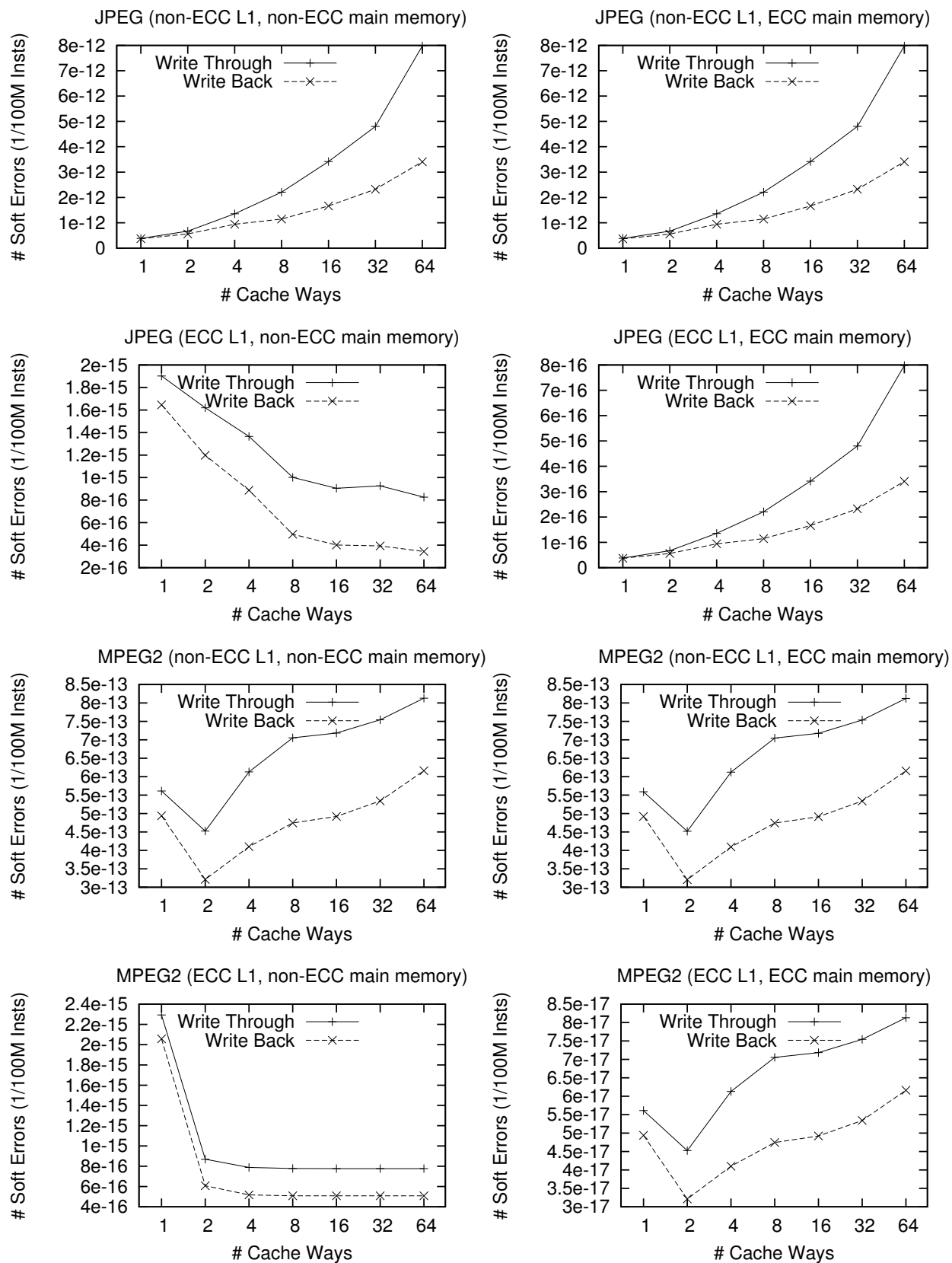


Figure 6. The number of soft errors (cont'd).