Cache Power Budgeting for Performance

Rathijit Sen
rathijit@cs.wisc.edu

David A. Wood
david@cs.wisc.edu

Department of Computer Sciences
University of Wisconsin-Madison

ABSTRACT

Power is arguably *the* critical resource in computer system design today. In this work, we focus on maximizing performance of a chip multiprocessor (CMP) system, for a given power budget, by developing techniques to budget power between processor cores and caches. Dynamic cache configuration can reduce cache capacity and associativity, thereby freeing up chip power, but may increase the miss rate (and potentially memory power). Dynamic voltage and frequency scaling (DVFS) can exploit the saved power to increase core performance, potentially increasing system performance. Detailed simulation models show that carefully budgeting power between cores and caches can *improve* system performance 2-16% for 11 of 17 workloads.

To intelligently budget power between cores and caches, we investigate using hardware support to drive analytical models of system power and performance. Online estimation enables real-time feedback and adaptation to dynamic changes such as operating system interactions or changing workload mixes. We demonstrate an integrated online framework that combines a cache reuse model, performance model, power model and DVFS model to identify optimal power-budgeted configurations.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Cache memories
; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Performance

Keywords

Cache, LLC, Stack Distance, Reuse Distance, Working Set, PLRU, Power Budget, DVFS, Static Power

1. INTRODUCTION

Performance, power and energy consumption are all first-class system design constraints today. However, in many applications system power consumption has become the dominant constraint. For example, the Intel Nehalem processors cannot simultaneously run all cores at the maximum frequency and voltage level without exceeding their power envelope [14]. This trend will likely continue as technology scales [54], making power the most critical resource. Given a power envelope, the system designer has to choose how to budget this resource among various system components.

Caches improve performance by reducing the effective memory access latency, but consume significant static and dynamic power. Just as caches cannot be simultaneously large and fast, they also cannot be both large and low power. Smaller caches consume less static power, but can degrade performance and increase dynamic power due to more misses. There is thus a tradeoff between performance and power consumption, which depends on the currently executing set of workloads and data sets. Previous work has shown that workloads often have critical working sets [55], and by reducing the cache size to just hold the working set it is often possible to save power without a significant performance loss [5]. The recent Ivy Bridge microarchitecture powers down a subset of cache ways during periods of low activity [26].

In this paper, we quantitatively show that *it is possible to increase system performance by decreasing cache size*, given a fixed power budget. The key insight is to treat power as a critical resource that should be budgeted wisely to maximize performance. Specifically, power should be distributed between cores and caches to maximize performance. Cache power can be controlled using dynamic cache reconfiguration and core power can be managed using dynamic voltage and frequency scaling (DVFS) techniques [25, 42]. By striking a balance between cores and caches, we can optimize system performance for a given power budget. This work addresses performance maximization for both on-chip and system power budgets. Improved performance at the same power also translates into energy savings.

Figure 1 illustrates the opportunity by showing the average power breakdowns for our workloads, running on an 8-core CMP system with a dynamically reconfigurable cache (details in Section 3). For `blackscholes`, using a 2MB 2-way cache saves 21.7% of system power while incurring a negligible performance penalty. Figure 8 (Section 7) shows the results of power budgeting for system power – on-chip frequency can be scaled by 16.1% for `blackscholes` to get a performance gain of 15.9% while still saving system power by 2.2%. Note that DVFS increases dynamic power and indirectly, also static power due to temperature rise. This can be observed by comparing the leftmost power stacks in Figures 1 and 8.

Power budgeting must be done carefully, however, as a poor choice of cache configuration can drastically degrade performance. As Figure 1 shows, a small cache for `jbb` reduces performance by 38.8% due to increased memory accesses. Further, using a smaller cache does not always save

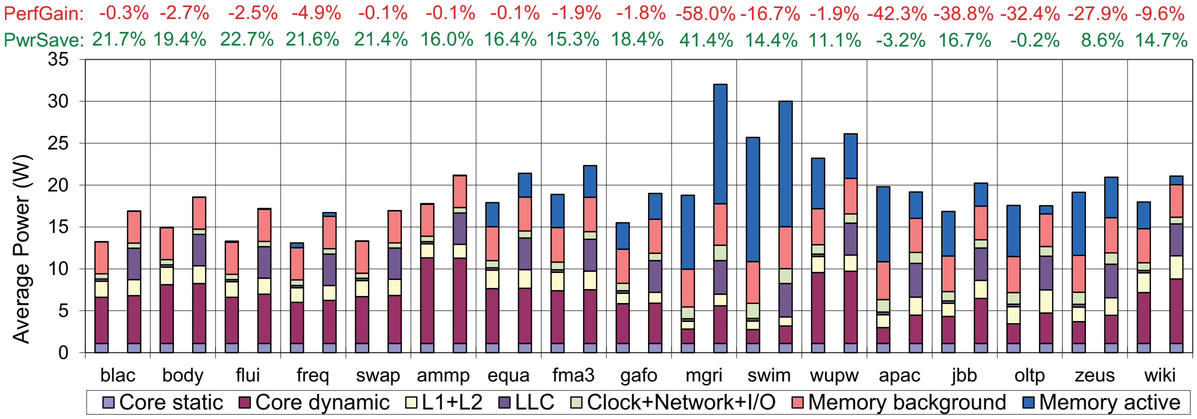


Figure 1: Two power stacks are shown for each workload: the left stack for the lowest-power LLC (2MB 2-way) and the right stack for the highest-power LLC (baseline configuration, 32MB 32-way). Power consumed by the 2MB LLC is not conspicuous as it constitutes a small percentage of total power. The topmost secondary horizontal axis (PerfGain) shows performance gain and the next secondary horizontal axis (PwrSave) shows *system power* saved with the 2MB 2-way cache with respect to the baseline. Core power includes ROBs, bypass networks, functional units, register files, TLBs, branch predictors, fetch & decode logic. We assume aggressive clock-gating. See Section 3 for details on system model.

system power – `apache` burns 3.2% more power as well as runs 42.3% slower. Selecting a cache configuration that optimizes power-efficient performance is challenging since the optimal point varies for different workloads, and even for different phases within a given workload. Exhaustive evaluation is impractical due to the large number of feasible system configurations.

In this work, we use analytical models to estimate the power and performance of different configurations, allowing rapid prediction of the optimal system configuration. This requires four predictors: cache miss rate predictor, performance impact predictor, power impact predictor, optimal DVFS scaling predictor. We evaluate two techniques for predicting cache miss rates online: way counters [28, 39, 49] and reuse sampling [41]. The miss rate predictions are combined with simple performance and power models (Section 5) and an on-chip DVFS model (Section 6) to identify power-budgeted configurations that will maximize performance.

We extend the current state of the art in two ways:

1. We show that careful cache power budgeting—using DVFS and cache reconfiguration— driven by *low-overhead* predictors can *improve* performance, not just save power. For our target system, budgeting for system power improves performance by 2-16% for 11 of 17 workloads.

2. We develop the first online analytic power and performance models for reconfigurable caches that work for practical replacement policies (i.e., PLRU not just true LRU), do not use shadow tags, and can configure up to larger caches, not just down to smaller ones.

Section 2 presents a big-picture view of our power-budgeting approach. Section 3 describes the system model used for our evaluation. Section 4 presents the analytical cache model. Section 5 presents the models used to predict performance and power. Section 6 combines these with DVFS models to predict optimal system configurations. Section 7 discusses the bottom line results, misprediction, thermal and sensitivity issues. Section 8 summarizes related work.

2. OPERATIONS OVERVIEW

This work focuses on *improving* performance by shifting power from the last-level cache to the cores. For cores, we use conventional on-chip DVFS techniques, similar to those used in Intel's Turbo Boost [52], to run cores at higher voltage and frequency. For caches, we use power-gating [35] to eliminate all static and dynamic power for disabled cache banks/ways.

Cache reconfiguration overhead: While power-gating can enable a higher power margin for utilization, it results in dirty data being written back to memory when downsizing the cache and non-trivial warmup time after up-sizing the cache. Assuming a sustained memory bandwidth of 8.53 GBPS (half of our max. bandwidth) a worstcase scenario with 32MB of dirty data needs ∼3.8 msec writeback time. To keep performance and energy overheads small ($< 1\%$) the reconfiguration interval should be at least 380 msec. *This automatically precludes reacting to high frequency events such as context switches that may occur between 300 to 5000 times per second* [18]. In contrast, drowsy mode [22] can enable small reconfiguration intervals with low overhead, but its need to maintain a minimum data retention voltage (DRV) limits the available power margin. Our models predict that maximum scaling decreases from 16% to 11% if drowsy mode reduces the cache power margin by one-third.

The predictor approach: Reconfiguration being *costly* and *infrequent*, trial-and-error search for the optimal configuration at runtime is not appealing. Stepwise adaptation [36] may progress through multiple intermediate states. In contrast, this work uses online power-performance predictors that enable *one-shot* reconfiguration. They have the following strengths:

- Ability to predict performance for caches larger or smaller than the currently active configuration so that the optimal configuration can be reached in one step.

- Ability to predict the effect of changes in both number of sets and ways (i.e., associativity) of the cache.

1. Concurrently,
   - (a) Specialized hardware (Section 4.2) tracks reuse distance distribution for L3 accesses.
   - (b) Simple hardware performance counters track activity factors of cores and caches.
   - (c) Simple hardware performance counters track correlation between miss rate and CPI.
2. Periodically (e.g., once per second), invoke a software routine (ISR) to determine optimal cache configuration:
   - (a) Predict miss rates (Section 4) using information from 1(a).
   - (b) Predict performance and power (Section 5) using information from 1(b), 1(c), 2(a).
   - (c) Predict DVFS scaling and possible gain (Section 6) using information from 1(c), 2(a), 2(b).
3. Reconfigure system with the best predicted configuration if predicted gain is > 2%. Write back dirty cache blocks as needed.
4. System continues to monitor predicted and actual performance and power metrics to detect and adjust in case of incorrect predictions (e.g., due to phase change effects).

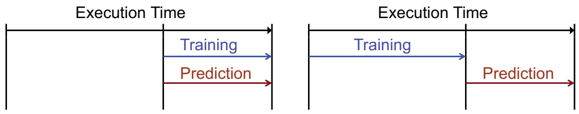
Figure 2: Operations Overview

- Ability to work with implementable cache replacement policies, such as PLRU, RANDOM, NMRU. Prior work [33] has assumed LRU replacement, which is impractical to implement for highly-associative caches.

Train and predict: Figure 2 shows an operational overview of our proposed system. Execution time is logically partitioned into intervals. Like most predictors, our work uses past execution behavior to predict behavior in subsequent intervals. Simple hardware mechanisms are used to observe execution characteristics that are then used by prediction software (ISR) to determine the optimal system configuration. We refer to the interval used to train the predictors as the *training interval* and the interval where the results of prediction are applied as the *prediction interval*.

Interval Length: Intervals should be long enough so that predictor and reconfiguration overheads are small ($< 1\%$). Predictor compute time depends on the number of target cache configurations evaluated. This has two components – miss-rate prediction (~ 0.03 msec/config, see Section 4.2) and DVFS scaling computation (≤ 0.06 msec/config, see Section 6.4). Together with cache reconfiguration overhead (< 3.8 msec), the time overhead with 25 possible target cache configurations is $< 3.8 + 25 * (0.03 + 0.06) = \sim 6$ msec. A small amount of additional time is needed to read various performance counters. So, we recommend an interval length >600 msec. Simple heuristics can be used to reduce the number of evaluated configurations depending on the available time budget. *Our work focuses on optimizing for the average execution profile over this interval.* Predictor power overheads are negligible (Section 4.2).

Error sources: There are two sources of error in the predicted values: *model error* and *phase error*. Model error results from simplifying assumptions (e.g., independence and identical distribution) that may not strictly hold in practice. Phase error results from changes in workload behavior as it moves though different phases of execution [43]. This distinction helps identify which errors could be reduced by further refining the model and which errors are orthogonal to it. When phase changes occur the behavior from a previous interval is not a good predictor for the next interval. The predictors may be improved using previously proposed online phase detection mechanisms [34].



(a) **CI Prediction with Model Error**

(b) **NI Prediction with Model + Phase Error**

Figure 3: Training and Prediction intervals

Prediction terminology: A *current-interval* or *CI* prediction uses the same interval for training and prediction and hence incurs model error alone. *CI* predictions offer insight into model accuracy, but are useless for reconfiguration purposes. A *next-interval* or *NI* prediction uses a preceding training interval to predict the behavior of a subsequent prediction interval and includes both model and phase errors. These predictions are used to reconfigure the system. Figure 3 illustrates training and prediction intervals.

3. SYSTEM MODEL

Table 1 describes the 8-core CMP we use in this study. We assume an 8-banked L3 cache that is dynamically reconfigurable, with capacities ranging from 2MB to 32MB and associativities of 2, 4, 8, 16 and 32 ways, for a total of 25 configurations. The access latency in cycles is conservatively assumed to be constant for all configurations. To evaluate power and performance, we perform *full-system* simulation using GEMS [31] augmented with a detailed timing and power model. We use CACTI 5.3 [45] to determine the static power and dynamic activation energy per component. Power-budgeting results depend on the *relative* proportion of LLC power to core power.

LLC index hashing: We use a simple XOR-based hashing function to distribute lines more uniformly among the L3 cache sets [16, 17]. This usually improves performance over the conventional bit-selection index function and also makes cache modeling easier by making the distribution more uniform random. Due to the self-canceling property of XOR, no extra tag bits need to be stored to retrieve original addresses, if necessary, from hashed values. Many real systems [15, 46, 51] use hashing functions to reduce conflict misses.

Workloads: We use 7 SPEComp [7] benchmarks (**`ammp`**, **`equake`**, **`fma3d`**, **`gafort`**, **`mgrid`**, **`swim`**, **`wupwise`**) with ”ref” inputs, 5 PARSEC [10] benchmarks (**`blackscholes`**, **`bodytrack`**, **`fluidanimate`**, **`freqmine`**, **`swaptions`**) with ”simlarge” inputs, 4 Wisconsin commercial workloads [3] (**`apache`**, **`jbb`**, **`oltp`**, **`zeus`**), and **`wiki`** which uses Apache Lucene [1] and Wikipedia search queries [53] to search a snapshot of the English Wikipedia [2]. Each workload uses 8 threads and runs

Core configuration	4-wide out-of-order, 128-entry window, 32-entry scheduler				
Functional Units	4 integer, 2 floating-point, 2 mem units				
Branch Prediction	YAGS 4K PHT 2K Exception Table, 2KB BTB, 16-entry RAS				
Disambiguation	NoSQ 1024-entry predictor, 1024-entry double-buffered SSBF				
Fetch	32-entry buffer, Min. 7 cycles fetch-dispatch time				
Number of cores	8	**Max. Aggregate IPC**	$4 \times 8 = 32$	**Min. Aggregate CPI**	1/32
L1I Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP				
L1D Cache	private 32KB 4-way per core, 2 cycle hit latency, ITRS-HP				
L2 Cache	private 256KB 8-way per core, 6 cycle access latency, PLRU, ITRS-LOP				
L3 Cache	shared, configurable 2MB 2-way - 32MB 32-way, 8 banks, 14 cycle access latency, PLRU, ITRS-LOP, serial				
Coherence protocol	MESI (`Modified`, `Exclusive`, `Shared`, `Invalid`), directory				
On-Chip Interconnect	2D Mesh, 16B bidirectional links				
On-chip frequency	2132-2665 MHz	**Technology generation**	32nm	**Temperature**	340K-348K
Main Memory	4GB DDR3-1066, 75ns zero-load off-chip latency, 2 memory controllers, 36 banks, closed page, pre-stdby				

Table 1: System configuration

workload	**blac**	**body**	**flui**	**freq**	**swap**	**ammp**	**equa**	**fma3**	**gafo**	**mgri**	**swim**	**wupw**	**apac**	**jbb**	**oltp**	**zeus**	**wiki**
MPKI	0.002	0.003	0.094	0.400	< 0.001	0.010	3.073	2.224	2.759	11.268	22.936	2.849	4.093	1.693	1.055	6.454	0.324

Table 2: Workload miss-rates. The MPKI shown is for the prediction interval (also see Figure 3) and a 32MB 32-way LLC.

for a fixed amount of work (e.g. #transactions or loop iterations [4]) that is logically partitioned into adjacent training and prediction intervals of ∼250M instructions each. A total of 25 cache configurations were simulated for each workload. Each simulation run starts from a mid-execution checkpoint that includes cache warmup. Simulating a full second of target execution is infeasible due to high simulation overheads of detailed models. Our workloads exhibit a range of miss rates from very small to quite large (Table 2).

4. CACHE MISS-RATE ESTIMATION

Accurately predicting the cache miss rate for all possible cache configurations is critical to our power-budgeting approach. In this section we briefly describe how to estimate miss rates using an established model based on LRU stack distance/reuse distance distribution [9, 19, 32]. *This section elaborates step 2(a) of Figure 2.*

Reuse distance: The reuse distance of an access with line address `a` is the number of unique line addresses accessed between consecutive accesses to `a`. For example, the reuse distance of `a` in the access sequence `a b b c d b a` is 3. (Note that reuse distance is 1 less than Mattson's much earlier stack distance [32]). The reuse distance distribution is a property of a memory access stream and captures information about temporal locality. In this work, we focus on accesses to the L3 cache (i.e., L2 cache misses), thus our reuse distance distributions depend upon the workload and the (fixed) L1/L2 cache configuration, but not the L3 cache configuration. Given a reuse distance distribution, cache miss rates can be estimated using cache-hit functions that are specific to the replacement policy [41]. Predicted miss rates can in turn be used to predict performance and power.

Example distributions: Figure 4 shows reuse distance distributions for our workloads. The dashed vertical lines with size annotations indicate fully-associative LRU cache sizes that would be necessary if all accesses with reuse distances less than or equal to that point must hit. Larger caches may be needed to deal with limited associativity, non-LRU replacement, and an inclusive hierarchy.

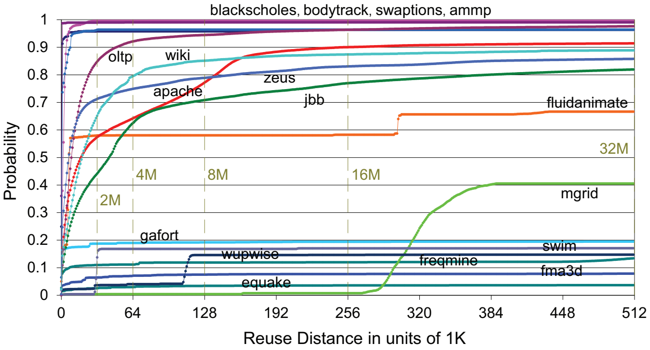


Figure 4: Reuse distributions (cumulative). A 32MB cache has 512K lines.

Workload classification: We categorize workloads into 3 classes depending on the sizes of their working sets in relation to the range of cache sizes that we study.

1. **Cache insensitive (low/medium locality)**: those that have a significant part of their working sets that never fit in the cache (e.g. bottom line in the figure). These have reuse distributions with slopes that change very little over the range of cache sizes, but have a high miss rate, e.g., low: `equake`, medium: `fma3d`, `fluidanimate`, `freqmine`, `gafort`, `swim`, `wupwise`.

2. **Cache insensitive (high locality)**: those whose working sets mostly fit in the cache (top line in the figure). These also have almost flat reuse distributions, but the relative change in small miss rates can be large, e.g., `blackscholes`, `bodytrack`, `swaptions`, `ammp`.

3. **Cache sensitive**: those whose working sets fit significantly more in large caches than in smaller ones (mid-

dle line in the figure). The reuse distribution curves have a significant slope, e.g., commercial workloads, mgrid, wiki.

Cache sensitive workloads incur high performance losses with small caches and require large caches for good performance. In contrast, cache insensitive workloads are good candidates for power-budgeting, since their performance largely does not change for different configurations.

4.1 Estimation accuracy with full information

For our purposes, good miss rate predictions should not have high absolute errors, as this can seriously degrade performance due to erroneous decisions. We consider two prediction techniques: PLRU reuse models [41] and way counters [28, 39, 49], both of which are based on the concept of reuse distances [41].

PLRU reuse models: Figure 5a shows the error in MPKI prediction vs actual values for all workloads using PLRU reuse models [41], assuming that the full distribution is available. Most points concentrated near the x-axis, demonstrating that the model is quite accurate. This is reinforced by Figure 5b that shows cumulative distribution of the absolute values in prediction error. 90% of predictions have absolute error <1 MPKI, with 80% predictions having error <0.2 MPKI for CI prediction and <0.6 MPKI for NI prediction. Errors with NI prediction are somewhat higher than those with CI prediction due to changes in program behavior between the training and the prediction intervals. A few outliers on Figure 5a (between 30-34 MPKI) have high absolute errors. These correspond to swim with a small cache where unexpectedly more misses happen.

Way-counter comparison: Figures 6a and 6b show errors when predicting using (PLRU) way counters [28] assuming full shadow tags [37,39] are maintained for the maximum 32MB 32-way configuration, regardless of the number of data ways actually used. The average magnitude of error is ∼48% higher with way counters compared to reuse model for CI predictions and ∼27% higher for NI predictions. The difference is due to the different algorithms used by the two methods to estimate miss-rates for PLRU. The two methods make identical estimates for LRU [41]. Note, that there are only 5 points, *one per cache size*, (32MB 32-way, 16MB 16-way, 8MB 8-way, 4MB 4-way, 2MB 2-way) for each workload in each figure, since way counters cannot predict for configurations with different numbers of sets.

4.2 Online estimation method and overheads

The techniques discussed in subsection 4.1 are not directly implementable since maintaining state for obtaining full information is prohibitively costly for large caches. Qureshi, et al. [39] showed that dynamic set-sampling can reduce shadow tag overheads. However, the constraint on limited reconfigurability (fixed number of sets) remains. As an alternative, in this work we use reuse sampling [41], that uses Bloom filters, set- and time-sampling to approximate reuse distances, and then apply PLRU hit functions to the estimated distribution. This method does not use shadow tags and can handle both set- and way-reconfigurability of the cache. We use two 1024-bit Bloom filters in parallel. This allows a maximum reuse distance of 512, assuming that at most half of each filter is allowed to be full.

Accuracy: We use the average reuse distance distribution obtained using the 2 Bloom filters over all contiguous set samples as a proxy for the sampled distribution. A practical implementation would additionally experience inter-sample variation, but we expect it to be small for long runs. With this, 90% of NI predictions have absolute error <1.13 MPKI and 80% predictions have error <0.66 MPKI. The errors are higher than those using the full distribution, but we expect small differences with sufficiently long training runs. The remaining results in this paper use the average sample distribution for miss-rate predictions.

Computational overhead: Given the sampled reuse-distance distribution, the time to compute the PLRU hit functions per target cache configuration was ∼0.03 msec, measured on a Nehalem (2.26 GHz) machine.

Power and area overheads: A small SRAM array is used to track the number of times each reuse distance is seen. This is used later to determine the probability distribution. Assuming 4-byte counters for each reuse distance, the size of this array is $512 \times 4 = 2KB$. Similar to recent work [29], we model the Bloom filter arrays and the counter array in CACTI as direct-mapped caches (ITRS-LOP) with 8-byte line sizes. The total static power is ∼1.2 mW and dynamic energy per activation is ∼10.8 pJ leading to a total overhead of < 0.03% of system power. These (negligible) overheads are included in our on-chip and system power calculations. CACTI numbers suggest the area overhead for the arrays to be less than 1 sq mm.

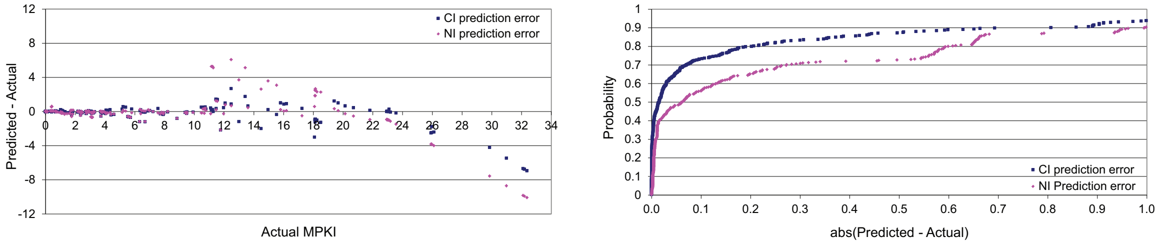
5. PERFORMANCE AND POWER PREDICTION MODELS

Predicting miss rates for target cache configurations is necessary, but not sufficient for making power-budgeting decisions; it is also necessary to predict performance and power impact. *This section elaborates step 2(b) of Figure 2.*

Performance prediction: To predict performance given an L3 miss rate r for an instruction interval, let $\hat{CE}(r)$ denote the estimated number of cycles to complete that interval. $\hat{CE}(r) = \hat{CPI}(r) \times$ number of instructions (aggregate, over all cores). We approximate $\hat{CPI}(r)$ by measuring the actual L3 misses, cycles and instructions committed during the training interval using hardware performance counters and then using least-squares regression to fit this to a simple linear model. This is done using a single run. At the end of every 2M cycles (∼1 msec), two metrics are computed: (Δ LLC misses/Δ instructions committed) and (Δ cycles/Δ instructions committed). The individual tuples are not stored. Instead, cumulative metrics (product, square, sum) with earlier values are computed. 4 registers and 1 counter (for tracking total number of tuples) are maintained. At the end of the training interval, the slope and offset of the best-fit line is computed. The actual values are gathered using the above method from a single run.

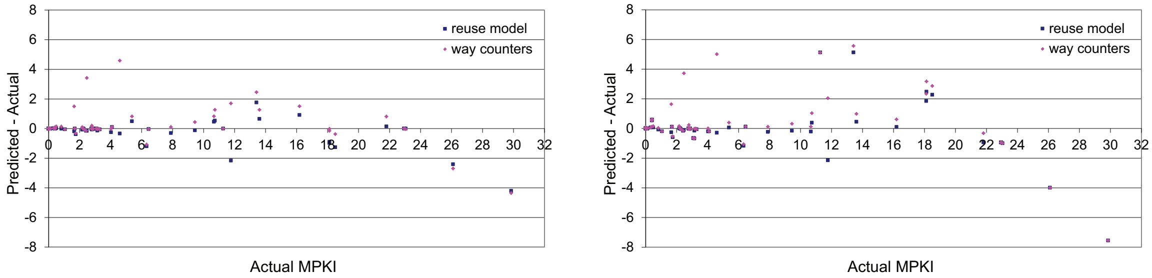
Special cases: The linear approximation does not always succeed. In some cases due to variations in program behavior, low MPKI or lack of variability in the tuples, the slope may be computed as negative. This is unrealistic as we expect execution time to increase with MPKI due to the long off-chip miss latency. In such cases, the computed slope is discarded and instead some predetermined value is chosen. We chose this to be the average value seen for the commercial workloads. Also, we predict CPI to be the same if predicted MPKI is within 0.01 or 1% of the current configuration.

Power prediction: We separately predict static and dy-



(a) **MPKI prediction error.** (b) **Prediction error distribution (cumulative).**

Figure 5: MPKI prediction using reuse model. 25 points per workload. Avg(abs(Predicted-Actual) MPKI) = 0.25(CI), 0.51(NI).



(a) **CI prediction for MPKI.** (b) **NI prediction for MPKI.**

Figure 6: Comparison between Way Counters and Reuse Model for MPKI prediction, without set-sampling. 5 points per workload. Avg(abs(Predicted-Actual) MPKI) = 0.25(CI), 0.55(NI) for reuse model and 0.37(CI), 0.70(NI) for way counters.

namic power, which varies with operating voltage and the L3 cache configuration—as the number of power-gated sets and ways change. We use CACTI to estimate static power and dynamic energy per activation per component. For the L3 cache, the number of tag and data activations for accesses, misses, replacements and coherence activities is tracked. To predict activations, the model makes some simplifying assumptions, e.g., L3 accesses, coherence activities, and the percentage of L3 misses that cause additional writebacks to memory is the same as that observed in the current configuration. These assumptions are not strictly true but work reasonably well. 90% of system power predictions have errors of 14.8% or less. Since static power is known, the errors are due to dynamic power estimations. This has two components: activation count estimation and performance estimation. Improving either will reduce errors.

6. MODEL-DRIVEN POWER BUDGETING

The analytical models of the preceding sections determine the power and performance of different cache configurations for a given workload. Here we describe how to estimate whether or not it is better to reconfigure the cache to a smaller configuration that uses less power, leaving more power available to run the core at a higher voltage and frequency. *This section elaborates step 2(c) of Figure 2.*

Subsection 6.1 presents a basic model for calculating maximum performance gains in any power-budgeting environment where power gating and on-chip DVFS are used for shifting power. We then consider two power-budgeting scenarios: *on-chip* (Subsection 6.2) and *system* (Subsection 6.3) using extended models that include leakage and performance losses. In both cases, we use only on-chip DVFS. Memory voltage and frequency are not scaled.

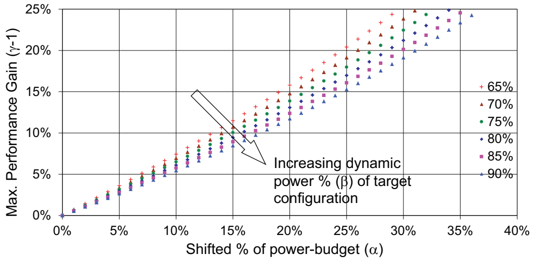


Figure 7: Max. performance gains with basic model (Section 6.1). $D_{old} = \beta(1 - \alpha)P$. Also see Section 7 (sensitivity).

6.1 Basic model

Let P be the power-budget (assumed to be fully utilized by the current configuration) and let α denote the fraction of this budget that can re-budgeted for better performance. The power margin, αP, can include both static *and* dynamic power of the current configuration. Ideally, DVFS transforms all of αP into additional dynamic power in the target configuration. However, there are leakage losses due to temperature and voltage rise and performance losses due to non-scaling of memory latency. So, after DVFS, the dynamic power of the target configuration, $D_{new} \leq D_{old} + \alpha P$. Since dynamic power is proportional to $V^2 f$, we have

$$\frac{D_{new}}{D_{old}} = \left(\frac{V_{new}}{V_{old}}\right)^2 \left(\frac{f_{new}}{f_{old}}\right) \leq \left(1 + \frac{\alpha P}{D_{old}}\right) \qquad (1)$$

Let $\gamma = \frac{f_{new}}{f_{old}}$. So, *Speedup* $\leq \gamma$. Using published data [23] for voltage-frequency pairs for the Pentium M, we assume that $(V_{new} - V_{old}) \propto (f_{new} - f_{old})$ and that every 200MHz change in frequency is accompanied by a 50mV change in voltage. Thus, $V_{new} = V_{old} + 50mV\left(\frac{f_{new}-f_{old}}{200MHz}\right)$. We also assume a base operating voltage of 0.9V and a base operating frequency of 2132MHz. Substituting values in Equation 1,

$$\left(\frac{V_{new}}{V_{old}}\right) = (1 + 0.5922(\gamma - 1)) \tag{2}$$

$$(1 + 0.5922(\gamma - 1))^2\gamma \le \left(1 + \frac{\alpha P}{D_{old}}\right) \tag{3}$$

Let β denote the *dynamic* fraction of power in the target configuration *before* DVFS, i.e. $D_{old} = \beta(1-\alpha)P$. Thus, $(1 + 0.5922(\gamma - 1))^2\gamma \le \left(1 + \frac{\alpha}{\beta(1-\alpha)}\right)$. Figure 7 shows maximum performance gains with $\gamma \le 1.25$. For example, for 20% power re-budgeting, the maximum gain possible is 11.7% to 15.8%.

6.2 On-chip power budgeting model

In this scenario, the requirement is that on-chip power cannot exceed the budget. For on-chip power-budget P, current operating voltage V, frequency f, temperature T and target cache configuration $\bar{C} = (capacity, associativity)$, let $\hat{P}_{st}(\bar{C}, T)$ and $\hat{P}_{dyn}(\bar{C}, V, f)$ denote the estimated on-chip static and dynamic power respectively. The dynamic power before DVFS, $D_{old} = \hat{P}_{dyn}(\bar{C}, V, f)$. The power margin, $\alpha P = P - \hat{P}_{st}(\bar{C}, T) - \hat{P}_{dyn}(\bar{C}, V, f)$ can be utilized by increasing the operating voltage and frequency to V_{new} and f_{new} respectively. However, scaling is accompanied by an increase in static power that reduces the available power margin due to temperature and voltage increase.

Temperature increase: Since chip power-budget is fixed, ideally, overall chip temperature cannot rise (Stefan-Boltzmann law). However, since processor chips are not ideal blackbodies, the higher dynamic power of the cores can cause the operating temperature to rise locally. This in turn increases the static power dissipation by $\Delta\hat{P}_{st}(T) = \hat{P}_{st}(\bar{C}, T_{new}) - \hat{P}_{st}(\bar{C}, T)$. $\Delta\hat{P}_{st}(T)$ depends on γ. For our implementation we assumed a maximum temperature rise of 8 °C corresponding to a maximum scaling of 533MHz (2132MHz to 2665MHz) with 3 °C contributed by every 200MHz [23]. We assumed a continuous scaling domain with temperature rise proportional to scaling. Thus, $\Delta\hat{P}_{st}(T) = (\frac{\gamma-1}{1.25-1}) \times (\hat{P}_{st}(\bar{C}, T+8) - \hat{P}_{st}(\bar{C}, T))$. By default, CACTI allows modeling temperature effects in steps of $10K$. We used linear interpolation to obtain static power at other points. So,

$$\Delta\hat{P}_{st}(T) = \left(\frac{\gamma - 1}{0.25}\right) \times \frac{8}{10} \times (\hat{P}_{st}(\bar{C}, T{+}10) - \hat{P}_{st}(\bar{C}, T)) \tag{4}$$

Voltage increase: The higher operating voltage increases static power dissipation. Assuming a linear model [13], the increase is $\Delta\hat{P}_{st}(V) = \left(\frac{V_{new}}{V_{old}} - 1\right) \times \hat{P}_{st}(\bar{C}, T))$. Combining with Equation 2, we have

$$\Delta\hat{P}_{st}(V) = 0.5922 \times (\gamma - 1) \times \hat{P}_{st}(\bar{C}, T) \tag{5}$$

Both $\Delta\hat{P}_{st}(T)$ and $\Delta\hat{P}_{st}(V)$ reduce αP. Plugging values into Equation 3, we get

$$(1{+}0.5922(\gamma{-}1))^2\gamma \le \left(\frac{P{-}\hat{P}_{st}(\bar{C}, T){-}\Delta\hat{P}_{st}(T){-}\Delta\hat{P}_{st}(V)}{\hat{P}_{dyn}(\bar{C}, V, f)}\right) \tag{6}$$

Correcting for performance loss: Since *memory voltage and frequency are not scaled*, the number of cycles to execute the same task in the scaled configuration is increased due to scaling of memory latency (in terms of processor cycles). This reduces the increase in on-chip dynamic power that Equation 6 assumes. We use this fact to improve Equation 6. With a linear performance predictor model, $\widehat{CPI}(r) = g + r \times h$. After scaling, $CPI(\hat{r})_{scaled} = g + \gamma \times r \times h$. So,

$$(1 + 0.5922(\gamma - 1))^2\gamma \le \left(\frac{g + \gamma \times r \times h}{g + r \times h}\right) \times \left(\frac{P - \hat{P}_{st}(\bar{C}, T) - \Delta\hat{P}_{st}(T) - \Delta\hat{P}_{st}(V)}{\hat{P}_{dyn}(\bar{C}, V, f)}\right) \tag{7}$$

6.3 System power budgeting model

In this scenario, the requirement is that on-chip + memory power cannot exceed the budget. We start from Equation 7, noting that since memory voltage and frequency are not scaled, $\hat{P}_{dyn}(\bar{C}, V, f)$ *still refers to on-chip dynamic power*. However, increase in memory power can reduce the available power budget. Let P', $\hat{P}_{mbp}(\bar{C})$ and $\hat{P}_{map}(\bar{C}, \gamma f)$ denote the available system power-budget, estimated memory background power and estimated memory active power. Equation 7 can now be reformulated as

$$(1 + 0.5922(\gamma - 1))^2\gamma \le \left(\frac{g + \gamma \times r \times h}{g + r \times h}\right) \times \left(\frac{P'{-}\hat{P}_{st}(\bar{C}, T){-}\Delta\hat{P}_{st}(T){-}\Delta\hat{P}_{st}(V){-}\hat{P}_{mbp}(\bar{C}){-}\hat{P}_{map}(\bar{C}, \gamma f)}{\hat{P}_{dyn}(\bar{C}, V, f)}\right) \tag{8}$$

For I instructions, the expected execution time with the target configuration is $I \times \left(\frac{g+\gamma\times r\times h}{\gamma}\right)$ with frequency scaling γ and $I \times (g + r \times h)$ without frequency scaling. So,

$$\hat{P}_{map}(\bar{C}, \gamma f) = \gamma \times \left(\frac{g + r \times h}{g + \gamma \times r \times h}\right) \times \hat{P}_{map}(\bar{C}, f) \tag{9}$$

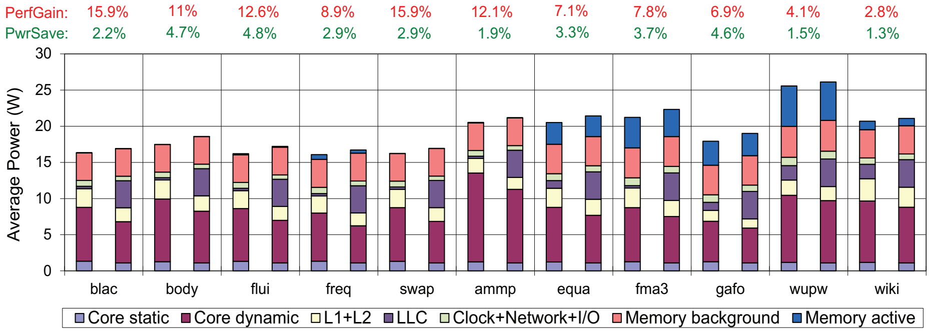
Equation 9 is plugged into Equation 8 for the final equation.

6.4 Computational overhead

Both Equations 7 and 8 can be rearranged in the form $f(\gamma) \le 0$ and solved in software using known techniques for solving cubic equations. To keep overheads low, we recommend using enumeration for $f(\gamma)$: for each allowed frequency step, evaluate $f(\gamma)$ and check the sign of the result. A change in sign between consecutive steps indicates a solution point. Each evaluation takes $\sim 6 \times 10^{-5}$ msec or less on a Nehalem (2.26GHz) machine. Even assuming 1000 possible steps (unlikely) in a real system, the DVFS computation per cache configuration should take $\le$0.06 msec.

7. RESULTS AND DISCUSSIONS

Figure 8 shows the performance gains and power-budget utilization of the best predicted configurations when optimizing performance subject to a system power budget. The baseline is the system with the highest-power LLC (32MB 32-way). The table at the bottom of the figure shows the configurations selected using reuse sampling, change in MPKI and expected temperature rise compared to the baseline. Eleven of seventeen workloads show 1% to 16% performance improvement over the baseline. Cache-insensitive workloads do well, but for the others, significantly reducing LLC capacity is not optimal due to large increase in memory accesses. There is some under-utilization of the baseline power budget (**PwrSave** numbers) due to conservative assumptions



Remaining workloads:

- **mgri, apac, jbb, oltp, zeus**: Predicted config = baseline config. 0% gain.
- **swim**: *Misprediction* leading to 7.5% perf. loss. Online monitoring can detect and revert system to baseline config (0% gain) (See Section 7).

workload	blac	body	flui	freq	swap	ammp	equa	fma3	gafo	wupw	wiki	mgri	swim	apac	jbb	oltp	zeus
cache config	2M 2w*	2M 8w	2M 4w	2M 8w	2M 2w*	2M 8w	8M 4w	2M 8w	8M 4w	16M 16w*	16M 8w	32M 32w*	4M 8w	32M 32w*	32M 32w*	32M 32w*	32M 32w*
Freq scaling	16.1%	11.7%	15.1%	16.2%	16%	11.1%	9.7%	12.1%	12.3%	5.2%	6.3%	NA	10.8%	NA	NA	NA	NA
Temp rise ($^\circ$C)	5.13	3.72	4.85	5.19	5.10	3.56	3.11	3.87	3.93	1.65	2.03	NA	3.47	NA	NA	NA	NA
MPKI increase	0.002	0.002	0.026	0.019	< 0.001	0.001	0.010	0.096	0.023	0.024	0.076	NA	2.314	NA	NA	NA	NA

Figure 8: System power budgeting (Section 6.3) with best predicted configuration (NI prediction). Two power stacks are shown for each workload: the left stack for the predicted configuration and the right stack for the baseline configuration (32MB 32-way LLC, 2132 MHz clock). For each workload, the system power consumed by the baseline configuration is the power budget. The topmost secondary horizontal axis (PerfGain) shows performance gain and the next secondary horizontal axis (PwrSave) shows remaining system power budget with respect to the baseline. 8 of 17 configurations, marked with *, have the same #sets as the baseline. On-chip power budgeting (Section 6.2) also predicts upto 3% gains for `apache`, `jbb`.

about the effect of DVFS on combinational logic and over-prediction of expected power of the target configuration.

Handling mispredictions: Unfortunately, prediction was not accurate for `swim` leading to loss of performance. This was due to under-prediction of miss-rate for this workload for a small LLC (see Section 4 results). Inaccurate predictions may lead to performance loss, under/over-utilization of the power budget and associated thermal overshoot. This situation is not unique to our system: any non-oracular predictive mechanism would need to detect and deal with occasional mispredictions. Apart from performance counters, we propose using online monitoring schemes similar to Intel's Turbo Boost technology [52] so that the system can continuously monitor and compare predicted power-performance with actual values. In case performance is below expectations, the system will revert back to baseline thereby restoring long-term performance loss to 0% (e.g. `swim`). In case of power/thermal overshoot with strict budgets, throttling mechanisms must be employed to scale down excess voltage and frequency immediately. For soft budgets, corrective action can be taken in the next interval.

Sensitivity: The opportunity for LLC power budgeting exists when total LLC power in relation to on-chip/system power is substantial. For our baseline system, 32MB of LLC consumes between 21.7% to 37.3% of on-chip power (excluding memory power), that is, between 0.7% to 1.2% per MB of LLC. A smaller LLC reduces the transferable portion of the power budget (α in Figure 7). While the maximum gain is 15.9% with a 32MB LLC (`swaptions`), it is 7.9% with a 16MB LLC and 3.5% with an 8MB LLC. A higher baseline frequency reduces possible gains (increases β in Figure 7), but simpler cores increase the opportunity (decreases β). We have obtained similar improvements with simple inorder cores and a 16MB LLC as with more complex cores and a 32MB LLC. A faster rate of temperature rise limits gains. For example, a 6°C rise instead of 3°C rise for every 200MHz reduces scaling from 16% to 15.2% for `swaptions`.

Thermal Headroom: Our DVFS model assumes that enough thermal headroom is available to accommodate the desired scaling. We claim that the maximum permissible scaling is limited by how efficiently heat can be moved away from each individual core to the heat sink, and is not significantly lowered by our scheme. Thermal resistance is directly proportional to thickness/separation and inversely proportional to cross-sectional area [47]. Typical chip thicknesses are < 1 mm and thermal conductivity of copper is higher than silicon [47]. Thus, lateral thermal resistance between cores is expected to be much higher than vertical thermal resistance. As long as on-chip/system TDP is not exceeded, which is guaranteed when predictions are accurate, any system that supports overscaling of voltage/frequency of cores should be able to benefit from LLC power budgeting for performance. Although we assumed a maximum frequency scaling of 25%, the maximum actual scaling was 16.2%. A lower scaling limit would reduce speedups and is similar to having a lower power margin.

Integration with other resource managers: Our proposed predictors can be used in conjunction with other managers targeting optimal system configuration, such as MaxBIPS [25] or machine learning [11]. Power-performance tradeoff information from our predictors can aid global power/resource managers that can then identify the optimal cache configuration directly without needing to perform additional search/ exploration. This would enable faster and more energy-efficient determination of optimal settings for the system. Online monitoring and control-theoretic approaches [30] can be used to further fine-tune predicted voltage and frequency settings. A guard mechanism may be used along with our predictors to make the system energy-secure [12].

Multiprogrammed environments: Our current results are more representative of commercial and scientific server workloads rather than the desktop's more diverse and interactive environment. However, our temporal locality and DVFS models are agnostic of whether one or more applications are executing. For the latter case, locality of the merged address stream and aggregate execution characteristics would be analyzed. Our work targets long-term trends in cache temporal locality and core compute power.

8. RELATED WORK

Isci et al. [25] introduced the concept of maximizing performance for a given power budget, using a global power manager and per-core monitors to set per-core DVFS modes. Sharkey et al. [42] explore front-end microarchitectural alternatives to maximize performance of multi-threaded workloads. Bitirgen et al. [11] used a machine learning approach for resource management with an application to allocating cache ways to provide performance isolation. Section 7 discusses integration possibilities with these approaches.

Meng, et al. [33] also uses way counters and analytic power-performance models for power management. Their work has several important limitations. First, they assume true LRU replacement, which is impractical to implement for highly associative last-level caches. This is critical, because practical implementations such as PLRU do not have the stack property and thus a single tag array cannot both provide replacement decisions and miss-rate predictions for larger sized caches. While Meng et al.'s work could probably be extended to use shadow tags and dynamic set sampling [37,39], the extra area and power would far exceed that of our reuse sampling approach. Second, their study evaluates 600μsecs observation intervals, which are far to short to amortize the reconfiguration overhead of a 32MB LLC. Finally, their approach only handles limited cache reconfigurability (number of ways, not sets), does not consider thermal effects on leakage power, and optimizes for the lowest-power configuration, not the highest-performance configuration.

Dynamic cache reconfiguration is well studied [5, 21]. Our work is largely orthogonal to the reconfiguration mechanism since we expect to amortize the overheads over long execution intervals. Balasubramonian et al. [8] studied memory hierarchy reconfiguration for energy-efficient performance. The optimal cache configuration is selected by exploration—successive cache sizes are chosen till the miss rate is sufficiently small. In our work, analytical models directly predict the globally optimum configuration and no exploration is needed. Dropsho et al. [21] propose using way counters to evaluate configurations with different associativities. Dreslinski et al. [20] discuss caches that can be reconfigured for energy savings, but with a (small) performance hit. Kaxiras et al. [27] study generational behavior of cache line usage to reduce cache leakage with (area-expensive) power-gating control per line and a (small) performance hit.

Previous work has extensively studied cache miss rate prediction using offline estimation of LRU stack/reuse distances [6, 9, 24, 32, 44, 48], but have limited applicability for online use. Tam et al. [50] use hardware mechanisms for address sampling and post-processing software for computing reuse distance distributions.

Way counters [28,39,49] exploit the LRU stack property to predict miss rates for configurations smaller than the current cache. Shadow tags [37] (or Auxiliary Tag Directories [39]) extend way counters to predict configurations with higher associativity than the active cache configuration. Dynamic set-sampling can reduce overheads [38].

Reuse sampling uses replacement-specific cache hit/miss functions to estimate the miss rate given a stack distance/reuse distance distribution [41]. It can estimate miss rates for caches with implementable replacement policies, such as PLRU, RANDOM, NMRU and configurable number of sets as well as ways. This paper examines using both way counters and reuse sampling as prediction techniques.

Intel processors/microarchitectures such as the Core Duo [36], and Ivy Bridge [40] dynamically size caches based on *activity* whereas our predictors are based on *reuse* that enable them to correctly select a smaller cache for highly cache-active but cache-insensitive workloads.

9. CONCLUSIONS

In this work, we quantitatively show that it is possible to increase system performance by decreasing cache size, for a given power budget. When optimizing for system power, 11 of 17 workloads achieve performance improvements of 2-16%. Cache insensitive workloads see significant improvements, while those with less locality and greater cache sensitivity are best left with larger cache configurations due to the effects of memory power. We demonstrated that favorable configurations can be selected using simple analytic models, driven by hardware performance counters to estimate the cache reuse distribution. As technology scales, intelligently budgeting power between system components will become increasingly important to obtaining optimum performance.

10. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation (CNS-0916725, CCF-1017650, CNS-1117280, and CCF-1218323), Sandia/DOE (#MSN123960/DOE890426), and a University of Wisconsin Vilas award. The views expressed herein are not necessarily those of the NSF, Sandia or DOE. Wood has a significant financial interest in AMD.

This work was improved by feedback from Brad Beckmann, Jayneel Gandhi, Dan Gibson, Gagan Gupta, Mark Hill, Lisa Hsu, Marc de Kruijf, Somayeh Sardashti, Nilay Vaish, anonymous reviewers and others.

11. REFERENCES

[1] Apache Lucene. `http://lucene.apache.org/`.

[2] Wikipedia:database download. `http://en.wikipedia.org/wiki/Wikipedia:Database_download`.

[3] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood. Simulating a $2M commercial server on a $2K PC. *IEEE Computer*, 36(2):50–57, Feb. 2003.

[4] A. R. Alameldeen and D. A. Wood. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8–17, July/Aug 2006.

[5] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *Proceedings of the 32nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 32, pages 248–259, Washington, DC, USA, Nov. 1999. IEEE Computer Society.

[6] G. Almási, C. Caşcaval, and D. A. Padua. Calculating stack distances efficiently. In *Proceedings of the 2002 workshop on Memory system performance*, MSP '02, pages 37–43, New York, NY, USA, June 2002. ACM.

[7] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady. SPEComp: A new benchmark suite for measuring parallel computer performance. In *Proceedings of the International Workshop on OpenMP Applications and Tools: OpenMP Shared Memory Parallel Programming*,

WOMPAT '01, pages 1–10, London, UK, UK, July 2001. Springer-Verlag.

[8] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 33, pages 245–257, New York, NY, USA, Dec. 2000. ACM.

[9] K. Beyls and E. D'Hollander. Reuse distance as a metric for cache behavior. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 617–622, Aug. 2001.

[10] C. Bienia. *Benchmarking Modern Multiprocessors.* PhD thesis, Princeton University, Jan. 2011.

[11] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 318–329, Washington, DC, USA, Nov. 2008. IEEE Computer Society.

[12] P. Bose, A. Buyuktosunoglu, J. Darringer, M. Gupta, M. Healy, H. Jacobson, I. Nair, J. Rivers, J. Shin, A. Vega, and A. Weger. Power management of multi-core chips: Challenges and pitfalls. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 977–982, Mar. 2012.

[13] J. A. Butts and G. S. Sohi. A static power model for architects. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 33, New York, NY, USA, Dec. 2000. ACM.

[14] J. Casazza. First the tick, now the tock: Intel microarchitecture (Nehalem). *White Paper, Intel Xeon processor 3500 and 5500 series, Intel Microarchitecture*, 2008.

[15] J. Chang, S.-L. Chen, W. Chen, S. Chiu, R. Faber, R. Ganesan, M. Grgek, V. Lukka, W. W. Mar, J. Vash, S. Rusu, and K. Zhang. A 45nm 24MB on-die L3 cache for the 8-core multi-threaded Xeon processor. In *2009 Symposium on VLSI Circuits*, pages 152 –153, June 2009.

[16] R. Cypher. Apparatus and method for determining stack distance including spatial locality of running software for estimating cache miss rates based upon contents of a hash table. *US7366871*, Apr. 2008.

[17] R. Cypher. Apparatus and method for determining stack distance of running software for estimating cache miss rates based upon contents of a hash table. *US7373480*, May 2008.

[18] A. S. Dhodapkar and J. E. Smith. Saving and restoring implementation contexts with co-designed virtual machines. In *In Workshop on Complexity-Effective Design*, June 2001.

[19] C. Ding and Y. Zhong. Reuse distance analysis. Technical Report UR-CS-TR-741, University of Rochester, Feb. 2001.

[20] R. G. Dreslinski, G. K. Chen, T. Mudge, D. Blaauw, D. Sylvester, and K. Flautner. Reconfigurable energy efficient near threshold cache architectures. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 459–470, Washington, DC, USA, Nov. 2008. IEEE Computer Society.

[21] 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 the 2002 International Conference on Parallel Architectures and Compilation Techniques*, PACT '02, pages 141–152, Washington, DC, USA, Sept. 2002. IEEE Computer Society.

[22] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, pages 148–157, Washington, DC, USA, May 2002. IEEE Computer Society.

[23] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio. Thermal response to DVFS: analysis with an Intel Pentium M. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 219–224, New York, NY, USA, Aug. 2007. ACM.

[24] M. D. Hill and A. J. Smith. Evaluating associativity in CPU caches. *IEEE Transactions on Computers*, 38(12):1612–1630, Dec. 1989.

[25] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, Washington, DC, USA, Dec. 2006. IEEE Computer Society.

[26] S. Jahagirdar, V. George, I. Sodhi, and R. Wells. Power management of the third generation Intel Core micro architecture formerly codenamed Ivy Bridge. In *Hot Chips 24*, Aug. 2012.

[27] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01, pages 240–251, New York, NY, USA, June/July 2001. ACM.

[28] K. Kedzierski, M. Moreto, F. Cazorla, and M. Valero. Adapting cache partitioning algorithms to pseudo-LRU replacement policies. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, Apr. 2010.

[29] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-aware intelligent DRAM refresh. In *Proceedings of the 39th International Symposium on Computer Architecture*, ISCA '12, Washington, DC, USA, June 2012. IEEE Computer Society.

[30] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 449–460, New York, NY, USA, June 2011. ACM.

[31] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R.

Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, pages 92–99, Sept. 2005.

[32] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[33] K. Meng, R. Joseph, R. P. Dick, and L. Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 177–186, New York, NY, USA, Oct. 2008. ACM.

[34] P. Nagpurkar, C. Krintz, M. Hind, P. F. Sweeney, and V. T. Rajan. Online phase detection algorithms. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 111–123, Washington, DC, USA, Mar. 2006. IEEE Computer Society.

[35] S. G. Narendra and A. Chandrakasan. *Leakage in Nanometer CMOS Technologies (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[36] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the Intel Core Duo processor. *Intel Technology Journal*, 10, May 2006.

[37] T. R. Puzak. *Analysis of Cache Replacement Algorithms*. PhD thesis, Dept. of Electrical and Computer Engineering, University of Massachusetts, 1985.

[38] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. A case for MLP-aware cache replacement. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 167–178, Washington, DC, USA, June 2006. IEEE Computer Society.

[39] 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*, MICRO 39, pages 423–432, Washington, DC, USA, Dec. 2006. IEEE Computer Society.

[40] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro*, 32(2):20–27, Mar/Apr 2012.

[41] R. Sen and D. A. Wood. Reuse-based online models for caches. In *Proceedings of the 2013 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, June 2013. ACM.

[42] J. Sharkey, A. Buyuktosunoglu, and P. Bose. Evaluating design tradeoffs in on-chip power management for CMPs. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 44–49, New York, NY, USA, Aug. 2007. ACM.

[43] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, Nov/Dec 2003.

[44] X. Shi, F. Su, J.-K. Peir, and Z. Yang. Modeling and stack simulation of CMP cache capacity and accessibility. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1752–1763, Dec. 2009.

[45] T. Shyamkumar, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical Report HPL-2008-20, Hewlett Packard Labs, 2008.

[46] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner. POWER5 system microarchitecture. *IBM Journal of Research and Development*, 49(4), 2005.

[47] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 2–13, New York, NY, USA, June 2003. ACM.

[48] A. J. Smith. A comparative study of set associative memory mapping algorithms and their use for cache and main memory. *IEEE Transactions on Software Engineering*, SE-4(2):121–130, Mar. 1978.

[49] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 117–128, Washington, DC, USA, Feb. 2002. IEEE Computer Society.

[50] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 121–132, New York, NY, USA, Mar. 2009. ACM.

[51] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM J. Res. Dev.*, 46(1), 2002.

[52] Intel Turbo Boost technology in Intel Core microarchitecture (Nehalem) based processors, Nov. 2008.

[53] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009. `http://www.globule.org/publi/WWADH_comnet2009.html`.

[54] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA, Nov. 2000. ACM.

[55] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ISCA '95, pages 24–37, New York, NY, USA, June 1995. ACM.