Cache Size Selection for Performance, Energy and Reliability of Time-Constrained Systems *

Yuan Cai[1], Marcus T. Schmitz[2], Alireza Ejlali[2], Bashir M. Al-Hashimi[2], Sudhakar M. Reddy[1]
[1]Department of Electrical and Computer Engineering, University of Iowa
E-mail: {yucai, reddy}@engineering.uiowa.edu
[2]School of Electronics and Computer Science, University of Southampton
Email: {ms4, ae04v, bmah}@ecs.soton.ac.uk

Abstract— Improving performance, reducing energy consumption and enhancing reliability are three important objectives for embedded computing systems design. In this paper, we study the joint impact of cache size selection on these three objectives. For this purpose, we conduct extensive fault injection experiments on five benchmark examples using a cycle-accurate processor simulator. Performance and reliability are analyzed using the performability metric. Overall, our experiments demonstrate the importance of a careful cache size selection when designing energy-efficient and reliable systems. Furthermore, the experimental results show the existence of optimal or Pareto-optimal cache size selection to optimize the three design objectives.

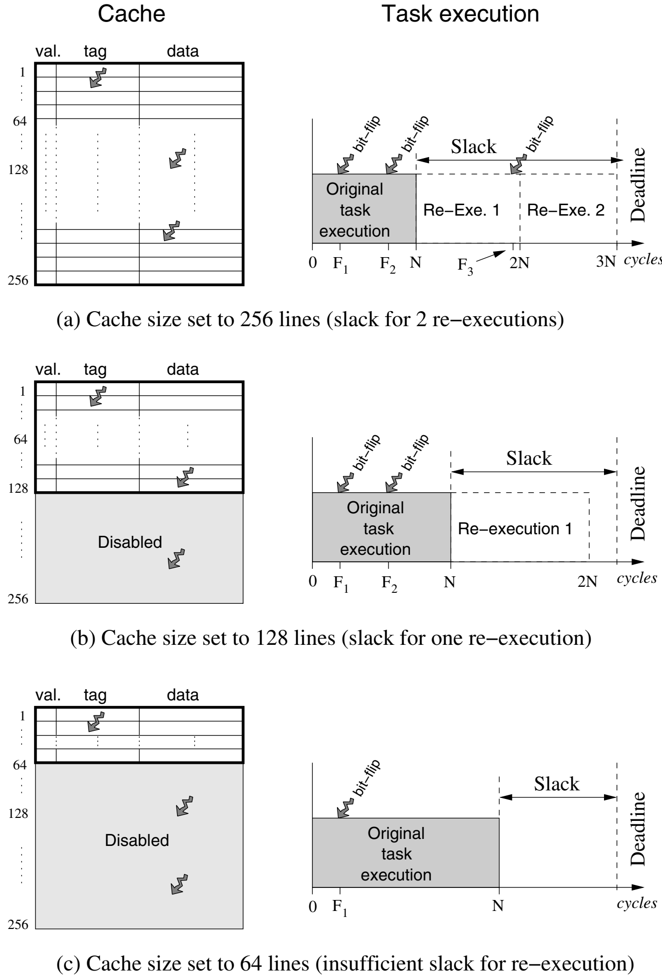
I. Introduction

Cache memories are widely used in microprocessors to improve the system performance [1]. As small, fast on-chip memories, caches store frequently accessed instructions and data to avoid a large number of accesses to the slow, off-chip main memory. Depending on the way the cache blocks are mapped onto the main memory, we distinguish between direct-mapped caches (each main memory address is mapped to one and only one cache block) and n-way set-associative caches (each main memory address can be mapped to n possible cache blocks). As opposed to the slow dynamic main memory, cache memories are implemented as flip-flops using static logic. Though cache memories can improve the system performance dramatically, they are responsible for a large portion of the overall system's power dissipation [2]. To reduce the energy dissipation, several approaches of dynamic cache reconfiguration have been reported. Zhang et al. [3] proposed a technique called way concatenation that tunes the cache ways between one (direct-mapped), two and four. Accordingly, at a reduced number of ways (1 and 2) the corresponding unused cache ways are disable. This is carried out under software control, i.e., at application run-time. They report an average energy saving of 40% compared to a fixed four-way cache. Similarly, Dropsho et al. [4] introduced a cache design, called accounting cache based on the selective ways cache [5]. The number of active ways is dynamically change under hardware control. Powell et al. [6] applied way-prediction and selective direct-mapping to reduce the set-associative cache energy. This is achieved by predicting the matching way and accessing only this matching one, instead of all ways. A dynamic online scheme that combines the processor voltage scaling and dynamic cache reconfiguration was proposed by Nacul et al. [7]. Their online algorithm adapts the processor speed and the cache subsystem to the workload requirements of the application. In a similar fashion, Zhang et al. [2] introduced an online heuristic that dynamically adjusts the cache size in order to minimize the cache energy. Their experiments point out that among all configurable cache parameters, cache size has the largest impact on cache performance and energy consumption. Yang et al. [8] investigated different design choices for resizable caches and evaluated their efficiency in reducing the system's energy dissipation. While these approaches are effective in reducing the energy dissipation, they neglect another important factor, namely the cache reliability.

Cache reliability is mainly threatened by transient faults, caused by strikes from alpha particles and energetic particles [9]. When a memory cell (flip-flop) is hit by such a particle, though the circuit itself is not damaged, the stored bit value can flip and cause an error. This problem is becoming more and more serious due to the ever-shrinking feature size and reduced supply voltage levels [10]. Hardware approaches that are dedicated to improve the cache reliability have been proposed. Such approaches make use of spatial redundancy to correct corrupted bits, for instance, data word parity [11] and Single Error Correct-Double Error Detect Error Correcting Codes (SEC-DED ECC) [12]. Li et al. [10] studied the impact of two leakage energy reduction approaches on the cache reliability. They also used the word parity and SEC to detect and correct the corrupted bits. However, the impact of cache size selection was not considered. Clearly, the spacial redundancy requires additional hardware and decreases the performance, hence is likely to increase the cache energy consumption.

Nevertheless, like cache energy and performance, the cache reliability is also affected by the cache size. The reason for this is threefold. Firstly, when the cache size is reduced (for instance, through disabling a portion of the cache), the probability of particle-hits in the smaller active area is also reduced and particles hitting the disabled part of the cache will not manifest in errors. Secondly, the execution time of the application generally increases as the cache size decreases. The probability of particle-hits during a longer execution time increases. Thirdly, if time redundancy techniques (e.g. rollback recovery)

*This work is supported in part by the EPSRC, U.K., under grant GR/S95770, EP/C512804



(a) Cache size set to 256 lines (slack for 2 re−executions)

(b) Cache size set to 128 lines (slack for one re−execution)

(c) Cache size set to 64 lines (insufficient slack for re−execution)

Fig. 1. Affection of the cache size on reliability

are used to correct faulty executions, the cache's influence on the task execution times will also affect the number of possible re-executions, hence affecting the system's reliability. For example, in Fig. 1(a), if we reduce the cache size from 256 lines to 128 and 64 lines (Fig. 1(b) and (c)), the number of faults in the active cache area decrease from 3 to 2 and 1, respectively. However, the smaller cache sizes result in prolonged task execution times, which, in turn, change the amount of slack left for re-executions.

The aim of this paper is to examine the combined effect of cache size selection on energy consumption, reliability and performance. To the best of our knowledge, this is the first investigation into the interaction between cache energy consumption and transient faults from an cache size perspective. We perform extensive fault inject simulations on five commonly used benchmarks, using a cycle-accurate microprocessor simulator. The experiments demonstrate that there exists a complex trade-off between the different objects. Ultimately, this trade-off can be exploited through dynamic cache resizing (enabling/disabling portions of the cache) at application runtime.

The remainder of this paper is organized as follows. We introduce the models of the transient faults, the cache performability and the cache energy in Section II. The simulation setup and results analysis are in Section III. Concluding remarks are given in Section IV.

II. Preliminaries

A. Transient fault model

Transient faults within the cache are mainly caused by alpha particles hitting the flip-flops of the cache [9, 17]. The physical procedure of the particle-hits causing faults is complex and the effect depends on many factors, like the energy transferred from the particle into the circuit, the transistor size, etc. [18]. In this paper, however, we are not directly interested in the combined circuit and particle properties that can lead to transient faults, but rather in the effect of the transient faults within the cache on the task execution result with respect to performance, energy and reliability. Thus we will use the *bit-flip* as the transient fault model, i.e, when an alpha particle hits a flip-flop, the value stored in it changes its value from 1 to 0 or vice versa [19]. The arrival process of the transient faults is typically modeled as a Poisson process with an average fault rate λ_{fault} [13, 14]. With the arrival of a transient fault, each flip-flop in the cache has an equal probability to be hit since alpha particles are uniformly distributed over the circuit area. Of course, the transient faults can also occur in other parts of the processor (e.g. registers) and cause errors. However, the number of registers in modern microprocessor is far below the number of cache memory cells and it was pointed out in [21] that more than 90% of errors in a processor are originating from the transient faults in the cache. Therefore, we will focus throughout this paper on transient faults in the cache. When a transient fault occurs in the cache, though the value in a certain flip-flop is corrupted, this does not necessarily manifest in an error, that is, the computational result can still be correct. For example, when the transient fault happens in a tag array and the processor wants to read the corresponding cache line, the effect may be just one read miss causing a read from the external memory. Nevertheless, the overall correctness of the computation is not jeopardized. Even when a transient fault happens in a data array of the cache, it is possible that the corrupted value in the data array is overwritten due to a cache write from the processor or a data read from the memory to the cache before it is propagated into the data path of the processor so that the fault is masked. Hence, one important factor that characterizes the system's vulnerability is the ratio between the errors and transient faults. We define the *vulnerability factor* (VF) as this ratio. Further, we define the error rate λ_{error} as the product of VF and the fault rate, i.e., $\lambda_{error} = VF * \lambda_{fault}$. The error rate λ_{error} will be used in Section II.B to compute the performability of the cache. Though the fault rate λ_{fault} is independent of the cache size (3 faults appear in all cache configurations of Fig. 1), we can find that VF is actually a function of the cache size from the simulation results (see Section III.B), so the error rate as well as the performability both depend on the cache size. We will outline how to obtain VF through fault injection experiments in Section III.A.

B. Performability model

For real-time systems, the most important criterion of the system performance is whether the processor can finish executing a task within a given deadline. More specifically, suppose the processor needs N cycles to execute a task and the processor frequency is f, if the deadline before which the task should

complete is D, then the performance requirement is $N/f \leq D$. In the case that the task finishes execution before the deadline D, then there exists a slack. For example, Fig. 1 (b) shows the task execution for a cache size of 128 lines. As we can observe, it takes N cycles to execute the task, leaving a slack of $D - N/f$. This slack time can be utilized to increase the system's reliability against transient faults by performing roll-back recoveries (re-execution) when errors occur [14]. That is, in the presents of an error, the task is re-executed with the aim to achieve a non-faulty run. Nevertheless, since re-executions require time, the number of possible re-executions is limited by the amount of slack. Accordingly, the number of possible re-executions is given by:

$$k = \lfloor \frac{D}{N/f} \rfloor - 1 = \lfloor \frac{D \times f}{N} \rfloor - 1 \qquad (1)$$

For instance, in Fig. 1(a), the slack is large enough to perform two re-executions (k = 2), when needed. The number of possible re-execution, however, decreases as the cache size is reduced. For instance, in Fig. 1(b) and (c) the number of re-execution is $k = 1$ and $k = 0$, respectively.

Since the appearance of transient faults follows a Poisson distribution, the probability of at least one error during the execution of a task is [14]:

$$\rho_e = 1 - e^{\frac{-VF \times \lambda_{fault} \times N}{f}} = 1 - e^{\frac{-\lambda_{error} \times N}{f}} \qquad (2)$$

We use a combined metric called *performability* to measure the system performance and reliability together [13, 14]. Here, the performability is defined as the probability of finishing the task correctly within the deadline in the presence of faults [14].

Based on Eqs. (1) and 2, the performability can be expressed by [14]:

$$P = 1 - \rho_e^{k+1} = 1 - (1 - e^{\frac{-\lambda_{error} \times N}{f}})^{\lfloor \frac{D \times f}{N} \rfloor} \qquad (3)$$

The clock cycles N that the processor needs to execute a task is heavily impacted by the cache size. The direct result is that k, the number of possible re-executions, will be different for different cache sizes. Also, as outlined in Section III.A, the error rate λ_{error} is a function of the cache size. As a result, the performability is fundamentally impacted by the cache size. As our experimental results indicate (Section III), a careful selection of the cache size is of utmost importance to achieve the required performance and reliability.

C. Cache energy consumption

The energy dissipated in the cache is comprised by a static and a dynamic component. Static energy is caused by leakage currents in the CMOS circuit, while dynamic energy is mainly due to the charging and discharging of the load capacitance which are driven by switching gates. Dynamic energy consumption consists of most of the total energy dissipation in level one caches [16], hence we only consider the dynamic energy of the cache here. Cache energy is dissipated during read as well as write accesses. Write accesses include the normal write accesses and the cache line replacements after cache misses. Accordingly, the cache energy is given by:

$$E = E_{read} \times N_{read} + E_{write} \times N_{write} \qquad (4)$$

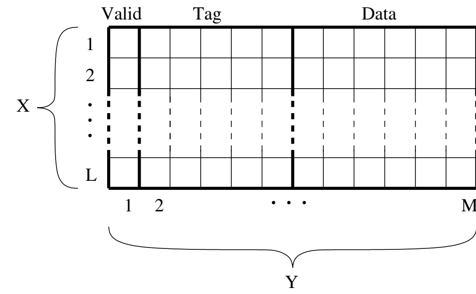


Fig. 2. Position of fault injection: instruction cache

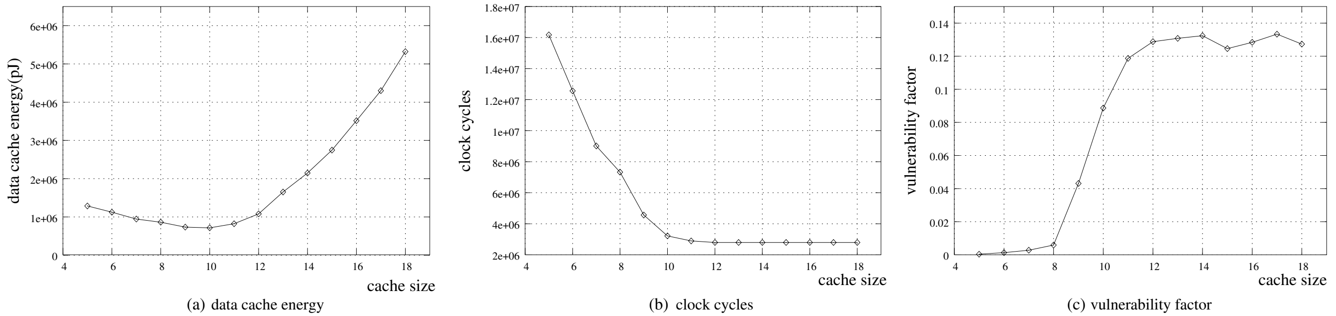
where N_{read} and N_{write} are the numbers of the cache read access and write access respectively, while E_{read} and E_{write} are the energy consumed during one cache read access and one cache write access, respectively. Both N_{read} and N_{write} depend on the cache size since the cache hit rates are generally different for different cache sizes. Furthermore, the energy per access, E_{read} as well as E_{write}, is also cache size dependent. The main reason is that different memory wire lengths have different capacitances to be charged [23].

III. Simulations and experimental results

A. Simulation setup

In order to study the impacts of the cache size on performability and cache energy, we use a simulation-based approach. The experimental platform is MPARM [24], a cycle-accurate simulator that includes an ARM7 microprocessor model. The simulator reports detailed information regarding the number of clock cycles used to execute a benchmark as well as the energy consumed within the cache. The cache size of MPARM can be configured from 32 bytes up to 1M bytes (1024$\times$1024) with the constraint that the size must be a power of 2. The cache line size is fixed to 16 bytes. In our simulation, we use separated data cache and instruction cache, and both have a maximum size of 256K bytes (256 $\times$ 1024). Cache memory accesses require one clock cycle, while accesses to the main DRAM memory are conservatively assumed to take 100 clock cycles [22]. Two way set-associativity is used for the data cache. The instruction cache is direct-mapped. Although the associativity of the cache is configurable, we do not change the cache associativity in our simulations, since our main focus is on the cache size.

We conduct the fault injection experiments to determine the vulnerability factor (Section II.A) of different cache configuration for five commonly used benchmark applications, namely, a fixed point FFT (FPFFT) with a 1024 points discrete sinewave as input, a cyclic redundancy check (CRC) with 300 ASCII characters as inputs, a 8$\times$8 matrix multiplication (MM), a 12$\times$12 matrix addition (MA) and a 100 integer quick sort algorithm (QSORT). In order to obtain accurate vulnerability factors, we inject 10^6 faults for each benchmarks. Each fault is injected in the following way: Two independent random variables X and Y are used to decide the position of the fault injection in the cache, as illustrated in Fig. 2. The random variable X is uniformly distributed between 1 and L, the number of lines in the cache. Accordingly, X decides in which line the fault will be injected. The random variable Y is uni-



(a) data cache energy (b) clock cycles (c) vulnerability factor

Fig. 3. FPFFT simulation results: data cache

formly distributed between 1 and M, the number of bits in a line, to decide which bit in the line will flip its value. This determines if the flipped bit belongs to the valid bit, the tag array or the data array. Fig. 2 shows the case for the direct-mapped instruction cache, the fault position in the two way data cache is decided in a similar way, only that Y is uniformly distributed between 1 and $2M$. When Y is less than M, the fault is within set 0; otherwise, the fault is within set 1. According to the values of X and Y, we check whether the selected memory cell falls within the disabled cache region. If so, the fault injection will definitely not cause an error and a simulation run is unnecessary. On the other hand, if a memory cell within the enabled cache is selected, then the injected fault might manifest in an error, i.e., the simulation has to be performed to observe the effect of the injected fault. Before the simulation is performed, we also randomly determine the clock cycle I of the task execution during which the fault will be injected ($1 \leq I \leq N$). After a simulation has finished, we compare the outcome with the expected outcome to see if the injected fault resulted in an error. The number of errors is counted and divided by the number of injected faults (10^6) to obtain the vulnerability factor.

B. Experimental results

In the first set of experiments, we concentrate on the data cache and the results obtained for the FPFFT benchmark. Fig. 3 shows the outcomes of the experiments. The three graphs give the energy dissipation, the number of clock cycles and the vulnerability factor as a function of the cache size. Note that the cache size in the figures is the logarithm of the true cache size, e.g., cache size 10 means the true cache size is $2^{10} = 1024$ bytes. As we can observe from Fig. 3 (a), the optimal cache energy consumption for the FPFFT benchmark is obtained for a cache size of 2^{10} bytes. It is interesting to see that the cache energy increases for smaller sizes. The main reason for this behavior is the high miss rate for smaller caches, which consequently results in a large number of cache line replacements. This increases the number of cache accesses (read/write accesses + replacement accesses). Although the energy per cache access increases when the cache size gets larger, the number of cache accesses drops relatively faster and the overall effect is a decreasing energy consumption. When the cache size is above 2^{10} bytes, the number of the cache accesses reduces slower and becomes fixed after the cache size is greater than 2^{12} bytes. However, the energy per cache access continuously increases with the cache size increasing. This is mainly due to the fact that the address/data lines in the cache become longer and hence a larger capacitance has to be charged for the accesses. As a result, the energy curve of the cache rises after the cache size is larger than 2^{10} bytes.

TABLE I
PERFORMABILITY FOR FPFFT: DATA CACHE

cache size	number of 9's	digits after 9
5	6	89742
6	6	74106
7	6	59974
8	12	52998
9	16	69592
10	26	81057
11	26	52011
12	26	39275
13	26	34412
14	26	30301
15	26	48551
16	26	40212
17	26	27900
18	26	42730

The processor clock cycles used to execute the benchmark is depicted in Fig. 3(b). As we can see, with increasing cache size the miss rate drops and the clock cycles decrease quickly. Clearly, less cache misses cause less time-consuming main memory accesses so that the total clock cycles are reduced. Nevertheless, once the cache size is above 2^{12} bytes, the clock cycles do not change any more. This is due to the fact that the application can not facilitate the extra cache and the increasing cache size will not further reduce the cache miss rate.

Fig. 3(c) shows the vulnerability factor (VF) as a function of the cache size. It shows an opposite trend when compared to the clock cycle curve of Fig. 3(b). This can be explained as follows. If the active cache portion is increasing, also an increasing number of the uniformly distributed transient faults will hit this active area, hence causing more errors. However, when the cache size has exceeded 2^{12} bytes, the FPFFT benchmark does not take advantage of additional cache and the processor will not access the additional cache lines. Hence, as the cache becomes larger and larger, although more and more transient faults fall into the cache, the number of those hitting the accessed lines of the cache does not change significantly. As a result, the VF curve becomes saturate when the cache size exceeds 2^{12} bytes.

Having obtained the clock cycles and vulnerability factor, we can now use Eq. 3 to compute the performability for each cache size. The deadline for the FPFFT benchmark is 80.9 ms, which is the execution time of the benchmark when the cache size is minimum (2^5 bytes). The performability results are given in Table I. Note that the performability is actually a probability, with a desired value of as close as possible to 1. To ease the comparison we report the results by the number of 9s after the decimal point (Column 2) and five more digits after the

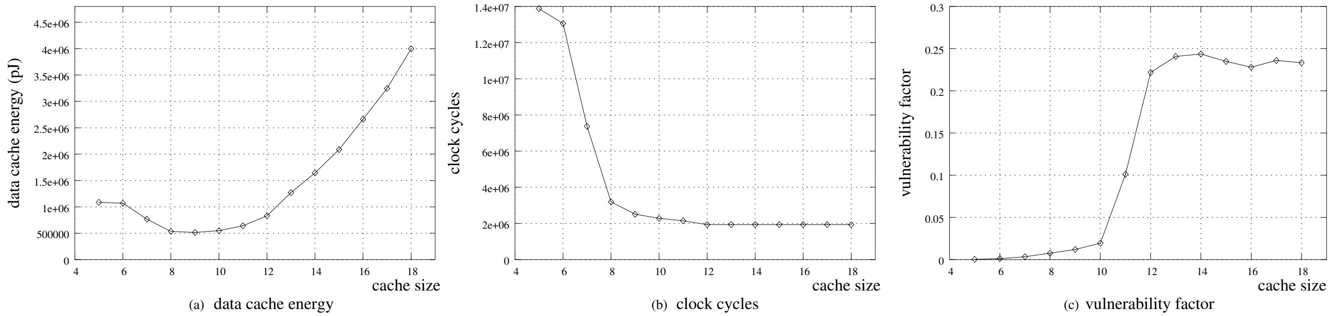


Fig. 4. CRC simulation results: data cache

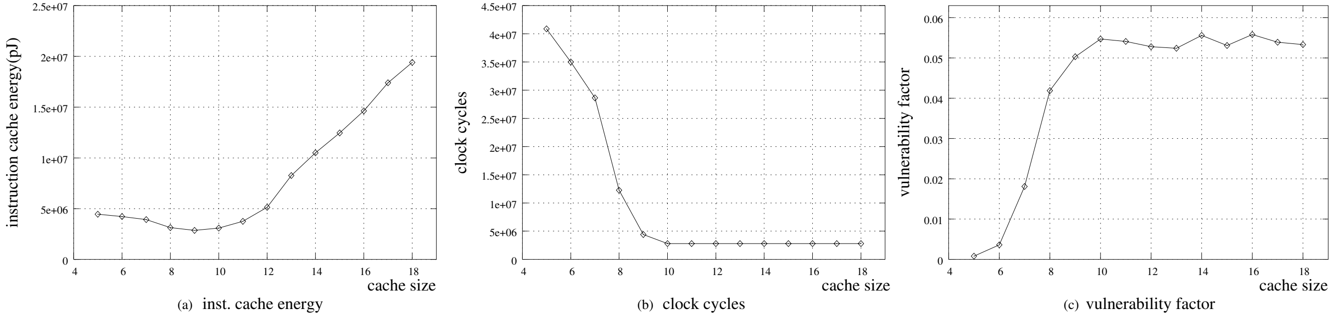


Fig. 5. FPFFT simulation results: instruction cache

last 9 (Column 3). As we can observe, when the cache size is smaller than 2^8 bytes, though the VF value is small (Fig. 3(c)), the execution time is relatively long due to the large number of clock cycles (Fig. 3(b)). As a result, there is not enough slack for the re-execution so that the performability becomes low. With increasing cache size, the execution time reduces and there is more slack left for re-executions, even though the VF value increases, the performability still improves. This illustrates that compared to VF, slack time is more important to improve the performability for this benchmark. After the size reaches 2^{10} bytes, the difference between each performability value is marginal and the number of 9s remains the same. From Table I, it can be found that with the cache size of 2^{10} bytes, we can achieve the best performability. Taking the energy curve of Fig. 3(a) into consideration, we can find that selecting a cache size larger than 2^{10} is not a good choice because it is not energy efficient and the performability will not be improved. Clearly, for the FPFFT benchmark, 2^{10} bytes is the optimal cache size with which the cache energy is minimal and the performability is maximal.

The next set of experiments is concerned with the CRC benchmark, for which we performed the same evaluations as for the FPFFT benchmark. The deadline of the CRC benchmark is at 69.4 ms. Fig. 4 and Table II show the experimental results. When comparing Figs. 3 and 4 as well as Tables I and II, we can notice similar trends, however, we should note some important difference. For CRC the cache energy is minimal when the cache size is 2^9 bytes while the performability is maximal when the cache size is 2^{10} bytes. That is, as opposed to FPFFT, there is a Pareto-optimal set of the cache sizes: $\{2^9, 2^{10}\}$. The decision of the cache size selection can be made according to the system requirement. For safety-critical systems, we should select 2^{10} bytes to achieve the highest performability; for systems with tight energy budget, e.g. some battery powered systems, the cache size should be selected as 2^9 bytes.

TABLE II
PERFORMABILITY FOR CRC: DATA CACHE

cache size	number of 9's	digits after 9
5	7	33991
6	6	75160
7	6	60276
8	25	78142
9	31	75245
10	36	87705
11	36	83265
12	36	31894
13	35	87930
14	35	86913
15	35	89884
16	35	17656
17	35	89517
18	35	03286

Due to space limitations, we do not report here the detailed results of the matrix multiplication (MM), matrix addition (MA) and quick sort algorithm (QSORT).Nevertheless, the general trends of these benchmarks follow observations made for the FPFFT and CRC benchmarks. Overall these benchmarks have optimal data cache sizes for MM, MA and QSORT of 2^{10}, 2^9 and 2^9, respectively.

Since the above given experimental results concentrated on the data cache, we have conducted additional experiments for the instruction cache. The results of the FPFFT benchmark are given in Fig. 5 and Table III. The energy curve reaches the lowest point at the cache size of 2^9 bytes. In Table III, we can find the performability is maximum when the cache size is 2^{13} bytes. However, since the performability at size 2^{10} is very close to the maximum value and the number of 9s at size 2^{10} is the same as that of size 2^{13}, we can choose 2^9 and 2^{10} as the Pareto-optimal set of the cache size.

The experimental results for the other four benchmarks on the instruction cache follow a similar trend and details are omitted due to space limitations. Nevertheless, for the MA and QSORT benchmarks, there are Pareto-optimal sets: $\{2^7$,

TABLE III
PERFORMABILITY FOR FPFFT: INSTRUCTION CACHE

cache size	number of 9's	digits after 9
5	5	48157
6	4	80023
7	4	17800
8	12	45910
9	40	19101
10	64	75657
11	64	79140
12	64	85160
13	64	86659
14	64	69409
15	64	83935
16	64	67832
17	64	80194
18	64	83067

$2^8\}$ and $\{2^8, 2^9\}$, respectively. For the MM and CRC benchmarks the optimal cache sizes are 2^8 and 2^9, respectively.

Summarizing, we can draw the following conclusion from the above experimental results. There exist optimal or Pareto-optimal cache size choices with respect to performability and energy consumption. Depending on the application requirement a proper cache size should be selected to achieve the optimal energy and performability simultaneously or the best trade off. Furthermore, as the optimal cache sizes depend largely on the running application, dynamically changing the cache size to suit the particular application is not only beneficial from an energy point of view but also to improve the system's performability. For instance, when running the FPFFT benchmark after the MA benchmark, the processor's data cache should be adapted from 2^9 to 2^{10} bytes and the instruction cache should be changed from 2^8 to 2^9 bytes. This adaption would reduce the data cache energy by 2.6% and increasing the performability from sixteen 9s to twenty-six 9s. The instruction cache would reduce its energy by 8.8% and improve the performability from twelve 9s to forty 9s.

IV. CONCLUSIONS

In this paper, we studied the impact of the cache size selection on three important design objectives, namely, the system performance, the cache energy consumption and the cache reliability, which has not been addressed explicitly in previous work. Performability has been defined to combine the analysis of the performance and the reliability. We have conducted extensive experiments to analyze the interplay between the three objects. These experiments were performed using cycle-accurate processor simulations and it was found that the cache size selection affects not only the energy but also the performability. The results indicate that a careful cache size selection is needed, in order to take advantage of the found optimal energy/performability trade-off points.

REFERENCES

[1] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd Edition, Morgan Kaufmann Publishing Co. 1996

[2] C. Zhang, F. Vahid and R. Lysecky, "A self-Tuning Cache Architecture for Embedded Systems", in Proc. of DATE, 2004.

[3] C.Zhang, F. Vahid, W. Najjar, "A Highly Configurable Cache Architecture for Embedded Systems", in Proc. of International Symposium on Computer Architecture, 2003.

[4] S. Dropsho et al., "Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power", in Proc. of the International Conference on Parallel Architectures and Compilation Techniques, 2002.

[5] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation", in Proc. of International Symposium on Microarchitecutre, 1999.

[6] M. Powell A. Agaewal, T. Vijaykumar, B. Falsafi and K. Roy, "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct Mapping", in Proc. of International Symposium on Microarchitecture, 2001.

[7] A. C. Nacul and T. Givargis, "Dynamic Voltage and Cache Reconfiguration for Low Power", in Proc. of DATE 04, March, 2004.

[8] S. Yang, M. D. Powell, B. Falsafi, T. N. Vijaykumar, "Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay", in Proc. of International Symposium on High-Performance Computer Architecture, 2002.

[9] G. Asadi, V. Sridharan, M. B. Tahoori, D. Kaeli, "Balancing Performacne and Reliability in the Memory Hierarchy", in Proc. of International Symposium on Performance Analysis of Systems and Software, 2005.

[10] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, "Soft Error and Energy Consumption interations: A Data Cache Perspective", in Proc. of ISLPED 04, Aug. 2004.

[11] S. Kim and A. K. Somani, "Area Efficient Architectures for Information Integrity in Cache Memories", in Proc. of International Symposium on Computer Architecture, 1999.

[12] W. Zhang, S. Gurumurthi, M. Kandemir, A. Sivasubramaniam, "ICR: in-cache replication for enhancing data cache reliability", in Proc. of International Conference on Dependable Systems and Networks, 2003.

[13] D. Zhu, R. Melhem and D. Mosse, "The Effecs of Energy Management on Reliability in Real-Time Embedded Systems", in Proc. of ICCAD 04, Nov. 2004.

[14] A. Ejlali, M. T. Schmitz, B. M. Al-Hashimi, S. G. Miremadi, "Energy Efficient SEU-Tolerance in DVS-Enabled Real-Time Systems through Information Redundancy", in Proc. of ISLPED 05, Aug. 2005

[15] R. Melhem, D. Mosse, E. Elnozahy, "The interplay of Power Management and Fault Recovery in Real-Time Systems", IEEE Transaction on Computers, Vol. 53, No. 2, February, 2004.

[16] H. Hanson, M. S. Hrishikesh, V. Agarwal, S. W. Keckler, D. Burger, "Static Energy Reduction Techniques for Microprocessor Caches", IEEE Transaction on VLSI systems, Vol. 11, No. 3, June, 2003.

[17] S. Mitra, N. Seifert, M. Zhang, Q. Shi, K. S. Kim, "Robust System Design with Built-In Soft-Error Resilience", IEEE Computer Magazine, Vol, 38, No. 2, Feburary, 2005.

[18] P. E. Dodd and F. W. Sexton, "Critical Charge Concepts for the CMOS SRAMS", IEEE Transactions on Nuclear Science, Vol. 42, No. 6, Dec. 1995.

[19] F. Faure, R. Velazco, M. Violante, M. Rebaudengo and M. Sonza Reorda, "Impact of Data Cache Memory on the Single Event Upset-Induced Error Rate of Microprocessors", IEEE Transactions on Nuclear Science, Vol. 50, No. 6, Dec. 2003.

[20] A. Maheshwari, W. Burleson, R. Tessier, "Trading off Transient Fault Tolerance and Power Consumption in Deep Submicron (DSM) VLSI Circuits", IEEE Transaction on VLSI systems, Vol. 12, No. 3, March 2004.

[21] M. Rebaudengo, M. S. Reorda and M. Violante, "An Accurate Analysis of the Effects of Soft Errors in the Instruction and Data Caches of a Pipelined Microprocessor", in Proc. of DATE 03, March, 2003.

[22] L. Li, I. Kadayif, Y-F. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin and A. Sivsubramaniam, "Leakage Energy Management in Cache Hierarchies", in Proc. of International Conference on Parallel Architectures and Compilation Techniques, 2002.

[23] G. Reinmann and N. P. Jouppi, "CACTI2.0: An Integrated Cache Timing and Power Model", COMPAQ, Western Research Lab, Research Report, 2000.

[24] http://www-micrel.deis.unibo.it/sitonew/research/mparm.html