Non-Uniform Power Access in Large Caches with Low-Swing Wires

Aniruddha N. Udipi
University of Utah
udipi@cs.utah.edu

Naveen Muralimanohar
HP Labs
naveen.muralimanohar@hp.com

Rajeev Balasubramonian
University of Utah
rajeev@cs.utah.edu

Abstract

Modern processors dedicate more than half their chip area to large L2 and L3 caches and these caches contribute significantly to the total processor power. A large cache is typically split into multiple banks and these banks are either connected through a bus (uniform cache access – UCA) or an on-chip network (non-uniform cache access – NUCA). Irrespective of the cache model (NUCA or UCA), the complex interconnects that must be navigated within large caches are found to be the dominant part of cache access power. While there have been a number of proposals to minimize energy consumption in the inter-bank network, very little attention has been paid to the optimization of intra-bank network power that contributes more than 50% of the total cache dynamic power in large cache banks. In this work we study various mechanisms that introduce low-swing wires inside cache banks as energy saving measures. We propose a novel non-uniform power access design, which when coupled with simple architectural mechanisms, provides the best power-performance tradeoff. The proposed mechanisms reduce cache bank energy by 42% while incurring a minor 1% drop in performance.

1. Introduction

To alleviate the growing gap between processors and main memory, contemporary processors have begun to provide large multi-megabyte last level caches, often occupying upwards of half the total die area. For example, the Montecito processor from Intel has 24 MB of L3 cache [1]. Intel's consumer desktop Nehalem processors have 8 MB of L3 cache [2]. With the memory wall showing no signs of breaking down, these trends are likely to continue, with future caches only growing larger.

Large caches are likely to use a Non Uniform Cache Access (NUCA) architecture, with the cache split into multiple banks connected by an on-chip network [3]. The CACTI 6.0 tool computes the characteristics of large NUCA caches and shows that optimal behavior is exhibited when the cache is partitioned into large banks, where each bank can accommodate a few megabytes of data [4]. About 50% of the NUCA cache's dynamic power is dissipated within these large banks (the rest is mostly within the inter-bank network). Processors that do not employ NUCA will implement monolithic multi-megabyte private or shared L2s or L3s (the Nehalem and Montecito serving as examples).

In addition to consuming silicon area, these caches will also contribute significantly to the energy consumed by the processor. Large last-level shared caches often serve as the cache coherence interface on multi-core processors. Multi-threaded applications will likely make frequent expensive look-ups into the L2 to access shared data. There already appears to be a trend to simplify the design and power efficiency of cores. Intel's shift from the Netburst to Core microarchitecture is a sign of things to come. Sun's Niagara and Rock processors are also designed for low energy-per-instruction among other things. In the meantime, if SRAM cache arrays remain stagnant in design, their contribution to overall chip power will continue to grow. Hence, this paper attempts to provide circuit/architecture innovations to improve energy dissipation within large cache banks.

We show that energy dissipation in a large cache is dominated by the H-tree network within each bank. To address this bottleneck, we propose various designs that leverage low-swing wiring within the cache. Low-swing wires are an attractive choice from the power perspective but are inherently slow. Pipelining them is inefficient and requires additional transceivers at intermediate points along the bus. If employed judiciously, however, their performance penalty can be mitigated while exploiting their low-power characteristics. We discuss these trade-offs in detail for a variety of designs. We finally show that limited use of low-swing wiring provides the best balance between performance and power. This leads us to introduce the notion of *non-uniform power access*, with certain regions of the

This work was supported in parts by NSF grants CCF-0430063, CCF-0811249, CCF-0916436, NSF CAREER award CCF-0545959, Intel, SRC grant 1847.001, and the University of Utah.

cache being accessible at low energy with low-swing wires. Architectural mechanisms are required to exploit the low-power region of a cache bank and we explore novel block placement policies to maximize use of the low-power region. Our results show significant cache energy savings at a very modest performance penalty, the penalty primarily arising from the non-pipelined nature of our low-swing transfers

The paper is organized as follows. Section 2 provides basics on cache bank organization. Section 3 discusses different novel designs that employ low-swing wiring within a cache bank. Architectural innovations that support non-uniform power access are described in Section 4. Section 5 shows results. We discuss related work in Section 6 and draw conclusions in Section 7.

2. Background

Large caches of the future are expected to experience increasing disparity between access delays to different parts of the cache depending on their proximity to the processor core or cache controller. This mandates a Non-Uniform Cache Access (NUCA) architecture [3], with large caches being divided into multiple banks connected by an on-chip network for data and address transfer between banks. A recent study [5] has shown that due to high power and latency overheads associated with on-chip network routers, NUCA caches will implement a few banks, each of non-trivial size. The bank count/size of each bank is determined by the relative contributions of the banks and the network to the total delay and energy consumption of the cache, and associated design constraints. According to CACTI 6.0 [4], a NUCA modeling tool that identifies an optimal trade-off point between bank and network components, a 64 MB NUCA cache will likely be partitioned into large 2 MB or 4 MB banks. Up to 50% of cache dynamic power is dissipated within these large banks. Some processors may also adopt a tiled architecture where every core is associated with a large L2 bank (either a private cache or a slice of a large shared cache) [1]. Thus, regardless of whether future processors adopt private/shared caches, or UCA/NUCA architectures, or tiled/contiguous L2 caches, it is evident that several large cache banks will be found on chip. This work focuses on reducing the significant component of dynamic power within these large cache banks. As per estimates from CACTI 5.3 [6], leakage in a 4 MB cache contributes about 20% to total power consumption. However, there are well studied circuit ([7], [8], [9]) and microarchitectural ([10], [11], [12], [13]) techniques in current literature to tackle leakage in caches. We assume that several of these can continue to be applied to the cache orthogonal to our optimizations, and focus on dynamic power for the rest of the paper.

We now describe briefly the factors that influence the organization, power, and delay of a cache bank. A naive cache takes the form of a single array of memory cells and employs centralized decoder and logic circuitry to store and retrieve data from cells. Such a monolithic model, however, has serious scalability issues. First, wordlines (due to a lack of available silicon area) and bitlines (due to differential signaling) cannot be repeated at regular intervals, causing their delay to increase quadratically with the size of the cache. Second, the bandwidth of the cache is a function of cycle time[1] and a single array cache's bandwidth deteriorates quickly as the array size grows. The performance of the cache is thus limited by long bitlines and wordlines that span the array. To reduce this quadratic delay impact, the cache is divided into multiple segments referred to as subarrays. These subarrays need to be connected through an interconnect fabric to transfer addresses and data within the cache. In order to reduce design complexity, an interconnect with easily predictable timing characteristics is essential. A balanced H-tree network (Figure 1) provides uniform pipelined access without complex switching circuits and proves to be an ideal choice. The number of subarrays that the bank is split into, the height/width of the grid of subarrays (the aspect ratio of the bank) and the aspect ratio of the subarrays themselves are defined by three fundamental parameters of cache bank design:

- NDWL - Number of vertical partitions in the array i.e., the number of segments that a single wordline is partitioned into. This determines the number of columns of subarrays.
- NDBL - Number of horizontal partitions in the array i.e., the number of segments that a single bitline is partitioned into. This determines the number of rows of subarrays.
- NSPD - Number of sets stored in each row of a subarray. For given Ndwl and Ndbl values, this decides the aspect ratio of the subarray.

An example organization is illustrated in Figure 1. The design space of bank design is defined by variations of these parameters and the resultant complex interaction of various internal power and delay components.

For example, a small subarray count would enable tighter packing of cells leading to increased area efficiency (cell area/total area), but would result in increased delay due to longer wordlines and bitlines. A large subarray count would give better delay characteristics but result in increased silicon area consumption.

1. The cycle time of a cache is the sum of the wordline, bitline, and senseamp delays.

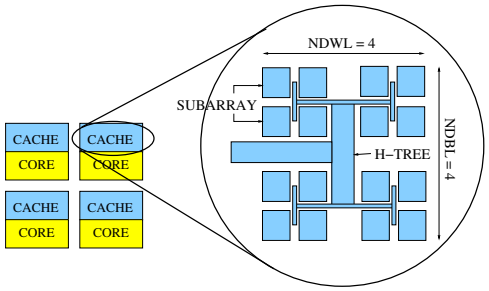

Figure 1. **An example cache bank organization**

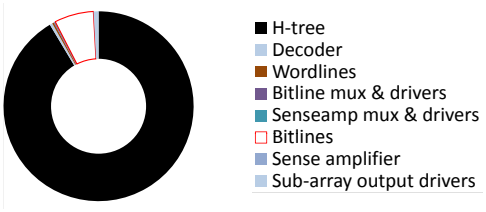

Figure 2. **Contributors to total cache access energy**

The effect of this design space exploration on the energy consumption of the cache is described in more detail in the upcoming subsection.

A typical cache modeling tool like CACTI iterates over a large range of NDBL, NDWL, NSPD etc., and determines the optimal configuration of the cache for a given set of design constraints, specified by weightage given to area/power/delay and the maximum permissible deviation of these values from optimum.

Understanding Bank Energy Consumption: The total energy spent on a cache access is the sum of energies consumed by the decoder circuitry, bitlines, sense-amps, multiplexers, and the H-tree network. For various large bank CACTI layouts that optimize delay, power, area, and their combinations, we consistently observe a large dynamic power contribution from the H-tree network. Figure 2 shows a representative dynamic energy breakdown across various cache components for banks designed for low latencies, resulting in a relatively larger number of small subarrays. The work in this paper targets the dominant contributor in this breakdown, the H-tree network.

3. Low-swing Wires in Cache Banks

The array of SRAM cells constituting a bank is typically not one monolithic structure but is split into multiple smaller units called subarrays. This helps keep capacitances within the bank low, reducing delay and allowing faster access. The subarrays in a bank are typically connected using a balanced H-tree structure that serves to provide uniform time access to every subarray in the bank and keep complexity low. A key insight of our proposal is that in the process, all accesses also become uniform power, which is an unnecessary constraint placed on the access. Combined with the fact that the H-tree is a major contributor to the total cache energy, this is clearly an area to target for significant reductions in energy.

3.1. Low-Swing Signalling

One of the primary reasons for the high power dissipation of global wires is the full swing requirement imposed by repeaters. While this can be somewhat mitigated by reducing repeater size and increasing repeater spacing, the overhead is still relatively high. Low voltage swing alternatives represent another mechanism to vary the wire power/delay/area trade-off. There is little silicon overhead to using low-swing wires since there are no repeaters and the wires themselves occupy zero silicon area. A low-swing pair does require special transmitter and receiver circuits for signal generation and amplification but these are easily amortized over moderately long wires. Reducing the voltage swing on global wires can result in a linear reduction in power (we assume a voltage swing of 100 mV [14], [4]). In addition, assuming a separate voltage source for low-swing drivers will result in quadratic savings in power (we assume a supply voltage of 0.4 V [14], [4]). However, these power savings are accompanied by some big caveats. Low swing wires suffer significantly greater delays than repeated full-swing wires and thus cannot be used over very long distances. There is also a non-trivial cost associated with pipelining low-swing wires. In spite of these problems, low-swing wiring is appropriate in some settings and a few studies ([15], [16], [17]) have considered them when connecting cache banks to CPUs. This paper is the first to consider the use of low-swing signalling within a cache bank to address the power bottleneck posed by intra-bank wires in large caches. We study several ways of utilizing low-swing wires for this purpose (discussed next), and consider their architectural ramifications in Sections 4 and 5.

3.2. Single Low-Swing Bus

A simple way to exploit differential low-swing signaling would be to build the H-tree entirely with a single low-swing bus covering the bank. This cache model provides excellent energy savings, coming at the cost of very significant drops in performance. Such a cache is first of all slow due to the higher latency of the long low-swing bus. Because the wire is not pipelined, accesses are essentially serial in nature, with the cache cycle time becoming equal to the access time, leading to drastically increased contention to access the bank. Such a scheme is not worth considering except in

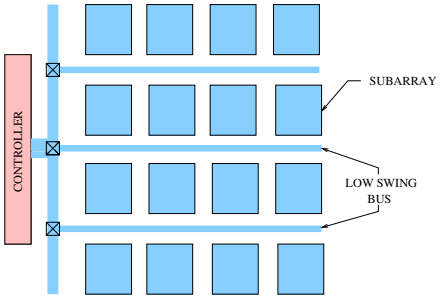

Figure 3. **Multi low-swing interconnect structure**

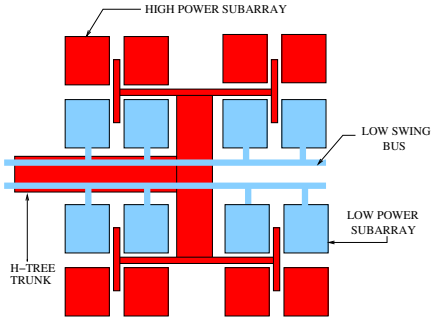

Figure 4. **Low-swing H-tree trunk**

niche cases where power is a far more important consideration than performance.

3.3. Multiple Low-Swing Buses

To address the contention in non-pipelined low-swing buses, we consider an alternate scheme with multiple low-swing pairs in the bank, as shown in Figure 3. There is one low-swing bus per row of subarrays, connected to the controller by a vertical bus. This seeks to essentially spread the contention around, by throwing more resources at the problem. Despite this increased cost and complexity, however, we find that there is still a non-trivial performance hit. Aggressively throwing even more resources at the problem would simply lead to tremendous design complexity and is not considered a feasible option.

3.4. Fully Pipelined Low-Swing H-Tree

Pipelining low-swing wires requires usage of differential transmitter and receiver circuitry at every pipeline stage. These circuits consume non-trivial amounts of energy; amortized over 1 mm of low-swing wire, a single transmitter-receiver pair causes a 58% energy overhead, calculated from values in CACTI 6.0. Considering that the bus is likely to be at least 128 bits wide, there is a significant energy overhead to pipelining low-swing wires. Despite this, we still see good energy savings due to the large energy gap between full-swing and low-swing wires. The bigger problem, however, is that the low-swing bus is inherently slower than regular wires, and still causes a significant IPC drop, especially in applications sensitive to L2 latency.

3.5. Low-Swing H-Tree Trunk

As a novel alternative to the various schemes discussed so far, we propose a *non-uniform power access* structure as shown in Figure 4. We overlay a low-swing bus over the central trunk of the H-tree. The rows of subarrays adjacent to this central low-swing bus are connected to the low-swing bus and not the H-tree. They can therefore be accessed with greatly reduced energy consumption. The remaining rows continue to use the balanced H-tree network. The introduction of this low-swing interconnect does not directly affect the basic H-tree design in any significant way. A simple switch connects either the low-swing or regular interconnect to the cache controller depending on the subarray being accessed. Limiting the low-swing pair to just the central subarrays ensures that *the access delay of the low-power region is the same as that for the high-power regions connected by the default H-tree.* For example, in a 4 MB bank, the H-tree delay as computed by CACTI is approximately 0.32 ns. The width of the bank (and thus the length of the low-swing bus along the center) is 1.53 mm which corresponds to a low-swing wire delay of approximately 0.26 ns. We can thus maintain uniform time access and the scheme can be kept transparent to the cache controller and the processor. Our experiments show that delay and contention are low enough at this point on the design spectrum to see little drop in overall IPC. The energy savings obtained are only proportional to the percentage of rows that are accessible by this low-swing bus, which is typically quite small (1/16th in our case). We believe, however, that the performance advantages of this model make it worth considering very seriously. We next propose simple architectural mechanisms to increase the percentage of accesses hitting the low-power region through dynamically reconfigurable data placement in the cache.

4. Exploiting Non-Uniform Power Access

4.1. Smart Data Placement

Energy estimates from CACTI 6.0 show that an access over the low-swing interconnect could be as much as five to seven times cheaper (depending on cache size and H-tree wire type) than an access over the default H-tree. Despite this, however, since only a small fraction of the total cache can be accessed via the low-power wires, the total energy savings obtained are marginal. It is clear that the default data placement will not effectively exploit the low-power resources at

our disposal. We will need a smarter mechanism to maximize the number of accesses occurring over the low-swing pair.

The bank is split into 'NDBL' rows of subarrays, only two central ones of which are accessible via the low-power interconnect structure, thus typically representing only a small fraction of the total cache capacity. We propose assigning a fraction of the ways of the cache to the low-power region and the remaining ways form the high power region. Since this is the last level cache, we assume a sequential tag and data lookup scheme, as is the norm. It must be stressed that in the interest of maintaining low-complexity, we do not alter the design or access mechanism of the tag array. On every access, the tags in all ways are looked up in parallel to determine the location of the required block. This would not affect energy consumption significantly since the tag arrays typically contribute less than a tenth of the total access energy in a cache bank. The real savings are to be had by optimizing data array accesses. A hit in a low-power way is a "low-power access" with immediate energy savings. A miss in the low-power way results in the block being brought into the low-power region from the high-power region or main memory. Assuming good temporal reuse of data, the next time that block is requested, it would hit in the low-power region, saving energy. Having set up the above basics for low-power cache access, we now describe two policies for managing blocks in low and high power regions.

In the first policy, *Swap*, blocks are brought into the low-power region on touch (either from the high-power region or main memory), and the blocks they replace are swapped out into the high power region, evicting the LRU way there. The low-power ways thus always represent the MRU ways for that set. On a high-power hit, the block is moved into the low-power region, and the block in the low-power region is moved into the high-power region, thus earning the *Swap* moniker. The most recently used ways (say, W) of every set in the cache are in the low-power region.In our experiments, W is 1, out of 16 ways. As already stated, there is no change to the UCA latency, so any change in performance is because of greater contention for the single unpipelined low-swing bus. Cache miss rates should also be the same as the baseline because replacement continues to be governed by LRU.

The more a block is touched (re-used) per fetch into the low-power region, the greater the energy savings. To estimate the level of re-use to make this fetch worthwhile, consider the following analytical estimates. For now, we will assume that the energy of a high-power access H is 7 times the energy of a low-power access L (actual numbers are presented in Table 1). If a block is touched N times before slipping out of the MRU position, the conventional cache would have incurred the energy cost of N high-powered accesses. With the proposed scheme, there would have been $N-1$ hits in the low-power way and one swap at the start, resulting in $N+1$ low-power hits and 2 high-power hits. For the proposed model to consume less energy than the baseline,

$$N \times H > 2 \times H + (N+1) \times L$$
$$N > 2.5$$

While this policy is effective, it is expensive in that every low-power miss incurs a swap that requires two low-power and two high-power accesses.

Now consider an alternative policy, referred to as *Duplicate*. On an L2 miss, the block is fetched from memory and placed in the low- and high-power region (thus allowing duplicate copies of a block in L2). When a block in the low-power region is evicted, it writes into its copy in the high-power region if the block is dirty. If the block is clean, it is simply dropped. On a high-power hit, the block is copied into the low-power region and the previously resident block is evicted following the rules above. Thus, if the block that is brought into the low-power region is written to, its eviction results in a swap and therefore incurs the cost of two high-power accesses and two low-power accesses (just as in the *Swap* policy). On an L2 miss, the block also incurs one additional high-power access than the *Swap* policy. However, if the block fetched into the low-power cache is typically only read, on its eviction, it incurs one less high-power and one less low-power access as the block is quietly discarded. Even though this policy seems initially wasteful because it maintains a duplicate copy of the block, it ends up being more efficient than the *Swap* policy because blocks are frequently read-only and having the duplicate copy means that a new block is simply fetched instead of a swap being required. Our results show that the *Duplicate* policy consumes less power than the *Swap* policy.

Forming equations similar to those developed for *Swap*, the first fetch that brings the block into the low-power way consumes H (reading from the high-power region) plus L (writing into the low-power region). The subsequent low-power hits consume L. A copy back into the high-power region again costs one L (reading from the low-power region) plus one H (writing into the high-power region).

If the block is evicted clean at the end of the reuse run,

$$N_{clean} \times H > H + L + (N_{clean} - 1) \times L$$

If the block is dirty and has to be written back to the high-power region on eviction,

$$N_{dirty} \times H = H + L + (N_{dirty} - 1) \times L + H + L$$

$$N_{clean} > 1.16; N_{dirty} > 2.6$$

If writes are not very frequent, *Duplicate* is clearly better than *Swap* even though it initially appears space-inefficient.

4.2. Dynamic Reconfiguration

The block placement scheme described in the previous subsection gives excellent energy savings, provided a modestly high number of accesses can be satisfied by the low power way. Below a certain threshold hit rate, we begin to see negative effects by bringing in blocks to the low-power way on every touch. The extra energy required to move blocks to the low-power region starts to overshadow the energy savings obtained through the low-swing pair when there is insufficient reuse before eviction. It may be the case that certain phases of the application show very poor data reuse in the low-power region, leading to negative energy savings. To handle application phases with low resue, we propose a dynamic reconfiguration scheme where the cache, if necessary, is able to switch off the placement scheme described above. In this mode, the L2 cache simply behaves like a conventional cache, and blocks are not brought in to the low-power way(s) on access. To facilitate such a mechanism, we would need to accurately characterize the local behavior of the application at any point in time. We simply maintain a saturating counter that increments every time an access hits in the low-power way and decrements on a miss. When the counter value falls below a certain threshold, the L2 starts to operate in the conventional manner. While in this mode, the counter increments on a hit in the most recently used (MRU) way and decrements on a miss. When the counter value goes above a threshold, the cache moves back into smart placement mode. We empirically found that a skewed 5 bit counter going between -15 and +15, with increments in steps of 2 and decrements in steps of 1, with the threshold value being 0, effectively captured various reuse scenarios. Note that there is a *single* global counter, not one per cache line.

4.3. Discussion

We now discuss some of the finer points of our proposals and a few associated overheads. Since our *Duplicate* scheme deviates slightly from the LRU replacement policy, a minor fluctuation in miss rates is observed. The duplicate entries also mean that capacity is slightly lowered. In most cases, the cache behaves like a 15-way cache (for our 16-way cache with 1 way in the low-power region). This is, however, not always true because the LRU policy may evict a high-power cache line while its copy is still resident in the low-power cache. This does not violate correctness as a duplicate copy is not a requirement.

The look-up for this scheme is also complicated slightly compared to baseline. If we are looking for block A, the tag search may reveal that block A is in way-6. This block must now be brought into the MRU way-1. This means that block B which is currently resident in way-1 and dirty has to be written back into its high-power copy. The tags must again be searched to locate the way that houses B's duplicate copy. Additional look-ups of the tags are not expensive, considering that high-power accesses are the bottleneck and this policy minimizes high-power accesses. The presence of duplicate copies also does not lead to coherence issues. If there are duplicate copies, there will be a hit in way-1 and this automatically causes the copy in the high-power way to be dis-regarded. When the low-power block is evicted, it will over-write the copy in the high-power way.

Also, under the smart line placement scheme, every access to a non-MRU way forces a swap in L2. This swap requires an access to a low and high power way, unlike the baseline that would have simply required one high power way access. However, we show with our analytical models that the energy savings obtained on every cache hit in the low-power way easily amortize this swap cost over as few as three accesses.

Our data placement and mapping schemes bear resemblance to an L2/L3 hierarchy or a filter cache [18] based hierarchy. However, we believe our approach is orthogonal to the actual hierarchy and can continue to be used for the largest last level cache structure. Further, we eliminate the need for interconnects between multiple physical cache structures. Our experiments show that our non-uniform scheme provides on average 25% more energy savings than a filter cache model with similar capacities, i.e., it is more efficient to access a portion of a large cache with low-swing wires than it is to access a small cache with conventional full-swing wiring.

The dynamic reconfiguration scheme is simply a decision between bringing the most recently touched block into the low-power way or not. It suffers practically no additional performance or energy overheads over the smart placement scheme. There is also little hardware overhead since we only use a single five bit saturating counter for the entire cache.

Model	Latency (cycles)	Access Energy (nJ)	IPC
Baseline	5	0.185	1.456
Single Low-swing	12	0.016	1.181
Pipelined Low-swing	12	0.040	1.376
Multi Low-swing	8	0.015	1.337
Non-uniform model	5	0.014(LP)	1.430

Table 1. **Access characteristics for a 4MB bank**

5. Results

5.1. Methodology

All of our architectural analysis was carried out using the SimpleScalar-3.0 [19] out-of-order simulator for an 8-issue Alpha AXP with an ROB size of 80 and an LSQ size of 40. The baseline processor is assumed to have separate 2-way, 1-cycle 32 KB I- and D- L1 caches with a unified 16-way 4 MB L2 cache. The L1 and L2 have cache line sizes of 32 and 64 bytes respectively. Main memory latency is assumed to be 300 cycles. The L2 cache is organized as a 32x32 grid of equally sized subarrays, based on Ndwl and Ndbl values obtained from CACTI 6.0. The two central rows are accessible via the low-swing pair, allowing 64 of the 1024 subarrays to be low-power. The low-power region is therefore one-sixteenth of the cache, *i.e.*, one way. As a workload, we employ the SPEC2k programs executed for a 100 million instruction window identified by the Simpoint toolkit [20]. The caches are warmed up for 25 million cycles before making measurements.

All delay and power calculations are for a 32nm process technology and a clock frequency of 5 GHz, as computed by CACTI 6.0. The baseline wiring is assumed to be 10% delay penalty[2] repeated full-swing wires. We also show a sensitivity analysis for other baseline wiring assumptions. We measure in detail the cache access statistics in the low-power and high-power ways, including hit rates and writebacks. Bank access characteristics and IPC values for the various models are shown in Table 1. Every high-power access consumes the energy of one default H-tree access plus the remaining components (bitlines, wordlines, senseamps etc.). Every low-power access consumes a reduced H-tree access energy with all other components assumed to be identical to the default case.

A hit in the low-power way simply consumes one low-power access energy. A hit in the high-power way requires a swap, thus incurring the energy for two low-power accesses and two high-power accesses. A cache miss requires the block to be placed in the low-power way; the current resident in the low-power way is copied to a high-power way; the LRU block is written back to memory if dirty. Thus, up to two memory accesses, two low-power accesses, and up to two high-power accesses are performed.

2. There are inherent tradeoffs between delay and power in the design of repeated full-swing wires. Smaller and fewer repeaters result in slow wires but decreased power consumption. More heavily repeated wires are fast but burn more power. A "10% wire" would incur a 10% delay penalty from optimal for reduced power consumption [21].

5.2. Analysis of Low-Swing Design Points

Figures 5 and 6 show the energy savings obtainable and performance degradation suffered by the various models that introduce low-swing wiring inside cache banks. Building the H-tree entirely out of low-swing wires provides more than 90% savings in energy compared to the baseline full-swing bus case. However, we see that this is accompanied by a 17% drop in IPC due to the increased delay and contention of the low-swing bus. By pipelining the low-swing bus with additional transmitters and receivers, an energy penalty of 12% is incurred, but IPC degradation is greatly reduced. Though the average IPC drop relative to the baseline in this case is just over 5%, there is a subset of benchmarks (not shown separately due to space constraints) that are sensitive to L2 latency and suffer as much as 17% decrease in IPC due to increased delay of the low-swing wires. The multiple low-swing model gives mediocre energy savings with moderate IPC degradation and does not represent an appealing design point. The non-uniform power access model displays IPC drops that are within error margins (just over 1% on average, with 3% in the worst case) and is the best from the performance view point. We note that not all SPEC2k benchmarks have large enough working set sizes to be significantly impacted by the capacity reduction of roughly 1/16 in our scheme. However, since even the largest program in SPEC2k is impacted by just 3%, we expect performance trends to be similar even for other larger benchmark suites. The energy savings of our scheme by itself are very marginal, typically less than 5%. When supported by simple architectural schemes, however, we see a considerable 42% energy saving, proving to be an attractive choice if both energy and performance are considered important. When we consider the overall processor ED^2 metric, the non-uniform access model provides a 5% improvement on average over the baseline, with a best case improvement of up to 25% (this assumes that the L2 cache contributes 20% of total chip power). The pipelined low-swing model has the next best ED^2, yielding an average 3% improvement over the baseline. Clearly, these two models represent the most compelling design points, with the proposed non-uniform power model having the best performance and ED^2 while incurring the cost of block movements.

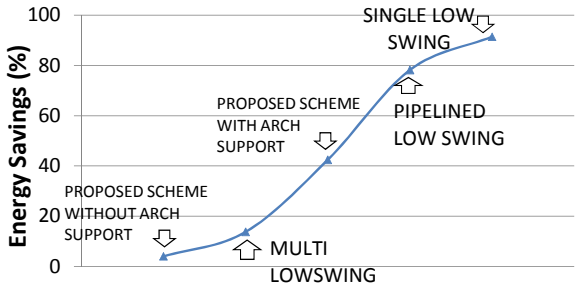

Figure 5. **Average Energy savings for different design points**

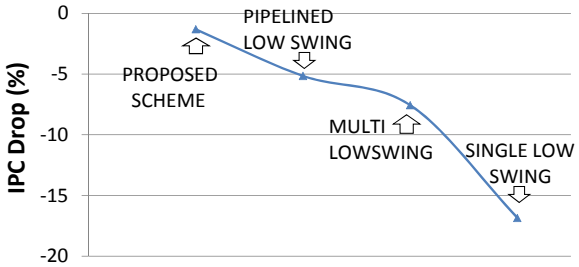

Figure 6. **Average IPC degradation for different design points**

Details of the block swap/duplication mechanisms are discussed next.

5.3. Architectural Mechanisms

Copying the most recently used block of every set into the low-power way is an effective way of maximizing the number of accesses hitting the low-power region. As shown earlier, the *Duplicate* scheme is more energy efficient and we pick this for all of our analysis. We also see only marginal fluctuations in miss rates and IPC when moving between *Duplicate* and *Swap*. Figure 7 shows the energy savings obtained by doing this kind of smart block placement. There is a 42% saving on average, with the best benchmark showing an 88% energy advantage. Table 2 details the access statistics for all applications in the L2 cache. We see that even with the moderate hit rates seen in the low power way, energy savings are substantial. This is explained simply by the huge energy difference

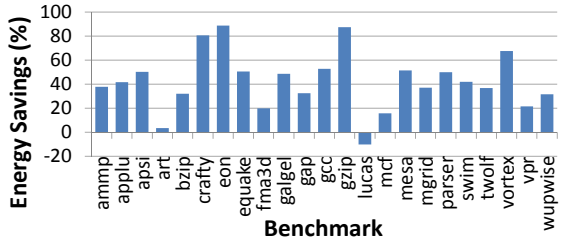

Figure 7. **Energy savings with smart data placement but no dynamic reconfiguration**

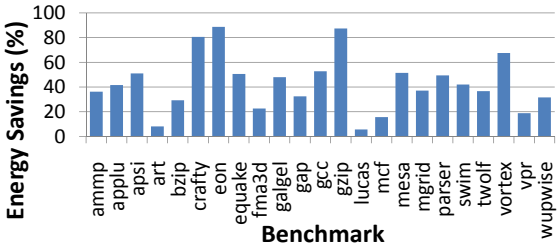

Figure 8. **Energy savings with dynamic reconfiguration**

between low-power and high-power accesses (7x-10x). The energy savings achieved are also proportional to the reuse count per fetch N, since the copy cost is amortized over multiple accesses and we begin to see additional savings. For example, eon and gcc have reuse counts greater than 35, and show energy savings of over 80%.

There are a couple of applications that show an increase in energy consumed when data is copied on first-touch. These are programs that have poor hit rates in the low-power way of the cache, typically under 40%, due to low reuse counts (lucas and art). Data is repeatedly copied into the low-power way, only to be thrown out without being reused sufficiently to amortize the cost of the copy. For example, lucas and art have reuse counts less than 0.5, which means that more often than not, blocks are brought into the low-power way and evicted without reuse. Such blocks are best left in the high-power way and accessed directly from there.

Our dynamic reconfiguration scheme keeps track of the application's access patterns at run-time and turns off block copying when required. Figure 8 shows energy savings obtained in this case. We see that the energy savings of applications that were already benefiting from the smart placement scheme remain practically untouched, while the previously negative numbers are now positive, though relatively small. Our experiments show that applications in the former class remain in "copy-on-touch" mode upwards of 95% of the time, whereas the latter are in this mode less than 30% of the time. This indicates that our simple counter is effectively capturing the behavior of the application in a small time window.

The energy savings obtained are clearly a function of the capacity of the cache, since this determines the baseline access energy and the percentage of accesses that can be satisfied by the low-power way for a given workload. As shown in Figure 9, the larger the cache, the better the savings are, making this scheme even more attractive as cache sizes increase. We see that beyond a point, simply employing smart data placement actually increases energy consumption due to insufficient reuse. The dynamic reconfiguration

Benchmark	L2 Accesses*	LP Hits*	LP Misses*	Writebacks (LP to HP)*	LP Hit Rate (%)	% of Dirty Misses	Avg Reuse Count
ammp	33.2	19.3	13.9	3.6	58.1	26.1	1.4
applu	24.0	12.8	11.2	0.5	53.4	4.9	1.2
apsi	18.7	13.4	5.3	2.1	71.8	40.6	2.5
art	172.2	54.7	117.5	31.2	31.7	26.6	0.5
bzip	23.3	12.0	11.2	2.3	51.7	20.8	1.1
crafty	13.4	12.2	1.3	0.1	90.7	10.5	9.7
eon	5.9	5.7	0.1	0.0	97.5	14.0	39.7
equake	83.8	55.1	28.7	3.8	65.7	13.2	1.9
fma3d	34.1	17.9	16.2	6.2	52.5	38.4	1.1
galgel	39.6	27.0	12.6	4.0	68.2	31.6	2.2
gap	5.7	4.1	1.6	1.4	71.7	84.9	2.5
gcc	57.7	46.4	11.3	9.0	80.5	80.1	4.1
gzip	31.2	30.4	0.8	0.4	97.4	49.4	36.7
lucas	70.8	19.5	51.4	20.1	27.5	39.1	0.4
mcf	212.6	86.5	126.1	28.5	40.7	22.6	0.7
mesa	4.1	3.2	0.9	0.7	78.8	81.9	3.7
mgrid	28.4	16.8	11.7	2.7	59.0	22.9	1.4
parser	22.1	15.6	6.5	2.3	70.6	35.1	2.4
swim	74.4	51.5	22.8	9.6	69.3	42.1	2.3
twolf	40.8	24.8	16.0	5.7	60.7	35.6	1.6
vortex	11.6	9.2	2.4	0.3	79.3	11.1	3.8
vpr	36.3	18.2	18.2	6.5	50.0	35.7	1.0
wupwise	7.5	3.8	3.8	0.6	49.9	15.6	1.0

Table 2. **Cache access statistics (LP - Low Power Way, HP - High Power Way, *Access Counts x100,000)**

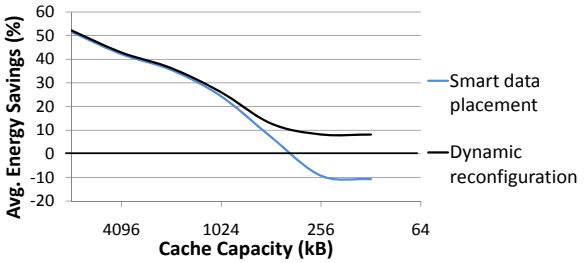

Figure 9. **Sensitivity of energy savings to bank capacity**

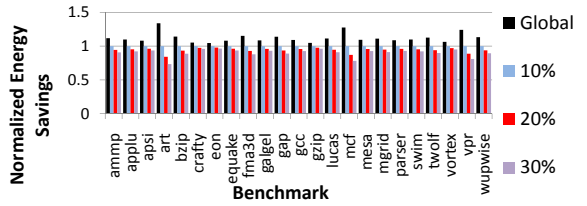

Figure 10. **Sensitivity of energy savings to baseline wire type**

scheme detects such situations and reverts to default behavior in program phases with poor locality.

The savings are also a function of the kind of wiring assumed for the default H-tree network. As the wires pay more delay penalty, they are themselves cheaper to operate in terms of energy, and the room for energy improvement reduces. Figure 10 shows the results of this sensitivity analysis, normalized to the 10% case.

6. Related Work

Low-swing wires are gaining popularity as energy consumption becomes a major issue in electronics design. The *Smart Memories* project ([15], [16], [17]) explored using a low-swing crossbar to connect processing elements and cache subarrays in a reconfigurable environment. CACTI 6.0 [4], the latest version of the popular cache modeling tool now incorporates support for low-swing wiring inside the cache. Ho et al. recently proposed a capacitively driven low-swing wire design for high-speed and low-energy [22]. There have been numerous other proposals to reduce energy consumption in caches. Flautner et al. [11] proposed putting a subset of cache lines into a state-preserving low-power drowsy mode based on cache activity. Agarwal et al. [23] utilize the concept of Gated-Ground (NMOS transistor inserted between Ground line and SRAM cell) to achieve reduction in leakage energy. Yang et al. [24] propose an L0 instruction cache before the L1 to achieve lower capacitance and thus lower energy without affecting performance. Ishihara et al. [25] advocate using non-uniformity in the number of ways per set to achieve energy advantages. All of the above schemes do little to reduce energy in the H-tree, a major contributor to cache energy. Our H-tree optimizations can work in tandem with all of these schemes to reduce the power consumed in the wiring connecting the cells.

7. Conclusions

Future processors will accommodate large multi-megabyte caches. The energy consumed by large cache banks will continue to be a growing problem, especially as CPU cores become more power-efficient. This paper isolates the H-tree within a bank as the major energy bottleneck within a large cache. We study various ways of introducing low-swing wires within the cache to address this bottleneck. Using a pipelined

low-swing bus for the entire H-tree provides the best energy savings but can lead to as much as 17% drops in IPC. We show that the use of a single low-swing bus, providing low-power access to a small part of the cache, gives the best energy-performance tradeoff. We thus introduce the notion of non-uniform power access within a cache bank. Since the low-power region is a very small fraction of the total cache, architectural mechanisms are required to boost its access frequency. We advocate a policy that is based on MRU access and block duplication within the L2. We see overall cache energy reductions of 42% on average with just over 1% drop in IPC.

We believe that non-uniform power access (NUPA) has much potential, just as non-uniform cache access (NUCA) has opened up several opportunities in recent years.

References

[1] C. McNairy and R. Bhatia, "Montecito: A Dual-Core, Dual-Thread Itanium Processor," *IEEE Micro*, vol. 25(2), March/April 2005.

[2] "First the Tick, Now the Tock: Next Generation Intel Microarchitecture (Nehalem)," Intel Whitepaper, Tech. Rep., 2008.

[3] C. Kim, D. Burger, and S. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches," in *Proceedings of ASPLOS*, 2002.

[4] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of MICRO*, 2007.

[5] N. Muralimanohar and R. Balasubramonian, "Interconnect Design Considerations for Large NUCA Caches," in *Proceedings of ISCA*, 2007.

[6] "CACTI: An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model," http://www.hpl.hp.com/research/cacti/.

[7] "High-k and Metal Gate Research," http://www.intel.com/technology/silicon/high-k.htm.

[8] S. Rusu, S. Tam, H. Muljono, D. Ayers, J. Chang, B. Cherkauer, J. Stinson, J. Benoit, R. Varada, J. Leung, R. Lim, and S. Vora, "A 65-nm Dual-Core Multithreaded Xeon Processor With 16-MB L3 Cache," *IEEE Journal of Solid State Circuits*, vol. 42, no. 1, pp. 17–25, January 2007.

[9] K. Zhang, U. Bhattacharya, Z. Chen, D. Murray, N. Vallepalli, Y. Wang, B. Zheng, and M. Bohr, "A SRAM Design on 65nm CMOS Technology with Integrated Leakage Reduction Scheme," in *Proceedings of VLSI*, 2004.

[10] S. Heo, K. Barr, M. Hampton, and K. Asanovic, "Dynamic Fine-Grain Leakage Reduction Using Leakage-Biased Bitlines," in *Proceedings of ISCA*, 2002.

[11] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *Proceedings of ISCA*, 2002.

[12] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *Proceedings of ISCA*, 2001.

[13] S. Yang, M. Powell, B. Falsafi, K. Roy, and T. Vijaykumar, "An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep Submicron High-Performance I-Caches," in *Proceedings of HPCA*, 2001.

[14] R. Ho, "On-Chip Wires: Scaling and Efficiency," Ph.D. dissertation, Stanford University, August 2003.

[15] R. Ho, K. Mai, and M. Horowitz, "Efficient On-Chip Global Interconnects," in *Proceedings of VLSI*, 2003.

[16] K. Mai, R. Ho, E. Alon, D. Liu, Y. Kim, D. Patil, and M. Horowitz, "Architecture and Circuit Techniques for a Reconfigurable Memory Block," in *Proceedings of ISSCC*, 2004.

[17] K. Mai, T. Paaske, N. Jaysena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," in *Proceedings of ISCA*, 2000.

[18] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," in *Proceedings of MICRO*, 1997.

[19] D. Burger and T. Austin, "The Simplescalar Toolset, Version 2.0," University of Wisconsin-Madison, Tech. Rep. TR-97-1342, June 1997.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *Proceedings of ASPLOS*, 2002.

[21] K. Banerjee and A. Mehrotra, "A Power-optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs," *IEEE Transactions on Electron Devices*, vol. 49, no. 11, pp. 2001–2007, November 2002.

[22] R. Ho, T. Ono, F. Liu, R. Hopkins, A. Chow, J. Schauer, and R. Drost, "High-speed and low-energy capacitively-driven on-chip wires," in *Proceedings of ISSCC*, 2007.

[23] A. Agarwal, H. Li, and K.Roy, "DRG-Cache: A Data Retention Gated-Ground Cache for Low Power," in *Proceedings of the 39th Conference on Design Automation*, June 2002.

[24] C.-L. Yang and C.-H. Lee, "HotSpot Cache: Joint Temporal and Spatial Location Exploitation for I-cache Energy Reduction," in *Proceedings of ISLPED*, 2004.

[25] T. Ishihara and F. Fallah, "A Non-Uniform Cache Architecture for Low Power System Design," in *Proceedings of ISLPED*, 2005.