University of Maryland Technical Report UMIACS-TR-2013-02

Symbiotic Cache Resizing for CMPs with Shared LLC

Inseok Choi and Donald Yeung
Department of Electrical and Computer Engineering
University of Maryland at College Park
{inseok, yeung}@umd.edu

Abstract

This paper investigates the problem of finding the optimal sizes of private caches and a shared LLC in CMPs. Resizing private and shared caches in modern CMPs is one way to squeeze wasteful power consumption out of architectures to improve power efficiency. However, shrinking each private/shared cache has different impact on the performance loss and the power savings to the CMPs because each cache contributes differently to performance and power. It is beneficial for both performance and power to shrink the LRU way of the private/shared cache which saves power most and increases data traffic least.

This paper presents Symbiotic Cache Resizing (SCR), a runtime technique that reduces the total power consumption of the on-chip cache hierarchy in CMPs with a shared LLC. SCR turnoffs private/shared-cache ways in an inter-core and inter-level manner so that each disabling achieves best power saving while maintaining high performance. SCR finds such optimal cache sizes by utilizing greedy algorithms that we develop in this study. In particular, Prioritized Way Selection picks the most power-inefficient way. LLC-Partitioning-aware Prioritized Way Selection finds optimal cache sizes from the multi-level perspective. Lastly, Weighted Threshold Throttling finds optimal threshold per cache level. We evaluate SCR in two-core, four-core and eight-core systems. Results show that SCR saves 13% power in the on-chip cache hierarchy and 4.2% power in the system compared to an even LLC partitioning technique. SCR saves 2.7X more power in the cache hierarchy than the state-of-the-art LLC resizing technique while achieving better performance.

1. Introduction

The power wall is currently the main limiter to achieving high performance in modern CPUs, and has been one of the most critical problems that computer architects face over the past several years [20]. Unfortunately, this problem will only get worse in the future as process technologies continue to scale down feature sizes. As such, power efficiency will remain an extremely important design goal, and will require hardware designers to continue to make efforts to squeeze wasteful power consumption out of architectures.

A key place to look for power savings is in the on-chip cache hierarchy. Caches occupy a large portion of the CPU's available die area–upwards of 50% in today's CPUs–so they take up a significant portion of a processor's overall power dissipation. In addition, caches are sized for the worst case. This means an average computation cannot effectively utilize all of the cache capacity. Such cache over-provisioning can result in significant waste that, if eliminated, can yield large power savings without sacrificing much performance.

Several researchers have investigated *cache resizing techniques* [1, 3, 4, 23, 24, 28, 40, 41] that target this form of waste. Cache resizing is an architecture-level power management technique to determine the minimum cache that a program needs to run at near-peak performance, and then reconfigure the cache by enabling/disabling cache ways or sets to implement this efficient capacity. Resizing can reduce the amount of cache activated per access, and also enable circuit-level techniques (*e.g.* gated-V_{dd} [28]) to shut down the unused portion of the cache. This can translate into significant dynamic and static power savings.

CMPs are prevalent because exploiting instruction level parallelism (ILP) incurs high power dissipation but yield only modest performance gains. CMP scaling, *i.e.* increasing the number of cores, will continue in the foreseeable future as transistor count increases. Modern CMPs are commonly equipped with a shared last-level cache (LLC) to efficiently utilize cache capacity and share data between cores. Ideally, finding optimal sizes of the private caches and the shared LLC in such CMPs leads to saving most of the wasteful power consumption in the on-chip cache hierarchy without any performance degradation or only with unnoticeable/acceptable performance degradation. While extremely effective on uniprocessors, existing resizing techniques cannot be simply applied to modern CMPs. This is because finding the optimal size of each private cache on CMPs is already an *NP-hard* problem. As such, having a scalable algorithm is crucial to alleviate the *power inefficiency* in the private caches in CMPs.

On the other hand, a shared LLC can be considered as a single cache, so we can directly apply an existing cache resizing techniques to the shared LLC. A recent study [36] applied the cache resizing technique to the shared LLC in CMPs on top of the utility-based cache partitioning scheme [29]. The study explored the wasteful power consumption of the LLC, mostly from the static power of the LLC, thus reducing the power consumption of the LLC without noticeable performance degradation by not allocating ways with lower utility than pre-specified threshold. Unfortunately, the technique alone can not squeeze most of the wasteful power consumption out of the on-chip cache hierarchy because most of the over provisioning, from the perspective of dynamic power, occurs at the private caches. Thus, it is important to find a way to resize private caches on top of the resizable shared LLC.

In this paper, we present *Symbiotic Cache Resizing (SCR)*, a novel greedy algorithm to resize private caches and the shared LLC in CMPs that squeezes wasteful power consumption out of both resizing. In particular, SCR achieves higher performance and saves more power than the existing LLC resizing technique. Figure 2 summarizes our offline-study results by showing weighted speedups and power consumption of the on-chip cache hierarchy, as we compare the static SCR to previous studies and our exhaustive searches (we will discuss our offline study in Sections 4.4 and 5.1 in detail). As shown, the static SCR can achieve both higher performance and higher power efficiency than the existing LLC resizing technique by eliminating the wasteful power consumption both from the private caches and the LLC.

SCR orchestrates the private cache and shared LLC by utilizing *Weighted Threshold Throttling (WTT)*. WTT finds optimal thresholds to control a private-cache resizing technique and a LLC resizing technique. Having an optimal threshold per different cache level is the key to *symbiosis*, because resizing each type of cache (private vs. shared) has a different impact on the overall performance loss and power consumption. Moreover, the impact varies across applications. For resizing the LLC, WTT employs the existing state-of-the-art technique [36]. However, we propose a new technique, called *LLC-Partitioning-aware Prioritized Way Selection (LP-PWS)*, to control private cache resizing in WTT. LP-PWS resizes private caches symbiotically by utilizing *reuse distance* profiles across cores. LP-PWS monitors power efficiency gain (PEG) per *LRU* way of each core and disables the ways in the PEG order to achieve the best power saving per *unit performance loss*. In addition, LP-PWS adopts a multi-level cache resizing technique to optimize the total power consumption of a cache hierarchy, thus achieving *symbiosis* when running with a resizable shared LLC. Resizing private caches to save power, however, can increase private caches' miss rates, resulting in greater power dissipation at the next level of cache (or the LLC) due to increased accesses. The access energy to a shared LLC is generally larger than the access energy of a private cache. For this reason, shrinking down a private cache usually reduces the total power consumption of the cache hierarchy, but after some point, it will increase the total power consumption due to the larger access energy at the shared LLC. Accordingly, it is crucial to control private cache resizing with awareness of the total power consumption in the cache hierarchy.

This paper makes the following contributions:

- We conduct a limit study that searches the solution space exhaustively to find the solution that outperforms the *state-of-the-art* LLC resizing technique in both performance and power savings.
- We propose *Symbiotic Cache Resizing*, a greedy algorithm that can *heuristically* find good resizing solutions in an online fashion.
- In particular, we explain and show that *naive* PWS, LP-PWS and WTT can achieve significant power savings while maintaining high performance relative to other techniques.
- We show that SCR can save up to 54.5% total cache-hierarchy power and 16.9% total system power.
- To the best of our knowledge, this is the first study proposing a run-time technique to find the optimal on-chip cache hierarchy in CMPs by resizing both private caches and the shared LLC.

The remainder of this paper is organized as follows. Section 2 studies background and motivation of SCR. Then, Section 3 presents analytical model for SCR and its architecture in details. Section 4 explains experimental environment and methodology and Section 5 evaluates power savings and performance of SCR. Finally, Section 6 discusses related work, and Section 7 concludes the paper.

2. Background and Motivation

Cache resizing has been known for several decades, but its application in CMPs is not fully investigated yet. Although there has been significant work on cache resizing, existing techniques are limited to uniprocessors. In particular, most studies consider resizing a single level cache for a uniprocessor only [1, 23, 24, 28, 40, 41], (typically the L1 caches). A recent study [36] investigates cache resizing in a multi-core platform, but it only studies it for the shared LLC and does not cover private caches. Unfortunately, there's no comprehensive study on cache resizing in a modern CMP cache hierarchy yet.

Moreover, since both dynamic and static power are important, only controlling the size of a single level of cache potentially misses significant opportunities for power savings. The trend for modern CPUs is towards deeper cache hierarchies which distributes the power consumption across different caching levels. For dynamic power consumption, the private cache is the greatest culprit, but for static power consumption, the LLC is by far the greatest concern due to its large area. As such, investigating multi-level resizing is mandatory to eliminate all wasteful power consumption in the cache hierarchy.

2.1. Private Cache Resizing in CMPs

Private caches can account for a significant part of the overall power consumption, and can also vary across programs. Figure 1 shows the power consumption breakdowns for the SPEC 2006 benchmarks in a 2-way out-of-order core with 32KB private cache and 1MB LLC (details will follow in Section 4). In particular, the dynamic power of the private cache alone can take up to 20% of the total system power consumption or 50% of the total power consumption of the cache hierarchy. In terms of the absolute power consumption, Table 1

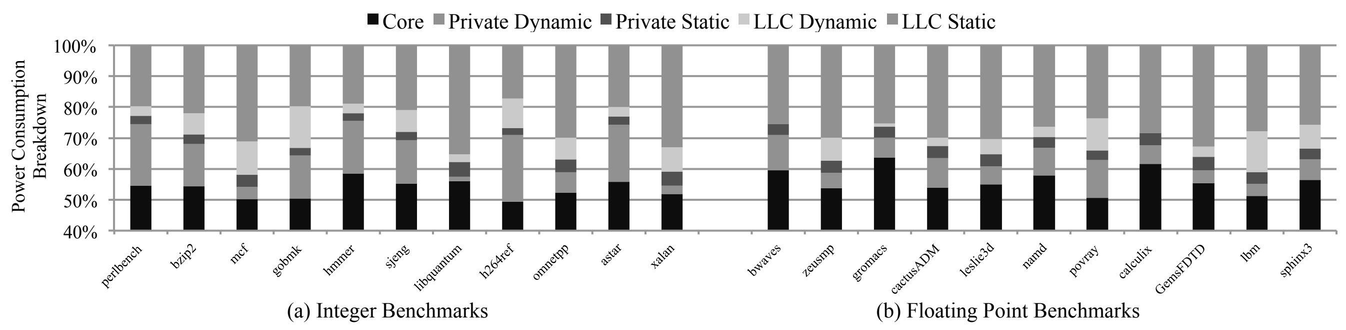


Figure 1: Breakdown of system power consumption for the SPEC CPU2006 benchmarks.

	perlbench	bzip2	mcf	gobmk	hmmer	sjeng	libquantum	h264ref	omnetpp	astar	xalan
TS (W)	2.627	2.397	1.696	2.724	2.766	2.513	1.468	3.066	1.751	2.606	1.579
L1D (W)	0.525	0.333	0.067	0.380	0.472	0.352	0.023	0.665	0.117	0.482	0.045

	bwaves	zeusmp	gromacs	cactusADM	leslie3d	namd	povray	calculix	GemsFDTD	lbm	sphinx3
TS (W)	2.015	1.752	2.033	1.730	1.724	1.976	2.232	1.811	1.587	1.908	2.029
L1D (W)	0.230	0.088	0.133	0.163	0.097	0.178	0.270	0.111	0.063	0.078	0.136

Table 1: Total system power (TS) and L1 dynamic power (L1D) for SPEC CPU2006 benchmarks.

mcf								astar							
Total accesses			856647	Total misses			166736	Total accesses			4071875	Total misses			30415
	Way 1	Way 2	Way 3	Way 4	Way 5	Way 6	Way 7		Way 1	Way 2	Way 3	Way 4	Way 5	Way 6	Way 7
M+	32534	6994	4755	4024	3406	2838	2442	M+	122031	14174	4894	2989	2437	2125	1900
PEG	26.3	122.5	180.2	212.9	251.5	301.8	350.9	PEG	33.4	287.3	832.0	1362.4	1671.2	1916.4	2143.0

Table 2: Increased number of cache misses (M+) and power efficiency gain (PEG).

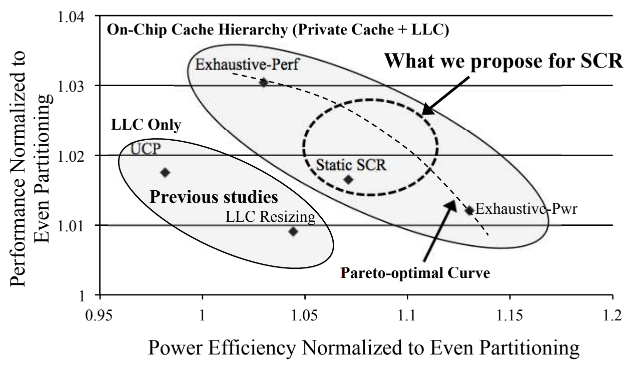


Figure 2: Scope and goal of this work.

shows the dynamic power consumption of the private cache varies from 0.02 W to 0.5 W. Resizing each core's private cache independently without considering the absolute power levels will fail to reach a globally optimal solution. Instead, for a given amount of acceptable performance loss (*i.e.* increased traffic to the shared LLC), it is crucial to apply resizing on the private cache that will save the most power.

2.2. Intra-Core Multi-Level Optimization

Cache-partitioning techniques have been widely investigated for higher performance and better power efficiency [29, 30, 22]. Most of runtime state-of-the-art techniques dynamically change partitioning. As such, resizing private caches in CMPs will encounter a dynamically changing LLC partition size. Although LLCs commonly use the serialized access technique, which saves the data-array access power when a cache miss occurs, its tag access power can be significant compared to the potential power savings from private cache resizing. The dynamic power consumption of tag accesses is dependent on the LLC partitioning, the allocated LLC ways in UCP [29] and thus, private cache resizing increases the power consumption of the LLC. To alleviate this problem, private cache resizing should be performed with the awareness of the increased dynamic power at the LLC constrained by a threshold to avoid severe performance degradation.

2.3. Private Cache Resizing vs LLC Resizing

As discussed earlier, multi-level cache hierarchies distribute power consumption across the different levels. Each level of cache has its own power savings and performance degradation when the waste is squeezed out. Since the overall performance impact and power savings are not trivial, when multiple cache-resizing techniques are combined, it is crucial to predict potential power savings and performance loss to control each technique for saving most of the power while maintaining high performance.

Cooperative Partitioning [30], or *LLC Resizing*, is a previous technique focused on resizing a CMP's shared LLC. It disables ways in the LLC when the *utility* of the way is not high enough. Combining the private cache resizing and the LLC resizing can be either *symbiotic* or destructive, depending on the workload. If the LLC resizing technique disables some number of ways on top of aggressive private cache resizing that causes significant additional traffic to the LLC, there can be significant performance degradation which is unacceptable in high performance computing. On the other hand, it is possible to combine the two techniques in such a way

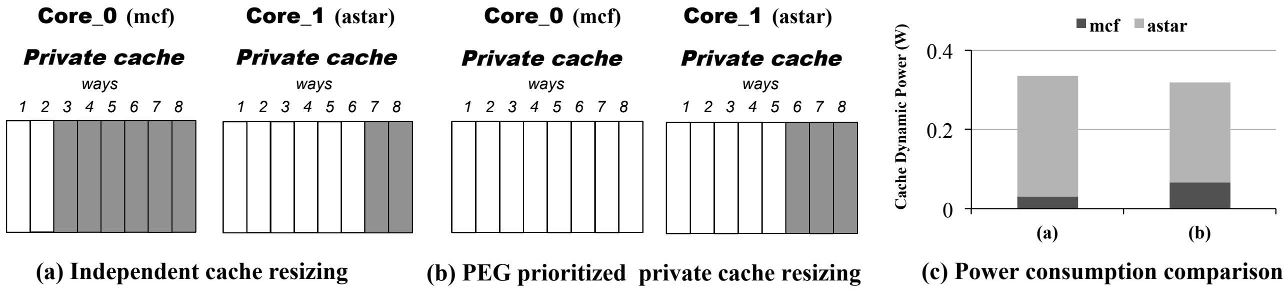


Figure 3: Comparing private-cache resizing techniques.

that achieves most of the potential power savings with negligible performance degradation. This is possible when the two techniques form *symbiosis* such that the private cache resizing saves power with near-peak performance and the LLC resizing employs more ways than it would in the absence of private cache resizing, to compensate the performance.

3. SCR

3.1. Framework

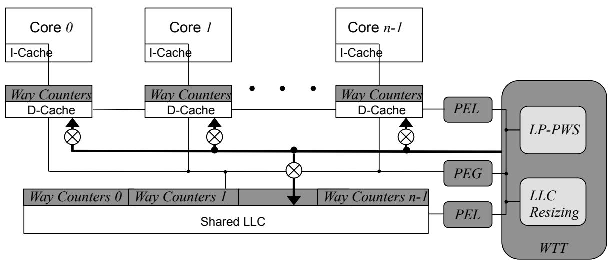


Figure 4: SCR framework

Figure 4 illustrates the framework of SCR. We develop the design of SCR to achieve most of the potential power savings and at the same time attain high performance based on the principal rule which enforces the configurations resulting in *best power saving* per *unit performance degradation*, which can be captured as *Power Efficiency Gain* (PEG). SCR consists of mainly two parts: the monitoring part and the control part. The monitoring part includes way counters to approximate the *reuse-distance* profile in the private /shared caches, and PEG/PEL (Power Estimation Logic). The control part consists of LP-PWS, LLC Resizing and WTT.

LP-PWS monitors PEG to prioritize the cache way in private caches, which the best power savings can be expected by disabling of. The power estimation logic (PEL) in SCR provides the approximation of dynamic and static power consumption of a given level of the cache and provides the values to the logic which determines the size of the private cache and the LLC. SCR works with the existing LLC resizing technique *symbiotically*. In particular, LP-PWS leads to an optimal multi-level cache resizing. Second, PEL-based WTT achieves further power saving without significant performance degradation. We detail the design of the two algorithms after presenting the analytical model for SCR .

3.2. Analytical Model for SCR

Power Optimization with Bounded Performance Degradation In high performance systems, achieving desired performance levels and reducing power consumption are both important considerations. As such, our objective function is the lowest power consumption with a *bounded* performance degradation, not *Energy-Delay Product* or *Energy-Delay-Squared Product* which are more common in circuit/device-level designs.

Assume that we solve the problem of finding optimal private cache sizes for a two-core system. Let x and y be the size of the private caches. Let $f(x,y)$ be the power consumption of the system consisting of a core with a private cache size of x and another core with a private cache size of y. Let $g(x,y)$ be the performance, *i.e. weighted speedup*, of the system.

$$Power = f(x,y),\ Performance = g(x,y)$$

Let μ be the normalized value of the degraded performance and x_B and y_B are the baseline sizes of the private caches (in this study, we only focus on homogeneous systems, so $x_B = y_B$). Then, a simple form of this problem we want to solve is

$$\text{minimize } f(x,y), \text{ subject to } \frac{g(x,y)}{g(x_B,y_B)} \geq \mu \qquad (1)$$

Solving the *Lagrangian* for this problem is very challenging. Moreover, its solutions are neither accurate nor practical because performance and power characterizations are challenging and the derived solutions are not always configurable. However, our empirical study shows that approximation, given below, discovers the solution effectively based on the exhaustive search over the solution space.

SCR Approximation We approximate the solution by searching x_{scr}, y_{scr} such that

$$\begin{aligned} &\text{maximize } f(x_B,y_B) - f(x_{scr},y_{scr}), \\ &\qquad \text{subject to } (T(x_B,x_{scr}) + T(y_B,y_{scr})) < threshold \end{aligned} \qquad (2)$$

where $T(a,b)$ is a function returning the normalized value of increased traffic to the LLC. In Equation 2, we change the constraints from being based on pure performance (normalized speedup) to being based on performance-related events, which is easier to predict. We predict the increased traffic with *reuse-distance profiling* by utilizing *way counters* at run time. We take a *greedy* approach to solve this problem, based on *power efficiency gain* to compare potential power savings by disabling each LRU way in private caches. The effectiveness of the approximation will be discussed in Section 5.1.

3.3. SCR Designs

Power Efficiency Gain Comparing the impact, from the power and the performance perspective, of disabling a single way of a given core's private cache in the CMP is crucial when searching for the global optimal. If the LRU way of the private cache in a given core (i) is disabled, then the expected power efficiency gain per *unit performance degradation* is:

$$PEG_i = total\_accesses/way\_counts[l], \qquad (3)$$

where $l = current\_LRU\_way$. Table 2 shows PEGs per way of the example workload consisting of *mcf* and *astar*. $M+$ numbers are measured when the number of the ways of the cache is reduced by one, *e.g.* "way 5" columns denote the increased number of misses when the cache has been changed from 6 ways to 5 ways.

Algorithm 1: *Naive* Prioritized Way Selection

```
PWS(threshold):
begin
    increasedTraffic = 0
    allowedTraffic = trafficToLLC * threshold
    while increasedTraffic < allowedTraffic do
        foreach core do
            PEG[i] = get_peg(i)
        end
        selectedCore = core with maximum PEG
        increasedTraffic += disable_way(selectedCore)
    end
    return ways per core
end

get_peg(i):
begin
    return total_accesses[i]/way_counts[current_way - 1]
end

disable_way (i):
begin
    current_way -= 1
    return way_counts[current_way]
end
```

PEG-based Prioritized Way Selection Each private cache consumes dynamic power that is proportional to the number of accesses which can vary significantly across cores. As such, disabling a single way in private caches may have different power impact or *power efficiency gain*. Decisions made by monitoring only each core's performance are prone to be destructive because each core's increased data traffic to the LLC aggravates resource conflicts in the LLC. Figure 3-(a) shows an example. Downsizing each core's private cache, while monitoring its own performance degradation though, will result in the sum of the worst IPCs of each application. This is because each downsizing causes more traffic to the LLC, resulting in poorer IPC than would occur in a uniprocessor without a shared LLC.

PWS addresses this problem by selecting the way that provides the biggest power efficiency gain first. This will result in either saving more power by selecting ways that provide higher power efficiency gain, or achieving higher performance by disabling fewer number of ways for the same power savings compared to Figure 3-(a). Figure 3-(b) shows an example of *symbiotic cache resizing* with the PWS technique. In Figure 3-(b), only 3ways in total are disabled, compared to 8 ways in Figure 3-(a), resulting in lower power dissipation compared to the power dissipation in Figure 3-(a). PWS selects the ways to be disabled based on their PEG values. PWS prioritizes the ways in PEG order until the expected performance degradation reaches a pre-defined threshold, as shown in Algorithm 1.

Algorithm 2: LLC-Partitioning-aware PWS (LP-PWS)

```
LP_PWS(threshold):
begin
    increasedTraffic = 0
    allowedTraffic = trafficToLLC * threshold
    prevPower[i] = power_estimation(i, private_max_way,
    llcPartition[i])
    while increasedTraffic < allowedTraffic do
        foreach core do
            PEG[i] = get_peg(i)
        end
        selectedCore = core with maximum PEG
        increasedTraffic += disable_way(selectedCore)
        ********throttling ********
        if power_estimation(selectedCore) >
        prevPower[selectedCore] then
            break
        end
        prevPower[selectedCore] = power_estimation(selectedCore)
    end
    return ways per core
end

power_estimation(i):
begin
    privPower = dynamic_power(i, priv) + static_power(i, priv)
    llcPower = dynamic_power(i, LLC) + static_power(i, LLC)
    return privPower + llcPower
end
```

Intra-Core LLC Partitioning-aware PWS Although *naive* PWS determines a solution to achieve high power efficiency, each per-core solution still may not achieve one of the *Pareto*-optimal solutions in the power-performance solution space because it does not take the dynamic access energy of the the LLC into consideration. The increased traffic to the LLC results in additional power consumption. The additional power can be significant as we shrink the private cache aggressively, resulting in a total power increase. In addition, tag-access energy keeps changing because a LLC resizing technique partitions the LLC and shrink the number of active ways dynamically. For this reason, the optimal degree of disabling private cache can not be determined independently. LP-PWS uses the power estimation logic (PEL), shown in Figure 4 to improve upon *naive* PWS by throttling the amount of private cache resizing. The technique is specified in Algorithm 2.

Algorithm 3: Weighted Threshold Throttling

```
WTT($T_{Priv}$, $T_{LLC}$): // default thresholds for LP-PWS and LLC Resizing
begin
    privWays[i] = LP_PWS($T_{Priv}$)
    llcWays[i] = LLC_Resizing($T_{LLC}$)
    foreach core do
        /* private cache resizing power */
        powerPrivResizing += power_estimation(i)
    end
    foreach core do
        /* LLC_Resizing power */
        powerLlcResizing += power_estimation(i)
    end
    powerSavingPriv = powerBaseline - powerPrivResizng
    powerSavingLlc = powerBaseline - powerLlcResizing
    powerSaving = powerSavingPriv + powerSaving Llc
    throtPriv = powerSavingPriv / powerSaving
    throttle = powerSavingLlc / powerSaving
    LP_PWS(throtPriv * $T_{Priv}$)
    LLC_Resizing(throtLlc * $T_{LLC}$)
    return private and LLC ways
end
```

Weighted Threshold Throttling The goal of SCR is to eliminate wasteful power consumption both from private caches and the shared LLC. To achieve this goal, we need to utilize both resizing techniques symbiotically so as not to degrade performance severely. In particular, we try to achieve the sum of power savings from both techniques while allowing only the performance degradation of one of the technique. Although we assume given thresholds for both resizing techniques, we adjust the *effective* thresholds dynamically.

We propose Weighted Threshold Throttling (WTT) which adjusts each threshold dynamically based on the ratio of the expected power savings of each technique to the overall power savings, as shown in Algorithm 3. This means WTT gives more weight to the cache resizing technique that saves more power with the given default threshold, while discouraging the resizing in the other caching level to prevent severe performance degradation. We use the algorithm described in [36] for the *LLC_Resizing()* in Algorithm 3, and omit its explanation here.

3.4. Scalability of SCR

As we mentioned earlier, finding an optimal private cache size in CMPs is already an *NP-hard* problem. This is because it has a time complexity of $O((N_p)^M)$, where the CMPs have M cores and each core has a N_p-way private cache.

Group	Benchmark	MPKI	Group	Benchmark	MPKI
High	Mcf	51	Low	Astar	0.82
	Libquantum	29		Perlbench	0.69
	Lbm	22		Hmmer	0.66
	Omnetpp	15		H264ref	0.54
	GemsFDTD	14		Sjeng	0.27
Medium	Leslie3d	9.5		Gobmk	0.2
	Sphinx3	8.2		Calculix	0.2
	Xalan	6.8		Gromacs	0.13
	Bwaves	4.8		Namd	0.07
	Zeusmp	4.1		Povray	0.03
	CactusADM	2.3			
	Bzip2	1.9			

Table 3: Benchmarks classification.

We reduced the time complexity down to $O(N_pM)$ for *naive* PWS and LP-PWS as shown in Algorithms 1 and 2. Previous studies [29, 36] showed that LLC partitioning techniques do not take longer than $O(M^2)$. SCR has a time complexity of $O(M(N_p+M))$ because WTT, shown in Algorithm 3, integrates the private-cache-resizing algorithm and the LLC-resizing algorithm.

3.5. Hardware Overhead

Way Counters The major additional hardware circuit to implement SCR is way counters. We assume the same circuits used in the shared LLC in [29] and employ similar circuits in the private caches. Way counter overheads in term of area and power consumption are minimal, though we take them into consideration when we measure the system power.

4. Experimental Methodology

4.1. Benchmarks

We use 22 SPEC CPU2006 benchmarks (11 integer and 11 floating point), as shown in Table 3. We compile the benchmarks on an Alpha CPU emulator [8]. We run the *Linux* (*Debian Etch*) system on the emulator, and then install SPEC CPU 2006 on it. We use the native Alpha compiler, gcc-4.1.1 (provided along with *Debian*). We compile the benchmarks with the -O2 option and link glibc-2.5 statically. One integer benchmark (403.gcc) and six floating point benchmarks (416.gamess, 433.milc, 447.dealll, 450.soplex, 465.tonto, and 481.wrf) could not be compiled, so they have been omitted from our study. Using the reference inputs, all of the benchmarks were run to completion on SimPoint [13]. We take the most representative *simpoint*, consisting of 1B instructions per benchmark. Each simulation point contains 1.1B instructions, 100M instructions for cache warmup and 1B instructions for the representative simulations.

Workloads We generate multi-program workloads randomly to mix all three categories in Table 3. We created 20 workloads for 2-, 4- and 8-core CMPs, resulting in a total of 60 workloads.

Cores	2.0 GHz 2-way out-of-order 64-entry ROB, 32-entry LSQ Gshare branch predictor, 1024-entry BTB
L1 I-Cache	32 KB, 2-way, 64-byte blocks
L1 D-Cache	32 KB, 2-ports, 8-way, 64-byte blocks, 4 cycles
L2 Unified Shared Cache	Noninclusive 16/32/64-way 2MB/4MB/8MB for 2/4/8 cores 64-byte blocks, 13/15/17 cycles

Table 4: Architectural configuration.

4.2. Architectural Simulation

We use modified Simplescalar tools for the Alpha ISA [5] to conduct our study. Table 4 shows our baseline processor configuration. As mentioned in Section 2, we model state-of-the-art power-efficient cores. As such, we use a relatively narrow 2-way issue core to achieve high power efficiency. The cores are attached to a 2-level cache hierarchy. In particular, the on-chip cache hierarchy has a split 8-way 32KB L1 private cache and a unified and shared LLC. The LLC is 2MB for 2 cores, 4MB for 4 cores, and 8MB for 8 cores. Its associativity increases by 8 ways for additional core counts. The cache block size is 64 bytes for all caches. The baseline cache hierarchy maintains the noninclusive inclusion property for the LLC. We will study different inclusion properties in Section 5 and compare their performance and power consumption. Before resizing caches, we apply existing techniques to ensure the baseline cache hierarchy is reasonably efficient. In particular, we assume the LLC cache serializes tag and data accesses such that only a single data way is ever accessed regardless of the number of configured cache ways. Figure 1 shows the cache-hierarchy energy breakdown of the baseline multi-level caches for each SPEC CPU2006 benchmark.

Cache Reconfiguration To enable cache reconfiguration, we modified Simplescalar's cache module to model *selective cache ways* [1]. We assume all caches in the hierarchy are reconfigurable and can change their capacity in increments of a cache way from 1 to the associativity number of ways in the cache. (Our work does not consider I-cache resizing, and assumes the I-cache is always fixed). While each cache's access delay also changes across different configurations, we assume a constant number of CPU cycles to access each cache chosen to handle that cache's worst-case access delay (*i.e.* with all ways enabled).

4.3. Power Modeling

We use McPAT [21] and CACTI 6.5 [26] for power modeling. Our baseline model uses the 32*nm* technology node and ITRS *high performance devices*. We adopt a state-of-art circuit-level static power reduction technique to model the static power of the shared LLC more realistically. Specifically, we assume high-V_t devices throughout [15], but apply reverse body bias (RBB) in standby mode to further reduce standby leakage [37]. When an access occurs, we apply a forward body bias (FBB) to restore the threshold voltage for low access delay. We assume that applying FBB does not impact the access delay for the cache [37]. We utilize stack effect in conjunction with ABB to model way selection [31, 37]. We use the Model for Assessment of cmoS Technologies And Roadmaps (MASTAR 2011) from ITRS [25] to derive parameters required for CACTI according to our assumptions.

4.4. Implementation

Static Study We conduct an off-line exhaustive evaluation of static SCR to examine its potential power savings and performance improvement compared to other schemes including *UCP* and *LLC Resizing* based on UCP as well as exhaustive searches for optimal power, called *Exhaustive-Power*, and for optimal performance, called *Exhaustive-Performance*. The static study has two goals: first, identify potential performance gains and power savings compared to the other techniques, and second, determine the limit on the maximum performance and the power savings. The latter will allow us to assess how well our on-line SCR technique performs.

To facilitate the study, we run all possible combinations of ways of private cache and shared LLC. The extensive simulations enable the exhaustive search over the entire solution space to find the best static solution. In the case of *Exhaustive-Power*, we search for the solution which consumes the least power while achieving better weighted speedup than *LLC Resizing*. Likewise, in the case of *Exhaustive-Performance*, we search for the solution which has the highest weighted speedup while consuming less power than *LLC Resizing*.

Dynamic SCR We implemented our dynamic SCR techniques in the simulator from Section 3.3. In particular, we modified our simulator to emit an interrupt every 1M cycles and execute an interrupt handler. The interrupt handler estimates performance and power to determine the best private cache sizes across cores. Every 5M cycles, the interrupt handler also runs an LLC partitioning algorithm. We also modified the simulator to allow software to reconfigure the caches within the interrupt handler.

Performance and power estimations are provided by PEG and PEL in the simulator. PEG logic and PEL reads the way counters in the simulator. In particular, at each iteration during the Prioritized Way Selection, PEG values from each core are read to pick the maximum value and LLC-Partitioning-Aware Throttling can veto the decision and pick the core with second biggest PEG value. PEL requires per-access energies and leakage energies from CACTI to predict power consumption per epoch, so we implemented configurable registers to store these values. To facilitate our SCR algorithm, our simulator implements PEG and PEL on top of implementing private cache way counters per core.

Our simulator accounts for the overheads associated with resizing each cache. When up-sizing the private cache/shared LLC, we assume 100/1000, private/shared-LLC, cycles to power up each way, and 10 cycles per way to flash invalidate the newly powered-on cache blocks. When down-sizing, we walk the down-sized way(s) to flush its contents. Clean cache blocks are discarded after checking upstream caches to maintain inclusion. Dirty cache blocks check upstream caches and are also written back to the next-lower level. We assume these operations are pipelined such that flushing takes 1 cycle per walked cache block. Down-sized ways are selected in reverse way ID order. Because we do not physically move cache blocks once they are filled, the flushed cache blocks have an equal probability of being at any position in the LRU stack. Moreover, we do not attempt to reconstruct the per-set LRU stacks after flushing. Resizing is performed on the private cache and the shared LLC.

5. Results and Analysis

5.1. Static SCR

Figure 5 presents our off-line study results. The figure shows weighted speedups and cache-hierarchy power consumption of two-application workloads. SCR consumes less power than LLC Resizing while improving performance. On average, SCR improves performance by 0.7% and consumes 2.5% less power compared to LLC Resizing. Exhaustive-Power and Exhaustive-Performance show two Pareto-optimal solutions on the *weighted speedup-power* cartesian space. The weighted speedup and the cache-hierarchy power consumption of SCR are within the enveloped defined by two Pareto frontiers.

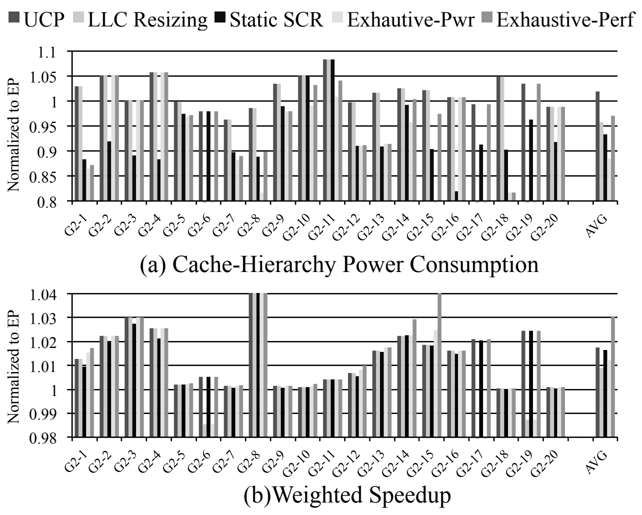


Figure 5: Offline study for static SCR.

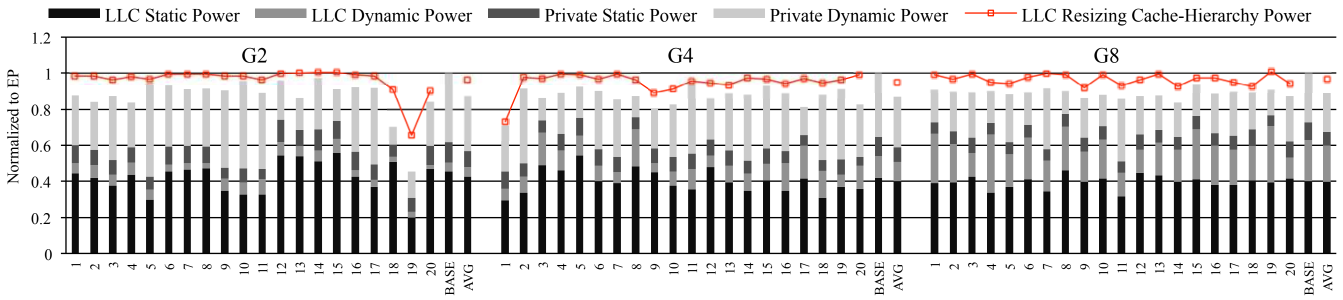
These results demonstrate our SCR techniques can provide significant power savings, even when using a static (fixed) configuration throughout the entire workload run, compared to the baseline cache hierarchy and to the LLC-only resizing. The bigger power savings and higher performance of SCR over LLC-only resizing can be explained as follows. First, private cache resizing can save dynamic power and static power consumption with some performance degradation. The results show that the average number of activated ways in the private caches is 6.5 compared to 8 in the baseline. And second, SCR compensates for the performance loss from the private cache resizing by utilizing more ways in the shared LLC compared to the LLC-only resizing. The average number of activated LLC ways per core of SCR and LLC-only resizing are 7.38 and 5.88, respectively, compared to 8 in even partitioning, and thus results in SCR's higher performance. Power consumption in the shared LLC of SCR is higher than LLC-only resizing, but overall, the cache-hiearchy power savings in SCR is greater than in LLC-only resizing due to the power savings in the private caches.

5.2. Dynamic SCR

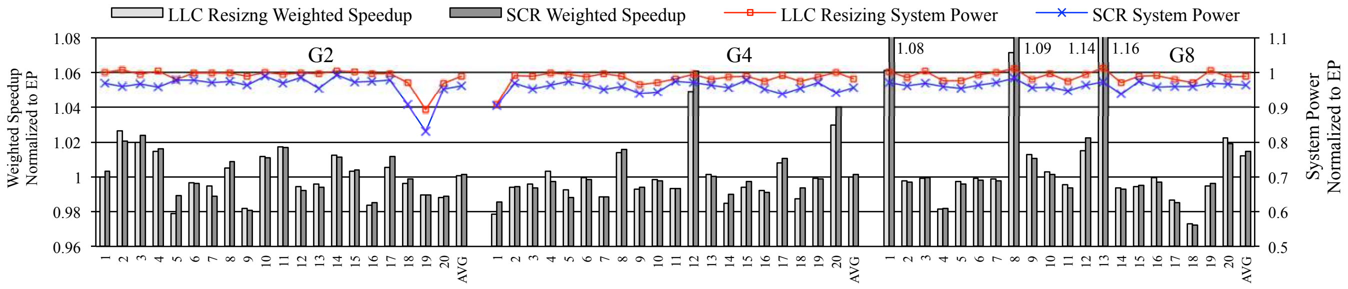
Symbiotic Cache Resizing Figure 6 summaries our dynamic SCR results by showing power consumption of the on-chip cache hierarchy and the system, and weighted speedups, as we compare the SCR technique to the LLC resizing technique. Figure 6-(a) shows the power consumption in the cache hierarchy for LLC-resizing and SCR techniques. The stacked bars break down cache-hierarchy power dissipation for SCR and the line reports total cache-hierarchy power consumption for the LLC resizing technique. These results demonstrate our SCR technique can provide significant power savings compared to the LLC resizing technique. SCR can save power in the cache hierarchy by as much as 55% in *G2-19* while the LLC resizing technique provides 25% power savings. On average, SCR saves private dynamic power by 20%, private static power by 25% and LLC static power by 4% across the workloads. LLC dynamic power changes by $\pm 10\%$ according to workload groups. As the "AVG" bars show, SCR provides 13% power savings from the on-chip cache hierarchy across workloads, representing a 2.7x increase in the power saved by the LLC resizing technique.

Figure 6-(b) shows weighted speedups and system power for the different techniques. The two bars per workload report weighted speedups for the two techniques, and the lines show system power consumption. These results show SCR achieves slightly better performance than the LLC resizing technique. In addition, the power savings that SCR provides translates into 4.2% power savings from the system power perspective.

Effectiveness of Prioritized Way Selection We compare the weighted speedup and the system power consumption of PWS to the *Independent* scheme. The disadvantage of the *Independent* scheme compared to PWS is proportionally aggregated performance loss and sub-optimal power savings. Figure 8 shows the cache-hierarchy power consumption and weighted speedups of PWS and *Independent*. PWS outperforms Independent by as much as 2.2% in performance and 2.6% in power savings in *G2-7*. There are a few workloads where the performance of PWS is lower than that of *Independent*, but the power savings of PWS is much more sig-



(a) SCR Cache-Hierarchy Power Consumption Breakdown and LLC Resizing Cache-Hierarchy Power Consumption



(b) Weighted Speedup and System Power Consumption

Figure 6: Power consumption and performance of dynamic SCR and LLC resizing.

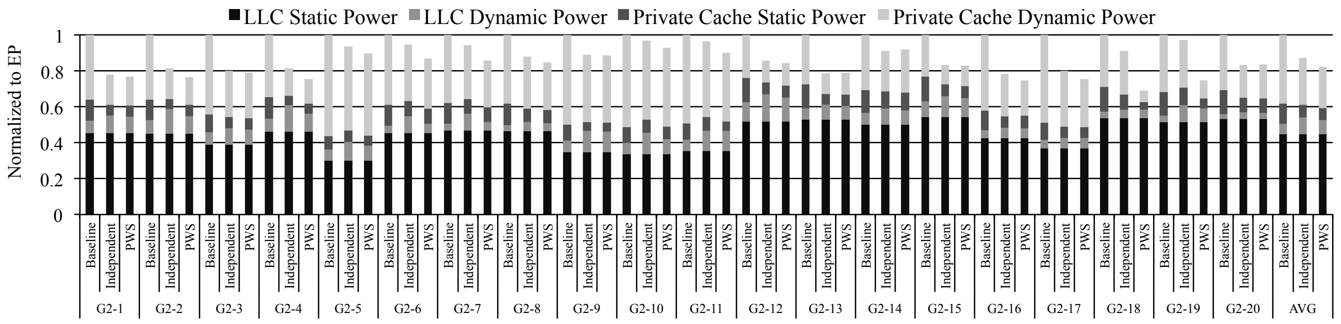
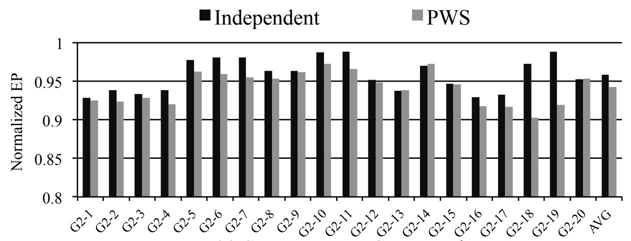
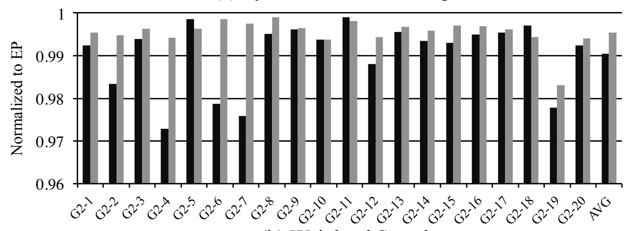


Figure 7: PWS effectiveness (breakdown of cache-hierarchy power consumption for two-application workloads).



(a) System Power Consumption



(b) Weighted Speedup

Figure 8: PWS effectiveness.

nificant in those cases. For example, the power savings of PWS over *Independent* in *G2-18* is 7.2% while its performance loss is only 0.3%. On average, PWS saves system power by 1.7% and improves performance by 0.5% over the *Independent* scheme.

The power savings of PWS compared to *Independent* is much more significant when we compare just the dynamic power of the private caches. Figure 7 shows the cache-hierarchy power consumption for PWS and *Independent*. When we compare the private-cache dynamic power consumption for the two techniques in Figure 7, PWS saves as much as 73% of the private-cache dynamic power compared to *Independent* in *G2-18*. On average, PWS saves the dynamic power in the private caches by 12% and the overall cache-hierarchy power by 5.9%.

Effectiveness of LLC-Partitioning-aware PWS Private cache resizing should be *symbiotic* in that aggressive resizing of the private caches may increase total power consumption of the cache hierarchy by increasing LLC access energy more than the power savings from the private cache resizing. We compare the weighted speedup and the system power consumption of LP-PWS to PWS, which does not consider the power consumption of the LLC. Figure 10 shows the system

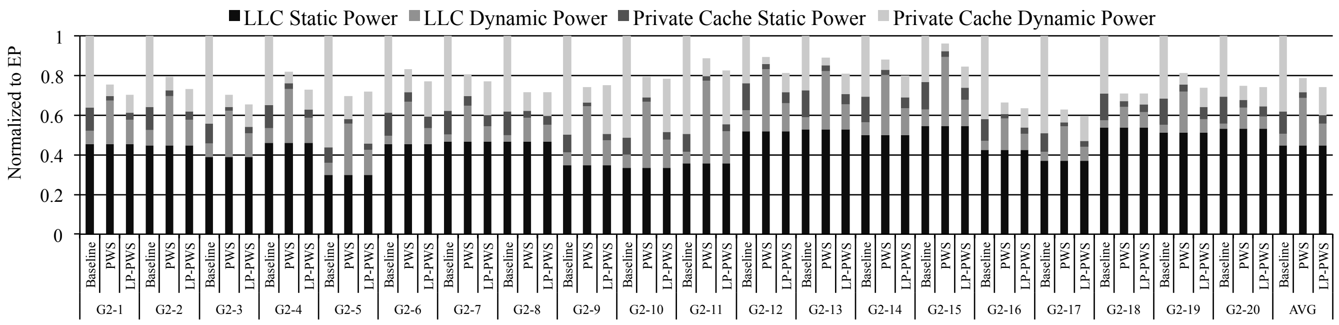


Figure 9: LP-PWS effectiveness (breakdown of cache-hierarchy power consumption for two-application workloads).

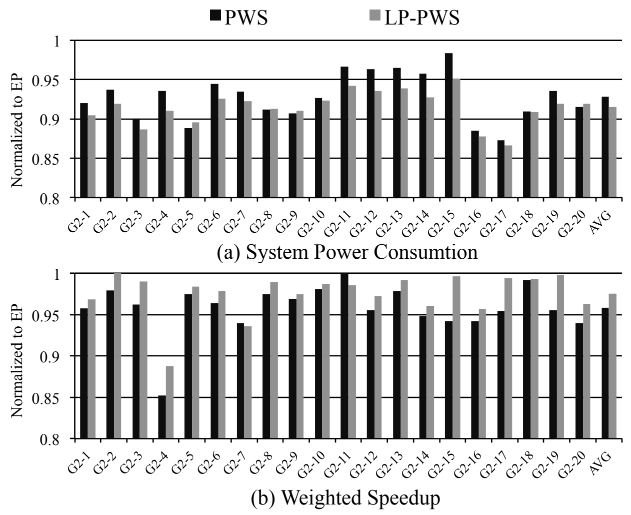


Figure 10: LP-PWS effectiveness ($T_{Priv} = 3$).

power consumption and weighted speedup of PWS and LP-PWS. The weighted speedup of PWS degrades significantly by allowing additional cache misses with *threshold* of 3 to save power aggressively. Nevertheless, PWS still does not save as much power as LP-PWS, which manages less performance degradation.

Aggressive PWS can harm the weighted speedup as much as 15% in *G2-4*, while achieving 6.4% total system power savings. For *G2-4* LP-PWS shows weighted speedup degradation of only 11% with power saving of 9%, which can be translated into 4% performance improvement and 2.7% power savings over PWS. On average, LP-PWS saves system power by 1.4% and improves performance by 1.8% over the PWS scheme. Figure 9 shows the power consumption of the cache hierarchy of PWS and LP-PWS. PWS saves dynamic power in the private caches by as much as 67% compared to LP-PWS, but it consumes 250% of the LLC dynamic power compared to LP-PWS in *G2-14*, resulting in 7.5% additional power consumption in the cache hierarchy. On average, LP-PWS saves the cache hierarchy power consumption by 5.7% consuming more dynamic power in the private caches by 99%, but less dynamic power in the LLC by 53% compared to PWS.

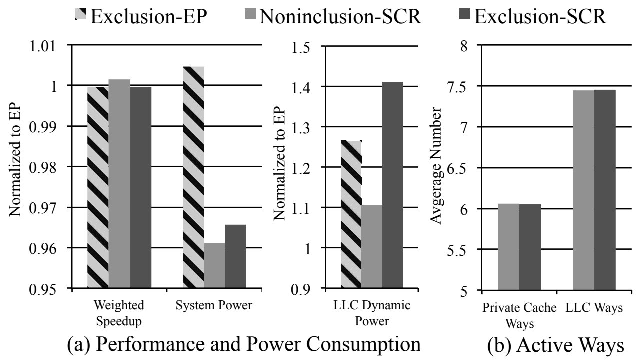


Figure 11: SCR with non-inclusive and exclusive LLC.

Non-Inclusive vs. Exclusive LLC The decision to use non-inclusive or exclusive LLCs is non-trivial because exclusive LLCs are more efficient in cache-capacity utilization while non-inclusive LLCs are more efficient in the on-chip bandwidth utilization and dynamic power. Moreover, which inclusion property is more beneficial depends on the workload. For this reason, the exclusive LLC with even partitioning shows slightly lower performance than the noninclusive LLC. As Figure 11 shows, there is marginal performance degradation in the exclusive LLC, unlike the performance improvement in the noninclusive LLC. The exclusive LLC shows higher DRAM-access increase, after applying SCR, 3.5% compared to 0.17% decrease of the noninclusive LLC, explaining the slight performance degradation in the exclusive LLC and the performance improvement in the noninclusive LLC.

The dynamic power consumption of the exclusive LLC is higher by 27.5% compared to the inclusive LLC, contributing to the higher system power. Although the noninclusive LLC shows lower system power consumption, the ratios compared to even partitioning are almost identical –0.961–implying that SCR saves around 4% of system power consumption in both cases. We note that SCR incurs bigger dynamic power increase in the exclusive LLC by 11.5% compared to 10.7% in the noninclusive LLC, but its overall impact on system power is almost negligible. The average numbers of activated ways, after applying SCR, are virtually the same too.

Epoch Size We ran experiments with 0.5M, 1M and 2.5M–cycle epoch size for private cache resizing to measure sensitivities of results to the epochs size while the epoch size for the shared LLC is fixed at 5M cycles like other studies [29, 36]. We found that 1M–cycle epoch size is a reasonable choice for generating good power savings compared to the cache reconfiguration overhead.

Impact of Thresholds SCR relies on two default thresholds, T_{Priv} and T_{LLC}, to control the power savings over the performance loss for the private-cache resizing and the shared-LLC resizing, respectively. We explored ranges of thresholds between 0 to 0.1 for T_{Priv} and 0 to 0.15 for T_{LLC}–both in 5 steps. Setting the threshold to 0 results in the best performance without any power savings for both techniques (they degenerate into the UCP technique as both thresholds are set to 0). On the other hand, we can save large amounts of power as the threshold increases at the cost of performance loss. Although we observed interesting trade-offs between power savings and performance by changing the thresholds, we fixed the thresholds to $T_{Priv} = 0.02$ and $T_{LLC} = 0.12$ for our dynamic SCR experiments in 2, 4 and 8-core CMPs.

6. Related Work

A large body of work exists on cache resizing. Selective cache ways [1] uses off-line profiling to drive disabling of cache ways for dynamic power savings. DRI caches [28, 41, 40] use cache-miss counts to detect over-provisioning, and resize across either cache sets or ways. In addition, DRI caches also gate the power supply to unused portions of cache, conserving both dynamic and static power. Malik *et al* [24] study selective ways in the MCore CPU. All of these prior studies consider resizing a single level of cache only in uniprocessors, whereas SCR addresses the problem of resizing multiple levels of caches in CMPs. In particular, we develop novel algorithms for solving an *NP-hard* problem in $O(N^2)$ time complexity with a *greedy* approach.

Cache partitioning explicitly allocates shared cache across multiprogrammed workloads, providing cache to those programs that can best utilize it. The majority of techniques focus on performance [6, 29, 19, 33, 34, 38]. More recently, techniques have also tried to reduce power consumption [14, 35, 36] by withholding allocation and shutting down portions of the shared cache, similar to cache resizing. Madan *et al* [23] propose resizing L2 caches by dynamically extending their capacity into stacked DRAM. Like SCR, cache partitioning also employs reuse distance profiles to drive allocation decisions. But LLC cache partitioning saves mostly static power consumption compared to SCR which resizes private caches where dynamic power dominates.

Balasubramonian *et al* [3, 4] propose resizing two levels of cache, either the L1/L2 or the L2/L3, by partitioning a common pool of SRAM arrays to different caching levels. Because partitionings always utilize all of the available SRAM, only one cache's size is controlled independently. Hence, in this technique, it is impossible to optimize the balance point of different caching levels simultaneously as is done in SCR. Moreover, the technique is only limited to uniprocessors. Wang *et al* [39] propose private cache resizing in conjunction with LLC partitioning. This technique requires off-line profiling. Besides, it adopts LLC partitioning only, losing the opportunity to save static power in the LLC.

Besides resizing, researchers have studied other adaptive cache techniques as well. Dropsho *et al* [7] propose *accounting caches* which divide a cache's ways into primary and secondary groups. Each cache access searches the two groups sequentially, accessing the secondary only on a primary miss. This saves power if secondary accesses are infrequent. Zhang *et al* [43] propose *way concatenation* which permits flexible organization of cache banks to form direct-mapped, 2-way, or 4-way set-associative caches. Neither accounting caches nor way concatenation address capacity allocation across different levels of cache, as done in SCR.

Silva-Filho *et al* [32] and Gordon-Ross *et al* [11] study design-time techniques for optimizing 2-level cache hierarchies. This body of work tries to find the best block size and associativity–as well as cache capacity–for two caching levels. They consider a more complex design space than we do, and employ more costly search techniques that are suitable for design analysis only. In contrast, SCR is an architecture-level power management technique. It solves a more constrained problem, but provides algorithms suitable for run-time use. Similarly, Zhang and Vahid [42] search for the best cache architecture using a reconfigurable hardware platform. But they only consider optimizing a single level of cache.

Finally, significant research has explored circuit-level techniques for reducing a cache's static power consumption. Multi-V_t techniques [2, 17] employ low-V_t devices along critical paths and high-V_t devices along non-critical paths to save power while still maintaining performance. Similarly, super high-V_t devices have been explored in [15]. Gated-V_{DD} [28] uses high-V_t devices to gate the power supply to unused portions of cache. Adaptive body bias [16, 27] and Forward body bias [37] control the back-gate voltage to place devices in a standby low-leakage mode when not in use, but then restores the devices to an active high-performance mode when the cache is accessed. Lastly, dynamic voltage scaling [10, 18] can similarly transition between standby and active modes by scaling the supply voltage. Similar to other cache resizing techniques [41, 40], SCR relies on Gated-V_{DD} to essentially eliminate leakage for unused portions of the cache.

7. Conclusion

This paper presents SCR, an architecture-level power management technique that resizes all caches in a modern CMP cache hierarchy. Our work shows a static-optimal version of SCR can reduce total power dissipation in the on-chip cache hierarchy by 2.5% while boosting performance by 0.7% across two-application workloads compared to the LLC resizing technique. We find that significant power savings comes from *symbiosis* of performing the private cache and the shared

LLC resizing simultaneously. Our work also develops the SCR algorithms which employ a *greedy* approach to find *pseudo*-(*Pareto*) optimal solutions at runtime in a scalable fashion. We show dynamic SCR can achieve between 2.6–54.5% power savings in the cache hierarchy while achieving performance boost up to15.8%.

References

[1] D. H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation," in *Proceedings of the 32nd Annual International Symposium on Microarchitecture*, November 1999, pp. 248–259.

[2] R. Bai, N.-S. Kim, D. Sylvester, and T. Mudge, "Total Leakage Optimization Strategies for Multi-Level Caches," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*, 2005, pp. Chicago, IL.

[3] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Dynamic Memory Hierarchy Performance Optimization," in *Proceedings of the workshop on Solving the Memory Wall Problem*, June 2000.

[4] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "A Dynamically Tunable Memory Hierarchy," *IEEE Transactions on Computers*, vol. 52, no. 10, pp. 1243–1258, October 2003.

[5] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," University of Wisconsin-Madison, CS TR 1342, June 1997.

[6] J. Chang and G. S. Sohi, "Cooperative Cache Partitioning for Chip Multiprocessors," in *Proceedings of the International Conference on Supercomputing*, Seattle, WA, June 2007.

[7] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scott, "Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power," in *Proceedings of 11th Annual International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[8] EmuVM, "AlphaVM-free, version 1.0.2 for Windows 7. Available at," http://www.emuvm.com/downloads.php.

[9] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 13–23.

[10] K. Flautner, nam Sung Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *Proceedings of the International Symposium on Computer Architecture*, Anchorage, AK, May 2002.

[11] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic Tuning of Two-Level Caches to Embedded Applications," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 04)*, 2004.

[12] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[13] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and More Flexible Program Analysis," in *Proceedings of the Workshop on Modeling, Benchmarking and Simulation*, June 2005.

[14] K. Kedzierski, F. J. Cazorla, R. Gioiosa, A. Buyuktosunoglu, and M. Valero, "Power and Performance Aware Reconfigurable Cache for CMPs," in *Proceedings of the Second International Forum on Next-Generation Multicore/Manycore Technologies*, Saint-Malo, France, June 2010.

[15] C. Kim, J.-J. Kim, S. Mukhopadhyay, and K. Roy, "A forward body-biased low-leakage SRAM cache: device, circuit and architecture considerations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 349–357, 2005.

[16] C. H. Kim and K. Roy, "Dynamic Vth Scaling Scheme for Active Leakage Power Reduction," in *Proceedings of the International Symposium on Design, Automation, and Test in Europe*, 2002, pp. 163–167.

[17] N. S. Kim, D. Blaauw, and T. Mudge, "Leakage Power Optimization Techniques for Ultra Deep Sub-Micron Multi-Level Caches," in *Proceedings of the International Conference on Computer-Aided Design*, 2003.

[18] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power," *IEEE Transactions on Very Large Scale Integration*, vol. 12, no. 2, pp. 167–184, February 2004.

[19] S. Kim, D. Chandra, and Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2002.

[20] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems."

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 469–480.

[22] W. Liu and D. Yeung, "Using aggressor thread information to improve shared cache management for cmps," in *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 372–383.

[23] N. Madan, L. Zhao, naveen Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2009.

[24] A. Malik, B. Moyer, and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Rapallo, Italy, 2000.

[25] "ITRS Working Group Models, MASTAR," http://www.itrs.net/models.html, 2011.

[26] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE*. IEEE Computer Society, 2007, pp. 3–14.

[27] K. Nii, H. Makino, Y. Tujihashi, C. Morishima, Y. Hayakawa, H. Nunogami, T. Arakawa, and H. Hamano, "A Low Power SRAM using Auto-Backgate-Controlled MT-CMOS," in *Proceedings of the International Symposium on Low-Power Electronics and Design*, August 1998, pp. Monterey, CA.

[28] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics & Design*, 2000, pp. 90–95.

[29] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 423–432.

[30] D. Sanchez and C. Kozyrakis, "Vantage: scalable and efficient fine-grain cache partitioning," in *Proceedings of the 38th annual international symposium on Computer architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 57–68.

[31] N. Shukla, R. Singh, and M. Pattanaik, "Design and Analysis of a Novel Low-Power SRAM Bit-Cell Structure at Deep-Sub-Micron CMOS Technology for Mobile Multimedia Applications," *International Journal of Advanced . . .*, 2011.

[32] A. G. Silva-Filho and F. R. Cordeiro, "A Combined Optimization Method for Tuning Two-Level Memory Hierarcnhy Considering Energy Consumption," *EURASIP Journal on Embedded Systems*, vol. 2011, September 2010.

[33] G. E. Suh, S. Devadas, and L. Rudolph, "A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2002.

[34] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic Partitioning of Shared Cache Memory," *The Journal of Supercomputing*, vol. 28, no. 7-26, 2004.

[35] K. T. Sundararajan, V. Porpodas, T. M. Jones, M. P. Topham, and B. Franke, "Cooperative Partitioning: Energy-Efficient Cache Partitioning for High-Performance CMPs," in *Proceedings of the 18th International Symposium on High-Performance Computer Architecture*, New Orleans, LA, February 2012.

[36] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke, "Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps," in *HPCA*, 2012, pp. 311–322.

[37] J. Tschanz, S. Narendra, Y. Ye, B. Bloechel, S. Borkar, and V. De, "Dynamic sleep transistor and body bias for active leakage power control of microprocessors," *Solid-State Circuits, IEEE Journal of*, vol. 38, no. 11, pp. 1838–1845, 2003.

[38] K. Varadarajan, S. K. Nandy, V. Sharda, and A. Bharadwaj, "Molecular Caches: A Caching Structure for Dynamic Creation of Application-Specific Heterogeneous Cache Regions," in *Proceedings of the International Symposium on Microarchitecture*, 2006.

[39] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 948–953.

[40] S.-H. Yang, M. D. Powell, B. Falsafi, and T. N. Vijaykumar, "Exploiting Choice in Resizable Cache Design to Optimize Deep-Submicron Processor Energy-Delay," in *Proceedings of the 29th International Symposium on Computer Architecture*, San Diego, CA, June 2003.

[41] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar, "An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001.

[42] C. Zhang and F. Vahid, "Cache Configuration Exploration on Prototyping Platforms," in *Proceedings of the 14th International Workshop on Rapid Systems Prototyping*, 2003.

[43] C. Zhang, F. Vahid, and W. Najjar, "A Highly Configurable Cache Architecture for Embedded Systems," in *Proceedings of the 30th International Symposium on Computer Architecture*, San Diego, CA, June 2003.