Application-Specific Memory Protection Policies for Energy-Efficient Reliable Design

Sheng Yang[†], Rishad A. Shafik[†], Saqib Khursheed[‡], David Flynn[⋆], Geoff V. Merrett[†] & Bashir M. Al-Hashimi[†]
[†]School of ECS, Uni. Southampton, UK [‡]Dept. of EEE, Uni. Liverpool, UK [⋆]ARM, Cambridge

***Abstract*—In this paper, we show that the vulnerability of memory components due to data retention in the presence of soft errors exhibit orders of magnitude variations with applications through extensive analysis of MiBench benchmarks. Underpinning such analysis, we propose a novel application-specific design flow for joint energy efficiency and reliability optimization. The energy efficiency is achieved through voltage/frequency scaling (VFS), while reliability is achieved through suitably choosing the appropriate protection policies (L1-Cache resizing and selective ECC) for hierarchical memory components. Fundamental to such joint optimization is a design analysis framework, which can analyze trade-off between memory protection policies considering the impact of VFS, and apply design optimization algorithm to provide with an energy-efficient design, while meeting a given reliability target. Using this framework the proposed design flow is validated through extensive number of application case studies based on ARMv7 processors modeled in GEM5. We show that the joint consideration of cache resizing and VFS can improve the L1-Cache reliability by up to 5x compared to VFS alone, while incurring <10% energy overhead. Additionally, using selective ECC for L2-Cache and DRAM, we show that energy consumption can be reduced by up to 40%.**

I. Introduction

Memory systems reliability is critical for the correct operation of processors as memory constitutes a significant proportion of modern embedded systems. However, with continued technology scaling, different memory components in these systems are increasingly becoming more susceptible to soft errors, such as single-event upsets (SEUs) [1]. These errors manifest themselves as perturbation of signal transfers and corruption of stored values leading to incorrect executions in embedded systems. Operating reliably in the presence of these errors is highly challenging, particularly for high availability or safety-critical applications [2].

Figure 1 shows a typical memory hierarchy in a system-on-chip (SoC). As can be seen, smaller and high-performance registers are located in the processor core at the top of hierarchy that execute instructions with their operands. Due to their direct performance impact, the protection of these registers in the presence of SEUs is generally carried out through simpler architectural duplication techniques, such as duplication between active and unused registers [3] and 64-bit registers to store duplicated 32-bit values [5].

To store computation data at high-speed the processor core is directly interfaced with a static random access memory (SRAM) based Level-1 (L1) caches, such as data and instruction caches (Figure 1). Similar to registers, these memories are expensive and small in size. Hence, to protect L1-Cache memories, low-latency and effective methods, such as cache resizing [6], low complexity parity caching [7, 8] and cache duplication [10] have been proposed. Next in the memory hierarchy, Level-2 (L2) caches are connected to L1-Cache, which are slightly lower performance with higher capacity. Due to less implications on processor performance these caches are protected using various coding techniques, such as multi-bit parity coding [11] and memory mapped ECC [4].

Further down the hierarchy dynamic random access memories (DRAMs) are used as main memories (Figure 1). These are large and high-latency memories. Information redundancy using error correction coding (ECC) is a popular memory protection method

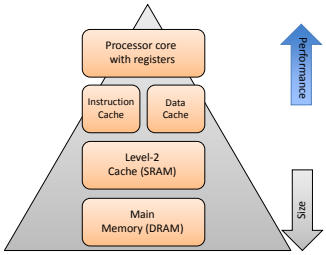


Fig. 1. Microprocessor memory hierarchy

for DRAMs. However, such coding is usually associated with overheads in terms of area and energy consumption. The overhead increases with higher error correction capabilities, depending on the coding word size. For example, increasing the ECC capability of a 64b-word from single error correction double error detection (SECDED) to double error correction triple error detection (DECTED) increases the area overheads from 15% to 25% and increases energy overheads from 25% to 55% [11].

Existing memory protection methods for various components in the memory hierarchy (Figure 1), such as [3, 8, 10–13], have the following two limitations. Firstly, these methods address the reliability improvement of a given memory component without considering the system-wide impact of SEUs. Due to lack of such system-wide insights, these methods cannot guarantee protection of different components at low-cost due to conflicting design trade-offs between memory components. Secondly, existing methods are application-agnostic, i.e. they do not consider the impact of application on the data retention-related vulnerabilities. To address these limitations, we make the following *contributions*:

- a holistic memory vulnerability analysis showing significant vulnerability variation, which depends on the application,
- based on the analysis a novel application-specific design flow is proposed for suitably optimizing VFS and memory protection policies, while minimizing energy for a given system-wide reliability target, and
- a prototype design and analysis framework implementing the proposed design flow, which is validated through Gem5-based extensive simulations.

To the best of our knowledge, this is the first complete energy-efficient design flow based on a holistic memory analysis. The rest of the paper is organized as follows. Section II and Section III provide memory vulnerability analysis. Based on such analysis, Section IV proposes a energy-efficient and reliable design flow. Finally, Section VI concludes the paper.

II. Memory Vulnerability Analysis

To set up the motivation of this work, memory vulnerability model in the presence of SEUs and its analysis are detailed.

A. Memory Vulnerability Model

Architectural reliability is usually expressed using failures in time (FIT), which defines the total number of SEUs experienced by an architectural component during one billion hours of operation. The FIT of an architectural component depends on the fabrication process and is influenced by the operating environment. Using

FIT as the architectural fault rate (λ_{FIT}), the rate at which an unprotected memory experiences faults per hour can be estimated using the following equation [15]:

$$\lambda_{mem}^{'} = \frac{\lambda_{FIT}}{10^9} \approx \lambda_{bits} \times N \quad , \tag{1}$$

where N is the size of the memory (in bits) and λ_{bits} is the

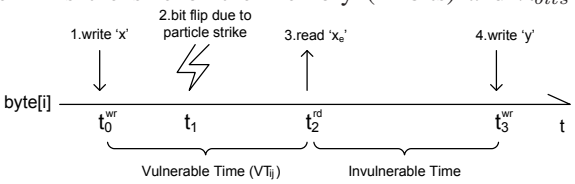


Fig. 2. Vulnerability of memory cells due to SEUs

logic-level fault rate in terms of the number of faults per cell per hour. Equation (1) gives the worst-case memory system fault rate estimation, generally used for over-engineering in safety-critical systems. In such systems, it is assumed that a system will fail if any storage node is corrupted. However, such assumption is pessimistic in other systems as some memory cells are not used during application runtime. Moreover, even if data corruption occurs in the cells within the application, the cells may be over-written before the fault actually takes place. To demonstrate this, Figure 2 shows the data retention lifetime of one byte of storage cells (8 bits). At time t_0, data 'x' is written into the storage cells. At time t_1, a particle strike causes a bit-flip corrupting the stored data. At time t_2, the corrupted data 'x_e' is read from the storage cells and propagated to the processor core or other memory components, which may lead to erroneous output. Therefore the data lifetime between t_0 and t_2 is susceptible to corruption; we refer this as the vulnerable time (VT). At time t_3 data 'y' are written into the same storage cells and mask data corruption. Hence time between t_2 and t_3 is not susceptible to corruption; we refer this as invulnerable time (IVT). From Figure 2, the effective vulnerable time of i-th byte in the j-th vulnerable storage node (VT_{ij}) can be expressed as

$$VT_{ij} = t_{ij}^{rd} - t_{ij}^{wr} \quad , \tag{2}$$

where t_{ij}^{rd} is the time of the last read operation of i-th byte storage in the j-th storage node and t_{ij}^{wr} is the time of the write operation of i-th byte storage in the j-th storage node. The total vulnerable time (VT) is the sum of the VT_{ij} of each storage cell:

$$VT = \sum_i \sum_j VT_{ij} \quad . \tag{3}$$

Average vulnerable storage (N_{vuln}) during the application runtime is calculated by dividing the total vulnerable time by the application runtime:

$$N_{vuln} = \frac{VT}{T_{ex}} \quad , \tag{4}$$

where T_{ex} is the runtime of the application. Therefore the N_{vuln} is With the given equivalent vulnerable storage size in (4), a more realistic estimation of memory error rate (λ_{mem}) following (1) can be expressed as

$$\lambda_{mem} = \lambda_{bits} \times N_{vuln} \quad . \tag{5}$$

Due to different patterns of memory accesses at various hierarchical levels, N_{vuln} is application-dependent, which is discussed further in Section II-C.

B. Analysis Framework

To facilitate a holistic reliability analysis of different components in the memory system hierarchy (Figure 1), a prototype reliability, performance and energy analysis (RPEA) framework is developed in Gem5 [16] as shown in Figure 3. The inputs to this framework consist of system configuration files and benchmark applications. With the given inputs, the RPEA framework carries out performance analysis through the built-in GEM5 generated performance statistics. To analyze the system energy consumption, McPAT [17] tool is used.

To generate reliability statistics from such simulations, vulnerable storage monitors were incorporated in GEM5 with to calculate N_{vuln} (given by (4)) during read and write accesses in each storage unit. Figure 4 shows the memory system architecture and read/write monitors introduced in GEM5 for analyzing the access information of each memory component and calculating the vulnerable storage. As can be seen, monitors are introduced in all read and write ports. These monitors are used to estimate the data retention related vulnerabilities (see Section II-C).

To accelerate the simulation speed, the simulations are executed on the Iridis3 super-computing cluster (https://cmg.soton.ac.uk/iridis) with parallel workloads distributed among its nodes. A python script is used to setup the analysis framework globally and initiate simulations. In this work, the design space consists of 24 configurations (including memory sizes and different operating voltages), each with 24 benchmark applications, leading to a total of $24 \times 24 = 576$ simulations running simultaneously.

C. Vulnerability Analysis

In Section II-A we have shown that memory system reliability depends on bit error rate (λ_{bit}) and vulnerable storage (N_{vuln}). Assuming the operating environment does not change during the system runtime, reliability of the memory components with a given process library is proportional to N_{vuln}. Figure 5 shows the N_{vuln} of memory components measured across various benchmark applications. It can be seen that the reliability of each memory component varies with application. For the Instruction Cache (I-Cache), the highest N_{vuln} is 1.9×10^5 bits in the case of "gsm_toast" and the lowest N_{vuln} is 1.1×10^4 bits in the case of "dadpcm". For the Data Cache (D-Cache), the N_{vuln} ranges from 2.4×10^5 bits in the case of "susan_smoothing" to $6,500$ bits in the case of "bitcount". For the L2-Cache, the highest N_{vuln} is caused by "patricia" and the lowest N_{vuln} is caused by "cadpcm". Similarly for the DRAM, the lowest N_{vuln} is caused by "typeset" and the highest N_{vuln} is caused by "stringsearch". These variations arise due to nature of computation carried out by the different applications and the different in memory usage and memory access patterns; memory intensive applications generally have higher N_{vuln} than computation intensive applications.

Figure 6 compares the statistical mean of the different N_{vuln} of memory components, along with their upper and lower bounds.

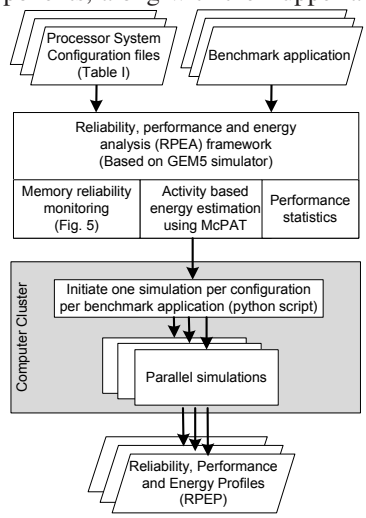


Fig. 3. Memory system analysis framework

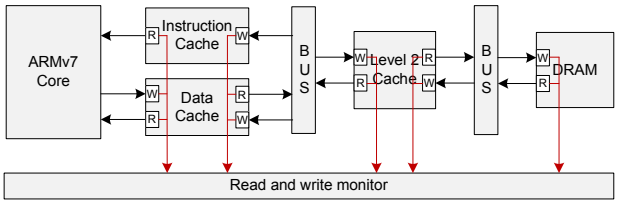


Fig. 4. Memory access monitors in GEM5

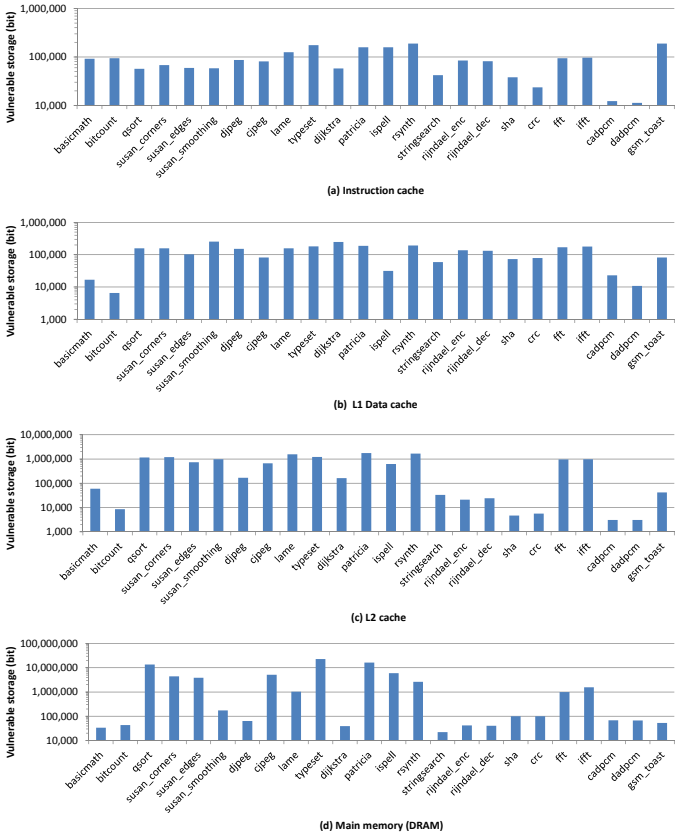


Fig. 5. Vulnerable storage of different memory components for different MiBench benchmark applications: (a) L1 instruction cache (I-Cache), (b) L1 data cache (D-Cache), (c) L2-Cache and (d) DRAM

The bar shows average N_{vuln}, and the error line shows the range between minimum and maximum N_{vuln} for a given memory component. As can be seen, I-Cache has the lowest average and worst-case N_{vuln}, while main memory (DRAM) has the highest average and worst-case N_{vuln}. The spread between the highest and the lowest N_{vuln} is around one order of magnitude for L1-Cache and about three orders of magnitude for L2-Cache and DRAM. Comparing the worst-case reliabilities, DRAM is the least reliable memory component with the worst-case N_{vuln} of 2.3×10^7 bits, followed by L2-Cache with the worst-case N_{vuln} of 1.8×10^6 bits (a difference of up to thirteen times). This indicates that L2-Cache and DRAM are more susceptible to failures in the presence of soft errors. The following two observations are made: *Observation 1*: The vulnerability of a component in the memory hierarchy in terms of vulnerable storage (N_{vuln}) is application-specific. Memory

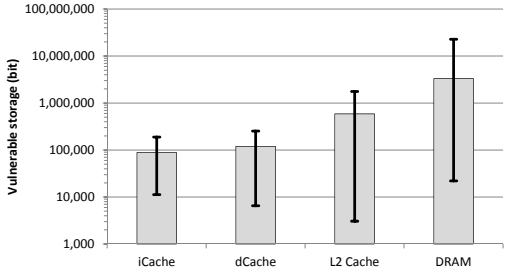


Fig. 6. Vulnerability of memory components across applications

intensive applications typically exhibit higher vulnerability, while computationally intensive applications with less memory access show less vulnerability. For the given benchmark applications (Figures 5 and 6), L1-Cache shows up to one order of magnitude variations, while L2-Cache and DRAM show up to three orders of magnitude variations.

Observation 2: For a given application, the vulnerabilities between components in the memory hierarchy also show variations. Due to its size, DRAM is invariably the most vulnerable component in the memory hierarchy, followed by L2-Cache and L1-Cache. The variations can be significant depending the application; for example vulnerability of DRAM is higher by up to 13x compared to that of L2-Cache for memory intensive "stringsearch" application.

From Observation 1, it is evident that the protection of a given memory component needs to consider application-specific impact on its vulnerability. However, to achieve a target overall reliability of an application it is important that such protection is carried out for all components in the memory hierarchy considering the relative vulnerabilities of memory components and their various design trade-offs (Observation 2). The following sections further investigates into the impact of low-power and reliable design considerations on the application-specific vulnerabilities.

III. Energy-Efficient Memory Protection Policies

In this section, the energy-efficient and reliable policies: VFS, memory sizes and their protection polices, are detailed.

TABLE I
VOLTAGE AND FREQUENCY SCALING

	1.2V	1V	0.85V	0.75V
Processor core clock [19]	1	0.5	0.25	0.125
Bit error rate [20]	1	1.7	2.56	3.34
Dynamic power [19]	1	0.347	0.126	0.049
Leakage power [21]	1	0.532	0.328	0.235

A. Voltage/Frequency Scaling

Voltage and frequency scaling (VFS) is an effective low-power design technique, widely used in modern processors. In this work, TSMC low power 65nm technology library is used as reference design library. Table I shows the corresponding normalized processor core operating frequencies, soft error rates, dynamic and leakage power consumptions at each supply voltage point. The normalization is carried using the same values at nominal supply voltage of 1.2-V. An empirical model based on the measurements from test chips [9] is used to estimate the relationship between delay and supply voltage. The corresponding soft error rates are found out using the relationships in [20]. The leakage power scaling ratio is technology-dependent [19]; therefore its calculation needs an empirical model based on a test chip as shown in [9].

To enable multiple supply voltages in the SoC, it is divided into two power domains: processor core and L1-Cache, including instruction and data caches, are located in power domain *PD1*, while the L2-Cache is placed in a separate power domain *PD2*. DRAM is normally off-chip but placed in *PD2* for simplicity. In this work, VFS is only applied to PD1 as a usual practice; PD2 scaling is usually costly in terms of performance of L2-Cache and DRAM.

B. Memory Protection Policies

The choice of memory protection policy depends on the hierarchical organization of memory components [6]. For most memory components, ECC is an effective protection scheme. However, it is not suitable for L1-Cache protection because it is the most performance sensitive memory component; moreover, the energy and performance overheads incurred due to ECC protection can render diminishing returns. As L1-Cache is the least unreliable

component in memory hierarchy (Figure 6), simpler architectural design choices, such as cache resizing [6] are made for its protection. In this work, L1-Cache resizing is proposed for protection of L1-Caches, while ECC is chosen for L2-Cache and DRAM. The protection policies is also depending on the application, where the protection of each memory component can be switched on and off. The impact of these protection policies is investigated next.

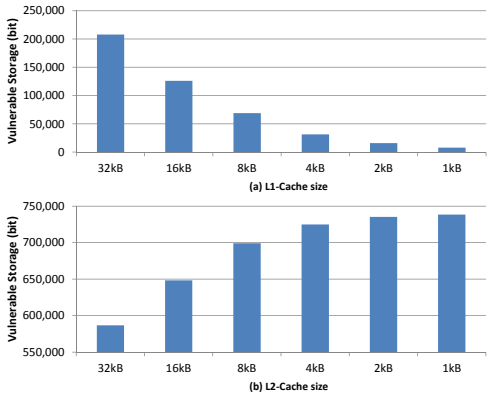


Fig. 7. The impact of L1-Cache resizing on (a) L1-Cache and (b) L2-Cache vulnerabilities

1) L1-Cache Resizing: To investigate the impact of L1-Cache resizing on application vulnerability (N_{vuln}), Figure 7 shows the N_{vuln} of L1-Cache and L2-Cache. As can be seen, L1-Cache N_{vuln} reduces almost linearly with L1-Cache size. For example, when L1-Cache size reduces from 32kB to 2kB, N_{vuln} reduces from 210kbits to 8kbits (21$\times$ reduction in N_{vuln} for 16$\times$ lowered size, Figure 7.(a)). However, the reduced N_{vuln} in L1-Cache due to L1-Cache resizing causes an increase in the L2-Cache N_{vuln}. This is because L1-Cache resizing moves vulnerable storage (N_{vuln}) from L1-Cache to L2-Cache and makes L2-Cache less reliable. Such increase in L2-Cache N_{vuln} is, however, less than the reduction in L1-Cache N_{vuln}. This can be explained using Figure 8. A cache line is marked as *clean* when the stored data has not been modified and a cache line is marked as *dirty* when the data has been modified by the processor core. Figure 8.(a) shows vulnerable time of clean data is duplicated on both L1- and L2-Caches. When the cache line needs to be replaced, it simply evacuates the data from L1-Cache. Therefore the L1-Cache vulnerable time is reduced and the L2-Cache vulnerable time stays the same. Figure 8.(b) shows that dirty cache line replacement triggers a write-back, which moves the vulnerable time from L1-Cache to L2-Cache. Therefore the reduction of L1-Cache vulnerable time increases the vulnerable time of L2-Cache. The impact of L1-Cache resizing on energy, reliability and performance are discussed in Section V-B.

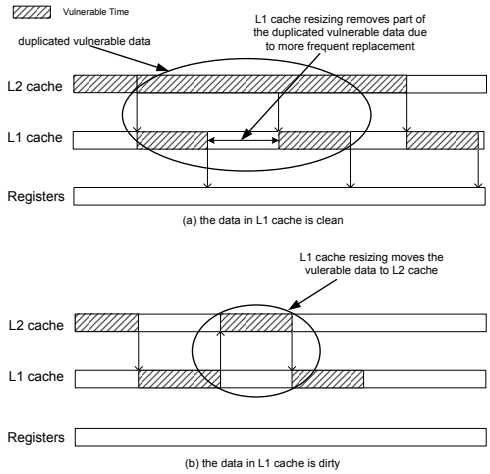


Fig. 8. Vulnerability under L1-Cache resizing when L1-Cache data are (a) clean and (b) dirty

2) L2-Cache and DRAM ECC Protection: Due to less energy and performance impact per bit protection, ECC is used to protect L2-Cache and DRAM in the presence of soft errors. Since DRAM is the less reliable than L2-Cache (Section II-C), it requires stronger ECC protection than L2-Cache. In this work, Double Error Correction and Triple Error Detection (DECTED) code is considered for DRAM; while Single Error Correction and Double Error Detection (SECDED) code is employed for L2-Cache. The impact of such ECC protection is studied in [11].

IV. Proposed Design Flow

Based on these analysis in Sections II and III, an application-specific low-cost reliable design flow is proposed. Figure 9 shows the proposed energy-efficient reliable design flow together with the conventional design flow (right) [22]. As can be seen, the conventional design flow is organized in three stages: design, implementation and runtime. In the design stage, the processor architecture, memory resources and interconnects are configured based on the specifications, which leads to a hardware prototype design. The system design is then synthesized and integrated in the implementation stage, which generates the actual hardware netlist and layout. Finally, in the runtime stage the software programs are loaded into the hardware system for execution.

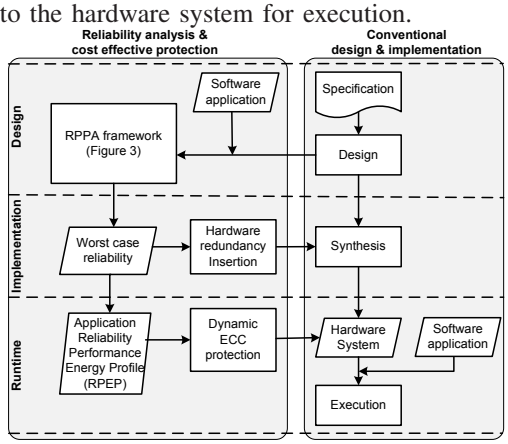


Fig. 9. Proposed reliable design flow

The proposed design flow is integrated with the conventional design flow and is shown on the left-hand side of Figure 9. Similar to the conventional design flow, the proposed design flow is organized and integrated in the design, implementation and runtime stages. During the design stage, dynamic timing constraint (DTC, in seconds), dynamic reliability constraint (DRC, in FITs), the prototype hardware design of the system and the target application software are inputs to Reliability, Performance and Energy Analysis (RPEA) framework (Figure 3). Given the constraints, the framework generates an application-specific worst-case reliability metrics of the components in the memory hierarchy using vulnerability analysis (Section II-C). For each memory component, if the worst-case vulnerability (given by (4) and (5)) is lower than the requirement, protection policies (such as L1-Cache resizing and selective ECC protection) for these components are incorporated during the implementation stage. The worst-case vulnerability analysis in the RPEA framework also generates application Reliability, Performance and Energy Profile (RPEP), which can be used to guide the energy-efficient reliability optimization before runtime.

Application specific memory protection policy is achieved through turning on/off the protection for each memory component depending on what application the system is running. Algorithm 1

Algorithm 1 Time- and reliability-constrained optimization for low-cost reliable design

Require: Reliability, performance and Energy profile (RPEP) with dynamic timing constraint (DTC) and dynamic reliability constraint (DRC)
Ensure: Energy(E), Minimum energy (E_{min}), Supply voltage (V_{dd}), L1-Cache size (S_{L1}), L2 ECC enable ($EccEN_{L2}$), DRAM ECC enable ($EccEN_{DRAM}$)

1: Initialize: V_{dd} = nominal supply voltage
2: **while** processor passes DTC check **do**
3: **while** L1-Cache fails DRC check **do**
4: reduce L1-Cache size {Ensures L1-Cache satisfies DRC}
5: **end while**
6: **if** L2-Cache fails DRC check **then**
7: $EccEN_{L2}$ = 1
8: $E = E + E^{L2}_{OVH}$
9: **end if**
10: **if** DRAM fails DRC check **then**
11: $EccEN_{DRAM}$ = 1
12: $E = E + E^{DRAM}_{OVH}$
13: **end if**
14: **if** Energy $< E_{min}$ AND processor passes DTC check **then**
15: $E_{min} = E$
16: $S_{L1} = S_{L1}$
17: $V_{dd} = V_{dd}$
18: **end if**
19: reduce supply voltage
20: **end while**
21: **return** cSize, V_{dd}, $EccEN_{L2}$, $EccEN_{DRAM}$

shows the reliability and performance constrained low-cost reliability optimization algorithm by using dynamic memory protection, cache resizing and VFS control. As can be seen the inputs are: reliability, performance and energy profile (RPEP), dynamic timing constrains (DTC) and dynamic reliability constrains (DRC). It is assumed that the DTC and DRC does not exceed the design time constraints. The RPPP is generated by the analysis framework (Figure 3) for each application under different supply voltages and L1-Cache sizes. The algorithm begins by setting the supply voltage to nominal supply voltage (1.2V), and checks whether the processor meets the DTC and DRC (lines 1-2). The reliability of L1-Cache is checked next and the L1-Cache size (S_{L1}) is reduced until it passes the reliability constraint (lines 3-5). This is then followed by the DRC check of L2-Cache and DRAM. If they fail, dynamic ECC protection is enabled by setting $EccEn_{L2}$ and $EccEn_{DRAM}$ to 1 for L2-Cache and DRAM respectively (lines 6-13)). When ECC is enabled for these memory components, their corresponding ECC energy overheads are also added to the overall energy (E). The memory protection techniques are carried out iteratively for reduced VFS settings (lines 2-19). The system configuration for which the energy is minimum (E_{min}) in each iteration, while meeting the DTC and DRC constraints, is saved before the next iteration. The design configuration with the lowest energy is returned as the low-cost reliable design. As the algorithm iterates through all configurations, the complexity is O(n), where n is the number of configurations.

V. Experimental Results and Case Studies

In this section, first the impact of VFS and memory protection policies on energy and reliability trade-offs is validated, followed by experimental results of joint optimization using the proposed design flow (Section IV).

A. Impact of VFS

Figure 10 shows the impact of VFS on performance, power, energy and reliability trade-offs. The results are normalized to the nominal V_{dd} of 1.2V. As can be seen, the performance measured in Instructions Per Cycle (IPC) increases marginally with VFS. However, the power consumption reduces with V_{dd} scaling. When V_{dd} is reduced to 1V, the power consumption is reduced to 0.4; further scaling reduces it to 0.17 for 0.85V and to 0.1 for 0.75V. Energy consumption also reduces with V_{dd}. Note that the minimum energy consumption takes place at supply voltage of 0.85V. The energy consumption at V_{dd} of

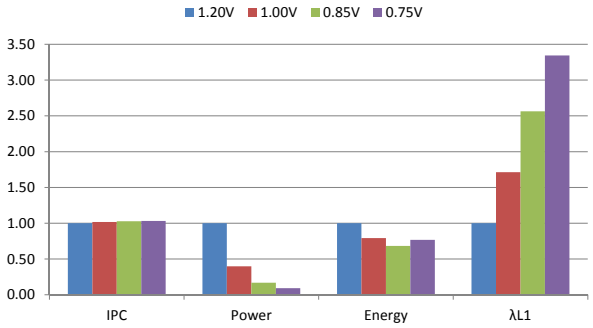


Fig. 10. The impact of VFS on performance, power, energy and L1-Cache reliability, normalized to nominal supply voltage of 1.2V

0.75V increases as the power consumed by PD2 starts to dominate. Without VFS on PD2, the power consumption of PD2 is almost the same, but the runtime increases when VFS is applied on PD1, which leads to the increase energy consumption of PD2. At 0.75V the increase energy consumption of PD2 surpass the energy reduction of PD1, which leads to the increase in overall energy consumption. Reduced V_{dd} also causes reliability problems for L1-Caches as it increases the error rates in L1-Cache (λL1). From Figure 10 it is evident that low power design using VFS achieves power and energy reduction at the expense of L1-Cache reliability degradation. To achieve energy-efficient reliable design, memory protection policies (cache resizing and ECC) need to be suitably incorporated, this is investigated next.

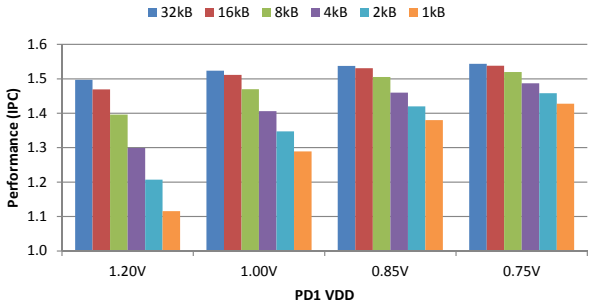


Fig. 11. The impact of L1-Cache resizing on performance under VFS

B. Impact of L1-Cache Resizing

L1-Cache resizing affects performance of the application because reducing the L1-Cache size increases L1-Cache misses. Figure 11 shows that L1-Cache resizing has a significant impact on performance under nominal supply voltage (1.2V), but VFS on the processor core (including L1-Cache) reduces this performance impact. At a 1.2V nominal supply voltage with a 1GHz clock, 1ns L1-Cache latency and 8ns L2-Cache latency, the processor core needs to wait for 8 clock cycles on an average for each L1-Cache miss. When the supply voltage of the processor core (including L1-Cache) is reduced to 0.85V with a 250-MHz clock, the L1-Cache latency increases to 4ns and L2-Cache latency stays at 8ns. Each L1-Cache miss requires only two clock cycles. Therefore a smaller L1-Cache has less performance loss at lowered VFS.

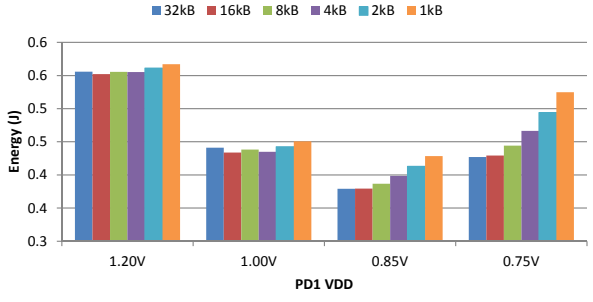


Fig. 12. The impact of L1-Cache resizing on energy under VFS

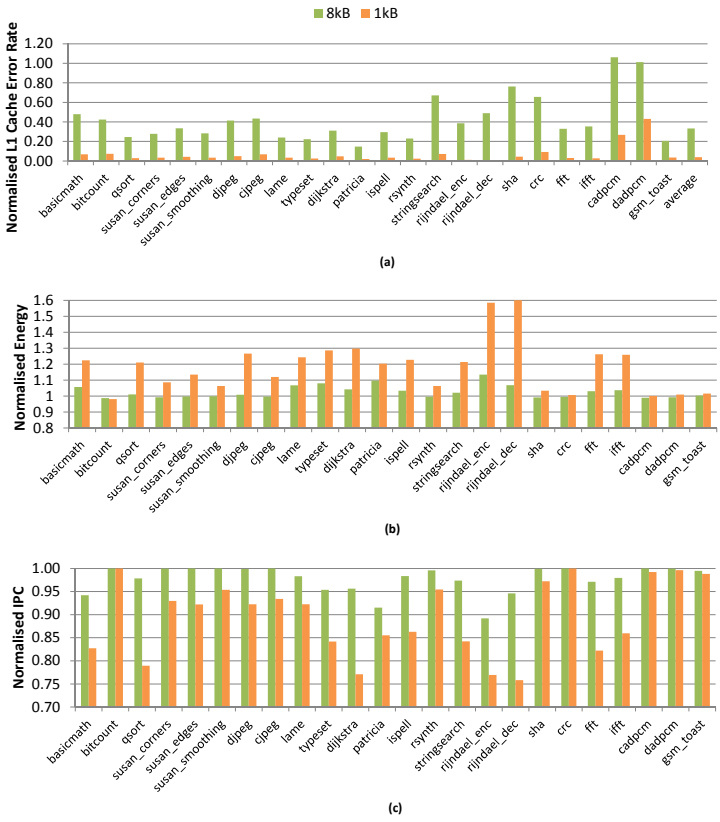


Fig. 13. The impact of L1-Cache resizing on (a) L1-Cache reliability (b) performance and (c) energy; under supply voltage of 0.85V

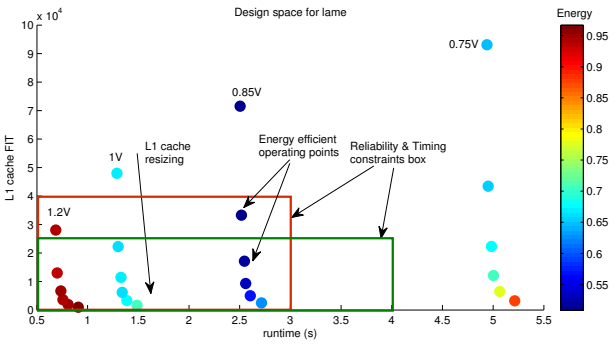
L1-Cache resizing also affects the energy consumption. Reducing the cache size means shutting down some cache lines, which reduces power consumption. However, due to increased latency (see Figure 11) the application runtime is increased. Figure 12 shows the effect of L1-Cache resizing on normalized energy consumption averaged across benchmark applications. As can be seen, under higher supply voltage cache resizing have smaller impact on energy consumption. This is because under higher supply voltage, cache consumes a significant amount of power which offsets the energy cost due to the increase in runtime. When VFS is applied the reduction is supply voltage reduces energy consumption. However, when L1-Cache size is reduced the energy consumption increases due to performance degradation.

Figure 13 shows the impact of L1-Cache resizing on reliability, performance and energy for different benchmark applications at a 0.85V supply voltage, which is the most energy efficient operating voltage (Figure 12). Figure 13.(a) shows the effect of L1-Cache resizing on L1-Cache reliability. As can be seen, when the L1-Cache size is reduced from 32kB to 8kB, there is up to a 7× error rate reduction in the case of "patrica", and the average error reduction is 3×. When L1-Cache size is further reduced to 1kB, the error rate reduction is up to 100× in the case of "rijndael_dec" and the average reduction in error rate is 30×. Figure 13.(b) plots the effect of L1-Cache resizing on the performance of the processor core. When L1-Cache size is reduced from 32kB to 8kB there is some reduction in performance for 13 applications; however 11 applications exhibit small reduction in performance. When L1-Cache size is reduced from 32kB to 1kB the impact on performance is higher, but for 5 applications this impact is still negligible. As can be seen, 1kB of L1-Cache is not sufficient and 8kB of L1-Cache is a better choice for most applications to maintain processor performance under reduced processor core clock speed.

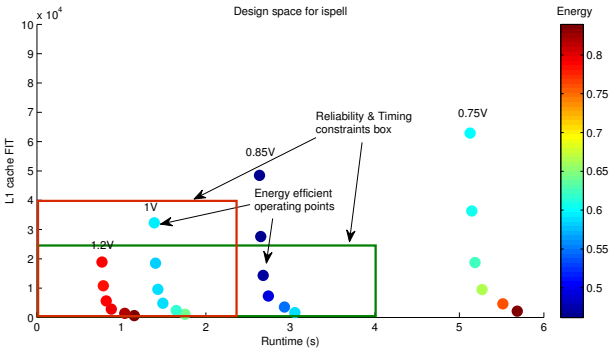
C. Joint Optimization of Reliablity, Performance and Energy

Figure 14 shows case studies of design optimizations (Figure 9) for two applications "lame" and "ispell", highlighting the design trade-offs between L1-Cache resizing and VFS control (as shown in Figure 1). The X-axis shows the run time of the applications. The error rates in terms of FIT are shown in left Y-axis. The color coded normalized energy consumption is shown on the right (darker red to darker blue represent higher to lower energy consumptions). A number of operating points as colored are dots shown, each dot representing reliability, performance and energy consumption trade-offs. The dots along each line are the results of different L1-Cache sizes; the top one represents operating point for 32kB L1-Cache, while the lower operating points result from lower L1-Cache sizes. Lower L1-Cache size also affects the runtime as it increased with L1 size reduction and improves reliability due to decreased vulnerability and fault rate (Section III-B). Different dotted lines represent a given VFS scalings applied.

Figure 14.(a) shows the design space for the application "lame". Two rectangular boxes enclose two different timing and reliability constraints (DTC of 3 and 4 seconds corresponding to DRC of 40,000 FIT and 25,000 FIT respectively). Therefore only the operating points inside the constraint box can be selected for a given DTC and DRC. The most energy efficient operating point is the darkest point inside the constraint box. It can be observed that L1-Cache resizing improves the energy efficiency by moving the most energy efficient voltage point into the reliability and timing constraint box. For example without cache resizing, if the DRC is 25,000 FIT the only available operating voltage is 1.2V. Reducing the cache size to 8kB makes 0.85V viable, which is also the most energy-efficient operating point. Similar observations can also be made with the DRC of 40,000 FIT and DTC of 3 seconds. The L1-Cache resizing to 16KB also makes 0.85V as the most energy-efficient and reliable operating point. Figure 14.(b) shows that the best operating points also varies with applications. For example, for a DTC of 3 seconds and DRC of 40,000 FIT, 1V with 32kB



(a) Design operating points for application "lame"



(b) Design operating points for application "ispell"

Fig. 14. Example optimizations for application (a) lame and (b) ispell.

TABLE II
OPTIMIZED ARCHITECTURAL CONFIGURATIONS OF MIBENCH APPLICATIONS WITH A DRC 25,000 FIT

	VFS		VFS, L1-Cache resizing,				VFS, L1-Cache resizing,dynamic protection			
	Vdd	Energy	Vdd	L1	Energy	Saving	ECC		Energy	Saving
Application	(V)	(J)	(V)	(kB)	(J)		L2	DRAM	(J)	
basicmath	0.85v	1.788	0.85v	32kB	1.788	0.0%	off	off	1.568	12.3%
bitcount	0.85v	0.396	0.85v	1kB	0.390	1.6%	off	off	0.342	13.7%
qsort	1.20v	0.295	0.85v	16kB	0.204	31.0%	on	on	0.204	31.0%
susan_corners	1.20v	0.020	0.85v	4kB	0.013	34.4%	on	on	0.013	34.4%
susan_edges	1.00v	0.040	0.85v	8kB	0.036	11.6%	on	on	0.036	11.6%
susan_smoothing	1.20v	0.226	0.85v	8kB	0.151	33.1%	on	off	0.135	40.3%
djpeg	1.20v	0.020	0.85v	16kB	0.014	31.6%	on	off	0.012	39.0%
cjpeg	0.85v	0.059	0.85v	16kB	0.059	0.9%	on	on	0.059	0.9%
lame	1.20v	0.961	0.85v	16kB	0.654	31.9%	on	on	0.654	31.9%
typeset	1.20v	0.427	0.85v	16kB	0.349	18.4%	on	on	0.349	18.4%
dijkstra	1.20v	0.161	0.85v	8kB	0.126	21.9%	on	off	0.111	31.2%
patricia	1.20v	0.604	0.85v	16kB	0.479	20.6%	on	on	0.479	20.6%
ispell	1.20v	0.814	0.85v	16kB	0.613	24.7%	on	on	0.613	24.7%
rsynth	1.20v	1.714	0.85v	8kB	1.265	26.2%	on	on	1.265	26.2%
stringsearch	0.85v	0.003	0.85v	32kB	0.003	0.0%	off	off	0.003	11.3%
rijndael_enc	1.00v	0.288	0.85v	16kB	0.262	8.9%	off	off	0.230	20.2%
rijndael_dec	1.00v	0.277	0.85v	8kB	0.266	4.0%	off	off	0.239	13.9%
sha	0.85v	0.066	0.85v	2kB	0.066	1.0%	off	off	0.058	12.9%
crc	1.00v	1.651	1.00v	2kB	1.614	2.3%	off	off	1.445	12.5%
fft	1.20v	0.426	0.85v	16kB	0.314	26.1%	on	on	0.314	26.1%
ifft	1.20v	0.234	0.85v	16kB	0.174	25.6%	on	on	0.174	25.6%
cadpcm	0.85v	0.380	0.85v	4kB	0.376	1.0%	off	off	0.332	12.6%
dadpcm	0.85v	0.273	0.85v	4kB	0.271	0.8%	off	off	0.239	12.5%
gsm_toast	1.20v	0.890	0.85v	16kB	0.684	23.2%	off	off	0.588	33.9%

L1-Cache is the most energy-efficient and reliable operating point.

Table II shows experimental results for the low-cost reliable design optimization (Algorithm 1) applied to various MiBench applications under the reliability constraint of 25,000 FIT. The 1st column shows the benchmark application; the 2nd main column shows minimum energy consumption under VFS and its corresponding supply voltage which was limited by reliability constraints; the 3rd main column shows the minimum energy consumption under VFS when L1-Cache resizing is used which allows the processor to operate under a more energy efficient supply voltage point; the last main column shows the minimum energy consumption under VFS when both L1-Cache resizing and dynamic ECC protection are used to further reduce energy consumption. 2nd column shows that, for some applications, supply voltage scaling is limited by the reliability constraints of L1-Cache, therefore reliability constrains can restrict the system from operating on minimum energy. L1-Cache resizing mitigates the impact of VFS on reliability; thus for all applications the supply voltage for minimum energy consumption can be used as shown in the 4th column. The 8th and 9th columns show the enable signals for ECC protection of L2-Cache and DRAM. It shows that L2-Cache and DRAM ECC protection are only enabled in the applications where their reliability is lower than 25,000 FIT. Comparing to the processor system with only VFS, L1-Cache resizing save upto 34% of energy, and 16% on average across all applications. This is achieved by enabling the processor to operate on more energy-efficient supply voltage, while still meeting the reliability constraints. Dynamic protection of L2-Cache and DRAM reduce energy consumption further, when used together with L1-Cache resizing. As can be seen, it achieves energy saving of upto 40% and on average 21% across all applications.

VI. CONCLUSIONS

A memory system analysis framework facilitating a holistic reliability analysis was presented, showing that memory component vulnerability varies greatly depending on the application. The analysis further highlighted that appropriate memory sizes and protection policies can reduce vulnerability significantly at the cost of increased energy overheads. Based on the analysis, a design flow is proposed with an aim of achieving energy-efficiency and reliability through careful optimization of VFS and these policies. The proposed design flow is evaluated through experiments in Gem5. The design flow is expected to be useful in energy-efficient and reliable designs for application-specific systems.

REFERENCES

[1] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE TDMR*, vol. 5, no. 3, pp.305–316, Sept. 2005.

[2] F. Wang and Y. Xie, "Soft error rate analysis for combinational logic using an accurate electrical masking model," *IEEE TDSC*, vol. 8, no. 1, pp. 137–146, Jan. 2009.

[3] M. Kandala *et al.*, "An area-efficient approach to improving register file reliability against transient errors," in *AINAW Intl.*, pp. 798–803, 2007.

[4] D. H. Yoon and M. Erez, "Memory mapped ECC: Low-cost error protection for last level caches," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 116–127, Jun. 2009.

[5] J. Hu *et al.*, "On the exploitation of narrow-width values for improving register file reliability," *IEEE TVLSI*, vol. 17, no. 7, pp. 953–963, July. 2009.

[6] Y. Cai *et al.*, "Cache size selection for performance, energy and reliability of time-constrained systems," in *ASP-DAC*, pp. 923–928, 2006.

[7] S. Kim and A. Somani, "Area efficient architectures for information integrity in cache memories," in *ISCA*, pp. 246–255, 1999.

[8] L. Li *et al.*, "Soft error and energy consumption interactions: a data cache perspective," in *ISLPED*, pp. 132–137, 2004.

[9] S. Yang *et al.*, "Improved State Integrity of Flip-Flops for Voltage Scaled Retention Under PVT Variation" in *IEEE TCAS-I*, vol. 60, no. 11, pp. 2953–2961, Nov. 2013.

[10] W. Zhang *et al.*, "ICR: In-cache replication for enhancing data cache reliability," in *DSN*, pp. 291–300, 2003.

[11] J. Kim *et al.*, "Multi-bit error tolerant caches using two-dimensional error coding," in *MICRO*, pp. 197–209, 2007.

[12] G. Memik *et al.*, "Increasing register file immunity to transient errors," in *Design, Automation and Test in Europe (DATE)*, pp. 586–591, 2005.

[13] L. Li *et al.*, "Soft error and energy consumption interactions: a data cache perspective," in *ISLPED*, pp. 132–137, 2004.

[14] S. Yang *et al.*, "Reliable state retention-based embedded processors through monitoring and recovery," *IEEE TCAD*, vol. 30, no. 12, pp. 1773–1785, 2011.

[15] A. Saleh *et al.*, "Reliability of scrubbing recovery-techniques for memory systems," *Reliability, IEEE Trans. on*, vol. 39, no. 1, pp. 114 –122, Apr. 1990.

[16] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[17] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, pp. 469–480, 2009.

[18] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *IISWC IEEE Symp.*, pp. 3–14, 2001.

[19] A. Chandrakasan *et al.*, "Low-power CMOS digital design," *IEEE JSSC*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[20] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *Reliability Physics Symposium, Intl.*, pp. 5B.4.1 –5B.4.7, 2011.

[21] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *IEEE JSSC*, vol. 40, no. 1, pp. 310–319, Jan. 2005.

[22] N. Weste *et al.*, "CMOS VLSI Design, A Circuits and Systems Perspective," 3/E, Addison-Wesley, 2005