Characterizing the Opportunity and Feasibility of Reconfigurable Memory Hierarchies for Improved Energy Efficiency

Pietro Cicotti
San Diego Supercomputer Center

Laura Carrington
San Diego Supercomputer Center

Andrew Chien
University of Chicago

Abstract

The end of Dennard scaling has made energy-efficiency a critical challenge in the continued increase of computing performance. An important approach to increasing energy-efficiency is hardware customization to specific application needs. In this study we explore the opportunity for memory hierarchy customization for energy-efficiency, exploring reconfigurable memory hierarchies. Using a workload of 37 diverse benchmarks, we address three key questions: 1) how much benefit is possible?, 2) how much reconfiguration is required?, and 3) can we automatically select a good memory hierarchy configuration? Our results show that the potential benefit is large – average reductions of 70% in memory hierarchy energy with no performance loss. Further, our results show that number of configurations need not be large; ten carefully chosen configurations can deliver 90% of this benefit (63% energy reduction) suggesting that configurable hierarchies may be practically realizable. Finally, we explore reuse distance as a guide to select the best memory hierarchy configuration as a first step towards automatic configuration, and show that it can effectively predict which memory hierarchies will both maintain performance and deliver energy efficiency.

1 Introduction

A major factor driving research in computer architecture is the increasing pressure for energy efficiency due to the end of Dennard scaling [1] combined with practical limits to the power a processor can consume [2, 3] in smartphone, tablet, laptop, and server environments. As Moore's law continues to deliver increases in transistors density [4], there are growing opportunities for CPU customization ranging from core extensions [5-7] to accelerators on systems-on-chip [8, 9]. These systems all exploit customization in the datapath – matching the transistor structure of the processor to the needs of the application - to improve both performance and energy efficiency.

While the aforementioned efforts focus on the micro-architecture, our focus here is complementary, exploring memory hierarchy customization for energy efficiency. Memory hierarchies have long been critical elements of system performance. As processors have continued to improve at a faster rate than memory, commonly known as the memory wall [10], multi-level caches have been a key element of computer architecture for decades with research studies spanning organization [11], write and prefetch techniques [12-14], and many other aspects. In short, all modern microprocessors depend on highly optimized memory hierarchies to achieve good performance.

Specifically, we explore the potential of a configurable memory hierarchy to increase energy efficiency by matching its structure to application program needs. And, a model where the memory hierarchy is configured for each application program. This approach posits CPU's with memory hierarchies with reconfigurable cache sizes at multiple levels, and with the growing presumption of dark silicon, [2, 3] we discard normalized comparison based on equal area or capacity. In many cases, using less produces higher energy efficiency. This holistic view builds on a wide variety of work to optimize memory hierarchy energy-efficiency, including reconfiguration for dynamic energy (e.g. cache and TLB access mechanism and organization [15, 16]), circuit level efforts to reduce leakage (e.g. drowsy caches [17]), even architecture level techniques to reduce leakage (e.g. cache decay [18]). However, by opening the design space by presuming the availability of dark silicon and doing macroscopic cache array configuration (in/out), we can address all aspects of dynamic and static energy, including even the perhaps 20% remaining leakage, which at deep submicron technologies and 16MB+ cache sizes is quite significant. While we do not study a detailed hardware design, we believe such macroscopic reconfiguration can be achieved with high efficiency, and is much less aggressive than recent proposals for reconfiguration deep into the core [19].

So in summary, we focus on configuration of the bulk of the on-chip memory hierarchy, setting cache sizes and number of levels in a modern technology context. In this paper, we explore three critical questions for such reconfigurable memory hierarchies:

- What is the potential energy efficiency benefit to application programs of a configurable memory hierarchy?
- How much configuration flexibility is required to achieve that per-application benefit?
- Can we exploit that benefit automatically, configuring the system to best match the application needs?

. To explore these questions, we use a diverse workload of 37 benchmarks to explore a wide range of memory hierarchy configurations of varying depth and size. Using CACTI [20] and DRAM energy modeling, we identify the best memory hierarchy configuration for each application, optimizing performance, energy, and power. With this

foundation, we explore the question of how much configurability is needed to capture the majority of the energy efficiency benefit – whilst preserving performance, and further explore how to choose those configurations based on application properties. The major contributions of this paper are the results of this inquiry and include:

- The opportunity for reconfiguration to reduce memory hierarchy energy is large – over 70% with no performance loss. If modest performance degradation is acceptable, as much as 95% of memory hierarchy energy can be saved.
- Across a wide range of leakage models, this picture changes little, with reconfiguration achieving 69% potential savings with no loss of performance assuming 90% leakage reduction by techniques such as drowsy caches.
- While our study considers over two thousand candidate memory hierarchies, cluster analysis identifies ten configurations that deliver most of the benefit, simplifying the configurable hardware structures needed. These ten configurations can deliver 63% energy reduction.
- Finally, we show that automatic configuration selection may be feasible based on reuse distance, demonstrating a 100% success across 37 applications in identifying the best memory hierarchy configuration from the reduced space of 10.

The rest of the paper is organized as follows. Section 2 outlines the workload, memory hierarchy space, and energy models used in the study. In Section 3, we explore the opportunity for energy reduction by customizing the memory hierarchy, also exploring the potential performance impact. Section 4 addresses the question of the range of configurability needed, and Section 5 the challenge of how to automatically select memory hierarchy configurations. Section 6 discusses our results and the most relevant related work, and then we summarize and point out some directions for future work in Section 7.

2 Workload and Energy Models

This work focuses on exploring the opportunity for energy reduction by customizing the memory hierarchy. This investigation is pursued utilizing a defined workload (see Section 2.1), and energy models of the memory hierarchy (e.g. on-chip cache and off-chip DRAM). The energy models incorporate both power and performance models of the hierarchy combined with details about the workload to determine energy usage (see Section 2.2 & 2.3). The goal is to then determine best memory configuration with respect to energy usage for each application in the workload.

2.1 Workload

The workload is selected to cover a broad variety of computational domains, including traditional HPC applications as well as emerging data intensive applications. The workload consists of 37 different applications from 6 different benchmark suites: NPB, PARSEC, Mantevo, UHPC, Minebench, and Biobench, and include a variety of compute and memory-bound applications as well as a range of server and desktop applications.

The NAS Parallel Benchmarks (NPB) is a collection of kernels and pseudo-applications that represent computation and data movement in computational fluid dynamics workloads [21]. The Princeton Application Repository (Parsec) [22] is a general workload suite designed to represent emerging applications, including vision, analytics, and image processing. The Mantevo suite is a collection of proxy applications intended to mimic large scale Finite Elements and Molecular Dynamics applications [23]. The Ubiquitous High Performance Computing (UHPC) application suite is a set of five applications representing DoD workloads and were used as reference in DARPA's UHPC program [24]. Minebench is a data mining benchmark suite with applications in different algorithmic categories such as clustering, classification, and optimization [25]. Biobench is a benchmark suite of bioinformatics applications including sequence alignment and assembly code [26]. The benchmark suites and the applications used in our study are listed in Table 1.

Table 1: Benchmark Suite

Package	Application
NPB	BT, CG, DC, EP, FT, IS, LU, MG, SP, UA
PARSEC	blackscholes, bodytrack, facesim, ferret, freqmine, swaptions, fluidanimate, vips, canneal, dedup, streamcluster
Mantevo	miniMD, miniFE, HPCCG
UHPC	chess, graph, lulesh, md, sensor
Minebench	ECLAT, Baeysian, semphy
Biobench	mummer, clustalw, hmmer, phyilip, fasta

2.2 Energy Access Models

In determining the energy usage for an application for a given memory hierarchy we start by estimating energy per access, latency, and leakage by using CACTI [20, 27] for the 32nm technology node. We model DRAM by assuming 400 cycle access latency and 40pJ/bit access energy [27]. Table 2 lists individual cache sizes and their characteristics.

For the remainder of the paper we will use the following terminology when referring to cache configurations:

- *L* as the number of cache levels (L+1 corresponding to DRAM)
- *S(i)* as the size of level i,
- *E(i)* as the access energy of level i, and
- *L(i)* as the leakage power of level i.

Table 2: Cache Specifications

Size (KB)	Associativity (ways)	Access Time (cycles)	Access Energy (nJ)	Leakage (mW)
8	4	2	0.08	0.007
12	6	2	0.08	0.008
16	4	2	0.08	0.008
24	6	2	0.13	0.014
32	8	3	0.17	0.020
48	6	3	0.18	0.023
64	8	3	0.18	0.026
96	6	3	0.18	0.035
128	8	3	0.19	0.043
192	6	5	0.19	0.072
256	8	7	0.31	0.100
384	12	7	0.52	0.128
512	8	7	0.53	0.156
768	12	10	0.53	0.242
1024	16	13	0.54	0.328
1536	12	13	0.55	0.528
2048	16	14	1.03	0.728
3072	12	16	1.07	0.947
4096	16	18	1.26	1.165
6144	12	19	1.45	1.671
8192	16	19	1.47	2.176
12288	12	19	1.68	3.120
16384	16	20	1.87	4.063
24576	24	21	2.00	5.832
32768	16	23	2.12	7.600
49152	24	25	2.28	11.644
65536	16	28	2.43	15.688

The data in Table 2 provide a starting point in determining the energy require for each data access within the execution of an application. The next step is determining the number and type (e.g. L1 cache, L2 cache, L3 cache, or DRAM) of accesses within an application's execution and the energy cost associated with those for a given memory configuration.

2.3 Energy Usage Model

For a given workload, energy consumption depends on the memory references generated by the workload, the configuration of the memory hierarchy, and how the references are serviced within the memory hierarchy. Specifically, the energy consumed in servicing a memory reference depends on the levels of the memory hierarchy involved; this is captured by the cache hit rate for an application.

2.3.1 Time Model

In order to model performance and power of each configuration we also need to model how execution time varies on each configuration. To model execution time we scale the running time measured on a reference system according to variations in Average Memory Access Time (AMAT). For each configuration we compute the memory access time based on hits and latencies as shown in Equation (1).

$$(1)\quad \sum_{i=1}^{L+1} h(i) \times T(i)$$

Then, we use the ratio between the modeled memory access time for a configuration to the modeled access time of the reference system to scale the running time as measured on a reference system.

This modeled running time is then used to calculate leakage for the applications execution and to compare configurations by their performance.

2.3.2 Energy Consumption Model

Finally, to determine the energy usage for each configuration, we calculate both dynamic energy and leakage. Dynamic energy is accounted by the linear combination of access energy and hits shown in Equation (2).

$$(2)\quad \sum_{i=1}^{L+1} h(i) \times E(i)$$

Leaked energy is modeled as the product of modeled leakage power for the given configuration and modeled running time.

2.4 Modeling Cache Hit Rate of the Workload

In order to calculate the energy usage for an application and target configuration we need to simulate the cache hit rate of the application on the target architecture. In modeling the cache hit rates for a target configuration for a given cache configuration via cache simulation we supply a cache simulator with the address stream generated by the memory references of each application in the workload. This is accomplished by employing the PEBIL Toolkit [28], a binary instrumentation toolkit. PEBIL is a tool that takes as input the application binary. PEBIL can disassemble the binary, analyze it, and insert instrumentation. PEBIL can be used to gather static information such as the types of

instructions in basic-blocks and block membership to loops and functions. PEBIL can also insert code into an application to gather information at runtime, which is a feature used by its cache simulation tool. The cache simulation tool is designed to instrument an identified set of basic-blocks in an application (e.g. memory references) and capture the memory addresses of those blocks during the execution of the application. To conserve time and space the address stream fed to a cache simulator on-the-fly while the application is running. PEBIL's cache simulator is capable of concurrently simulating many different cache structures, beyond just the machine being used to run the application. The result is the hit and miss counts for the application on the target configuration(s).

3 Energy Savings with Reconfigurable Memory Hierarchies

To compute the potential energy savings, we used the PEBIL infrastructure for functional simulation of the memory hierarchy (producing hit/miss rates and performance) and combined it with the energy consumption model in Equation (2) to model the energy used by the memory hierarchy. Energy savings are computed relative to a cache configuration based on a typical Intel Sandy Bridge CPU with L1 of 32KB, L2 of 256KB, and L3 of 16MB [29] (one of the configurations).

We extracted hit/miss data for each application and cache configuration and applied the energy model, generating 98,124 data points. From these data points, we selected the optimal configuration with respect to a range of objectives in the following sections. In this section, we explore optimization with respect to energy usage.

The cache configuration search space is bounded at 8KB (minimum L1 cache size) and 64MB (maximum cache size). Parameters such as associativity, banking, and access latency, are tuned to approximate existing architectures and kept constant for any given size. The resulting search space includes one-level, two-level, and three-level configurations, with the constraint that any given level has to be at least twice as big as the previous level. In total, the search space includes 2,652 cache configurations. Figure 1**Error! Reference source not found.** shows the points in the configuration space.

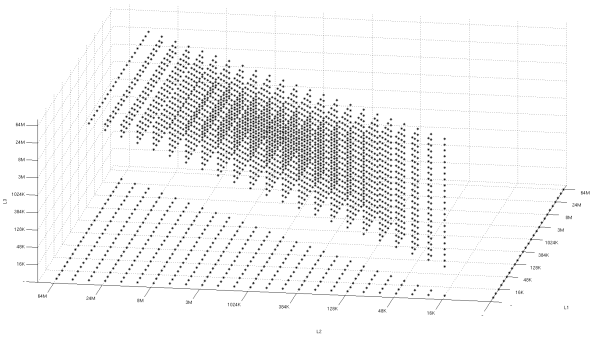

Figure 1: Configuration space.

Optimal energy usage configurations within this space were determined using three different criteria. In the first the space was searched for the most energy efficient configuration regardless of its effect on the workloads overall performance. There are many configurations which offer great energy savings with small performance penalty. We explore this configuration space in Section 3.1 labeled Optimizing Energy Efficiency without regard for Performance. Next we restrict the space by only allow configurations which do not affect the performance negatively in Section 3.2. Finally, in Section 3.3, we explore the same performance preserving space but with the addition of various leakage reducing techniques.

3.1 Optimizing Energy Efficiency without Regard for Performance

In optimizing for energy savings without restrictions we can see dramatic benefits, but at a cost in performance. Remarkably, savings for each (and every) application is greater than 80% saving with an average 95% saving, as shown in Figure 2. However, the memory hierarchy energy saving comes at a significant performance cost[1]. All the energy-optimal configurations selected have no L3, and the majority do not have an L2 either. The results is that many of the applications run significantly slower, and there is an average 1.4 slowdown over the entire workload. Essentially, all of these configurations trade performance for leakage energy in L3 as soon as that cost becomes greater than the energy cost to access DRAM. Often the L1 and L2 caches capture most of the useful locality. Another aspect of this selection is that applications with high locality can enjoy high energy saving, because the overal dynamic energy cost in accessing DRAM is low and easily outweighed by leakage.

[1] And this greater runtime will increase the static power costs in the rest of the system (i.e. datapath, rendering the overall choice suboptimal for energy. However, we focus on the memory hierarchy energy here.

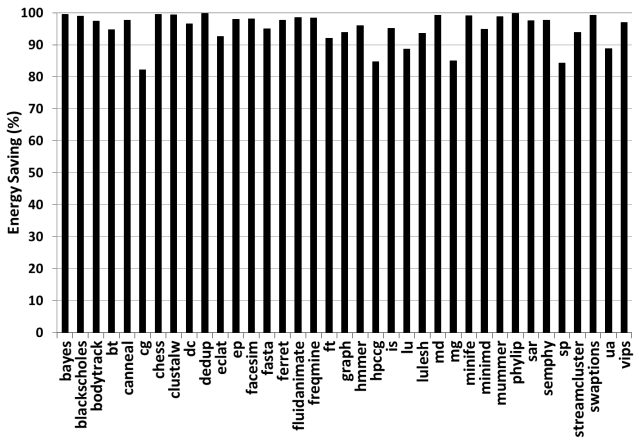

Figure 2. Energy savings with performance oblivious selection.

If we perform a similar optimization for power instead of energy we see large reduction in power but an even worse effect on performance because in a model in which power is averaged over execution, both lower energy and longer runtime contribute to reduce power consumption. Figure 3 illustrates the power reduction for each application in the workload but Figure 4 highlights the difference in performance when optimizing for energy verse power. Figure 4 shows how the optimization for power results in significant performance penalty with an average speedup of 0.5 times the reference system while the average speedup when optimizing for energy yields a speedup of 0.7 time the reference.

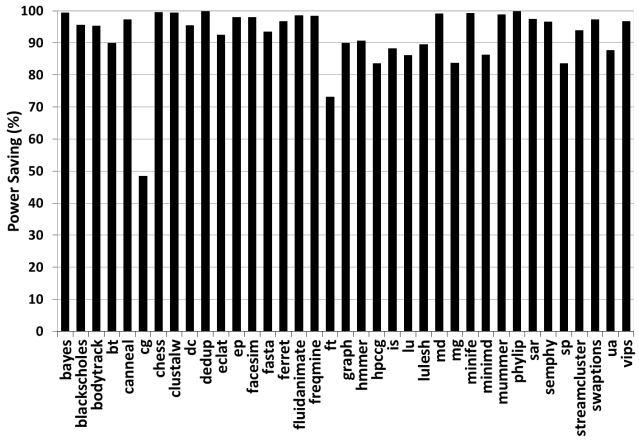

Figure 3. Optimizing for power with performance oblivious selection.

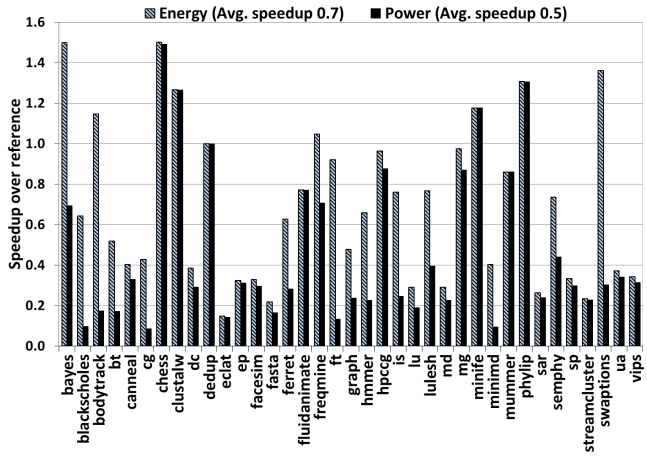

Figure 4. Relative speedup for each application when optimizing for energy and power.

3.2 Optimizing Energy Efficiency while Preserving Performance

To refocus the analysis onto more practical configurations we restrict the selection of the optimal configuration to those that maintain performance. This eliminates many low energy configurations, so, the average energy saving drops to 70%. However, despite the restriction, the energy saving is still considerable and comes with no slowdown in any of our 37 application programs. In fact, overall there is an average 1.2x speedup. Energy savings for each applications are shown in Figure 5.

While the average saving does not drop significantly, some of the applications enjoy a much lower savings as they are well matched to the reference memory hierarchy configuration. Nevertheless, in many cases we observe high savings and the average is still high.

With the performance restriction, five applications had best energy efficiency with a single-level cache (L1) or modest size, reflecting the fact that the remainder of their data was streamed – and no size of L2 or L3 would be a net energy (and performance) win. Half of the applications did not benefit from an L3 cache for similar reasons. These results show the major disincentive for large on-chip caches that leakage entails. From the large memory hierarchy configuration space, nearly every application best matches a unique configuration with 32 different configurations selected for the 37 applications. This substantiates the use of a large configuration space to allow the unique needs of each application's locality structure to be exhibited. Deeper analysis of the configuration space is described in Section 4.

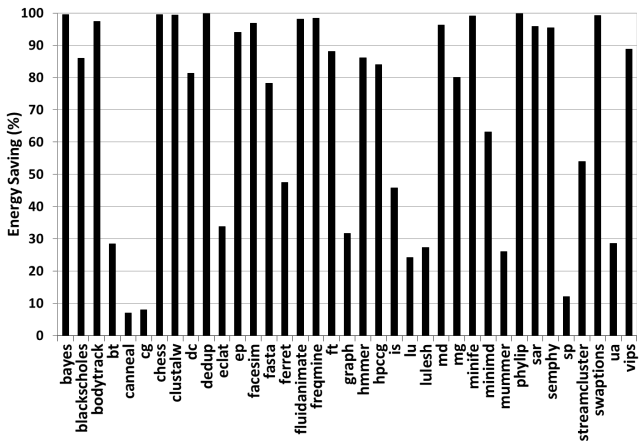

Figure 5. Energy savings preserving performance.

3.3 Energy Efficiency Opportunity vs. Leakage Model

CPU's with large on-chip memory hierarchies can have significant leakage power. For example, in our CACTI-based modelling studies using 32nm process models, the leakage energy is generally the dominant energy cost in the memory hiearchy. To test the sensitivity of our assessment of opportunity, and to take into account the potential of the numerous approaches to reducing cache leakage energy, we reassess the potential benefit under four different power models, as described in Table 3. This study expands the 98,124 data points to nearly 400,000 configuration/performance/energy points. The results are presented in Figure 6. In all cases, we also include the DRAM access energy as described in Section 2.

Table 3: Power models, based on CACTI modeling for cache size, dynamic and static power

DynOnly	**Dynamic Only** Includes dynamic power only, no accounting for leakage
AggLkR	**Aggressive Leakage Reduction** Includes dynamic power and 10% of the leakage power. Models very aggressive leakage reduction, including transistor optimization, drowsy cache, and other techniques with optimistic assumptions
LkR	**Leakage Reduction:** Includes dynamic power and 50% of the leakage power. Consistent with the published benefits for drowsy caches and transistor threshold optimization leakage reduction.
Full	**Full Power**: Includes dynamic power and leakage power

The results in Figure 6 show that the results for the aggressive leakage reduction (AggLkR), leakage reduction (LkR), and full power (Full) are all quite similar. That is, the specific leakage model we choose doesn't affect the proportional benefit of reconfiguration to the best cache hierarchy structure. In a few cases, the dynamic only model (DynOnly) shows significantly better percentage reduction in energy. These results reflect the benefits of appropriately sizing and managing dynamic access costs, but reflect much smaller absolute gains in energy efficiency than any of the models including leakage energy. The average energy efficiency opportunity under each leakage model is summarized below:

Table 4: Energy saving for different power models.

Model	Average Energy Saving (%)
DynOnly	44
AggLkR	65
LkR	69
Full	70

4 Flexibility Needs in Configurable Memory Hierarchies

Given a large configuration space, applications tend to select different optimal configurations. Having flexibility favors fine selection of the optimal configuration and, as observed in Sections 3.1and 3.2, most applications select a unique configuration. However, restricting the space in terms of complexity has a clear benefit in the selection and implementation of reconfigurable caches.

We gradually reduce the configuration space and observe how much energy saving is retained when reducing the search space. First, we coarsen the granularity of the space by removing all cache sizes that are not a power of 2. As a further refinement step we restrict the space around the reference cache by allowing only an 8x variation. In this way, for each cache level a configuration cannot be more than 8x smaller or larger than the corresponding level in the reference although we keep the option for not having L2 or L3 caches. Finally, we do a hierarchical clustering of the selected configurations to group similar configurations. The final configuration space counts 10 configurations, which are listed in Table 5. The configuration spaces corresponding to the complete space and the refinement steps described are called respectively *Full*, 2^n, *Restricted,* and *Cluster*.

Table 5: Final configuration space (in kilobytes)

L1	16	16	16	8	16	16	16	16	16	16
L2	256	1024	2048	32	64	64	128	128	256	512
L3	0	0	0	2048	2048	8192	4096	16384	2048	16384

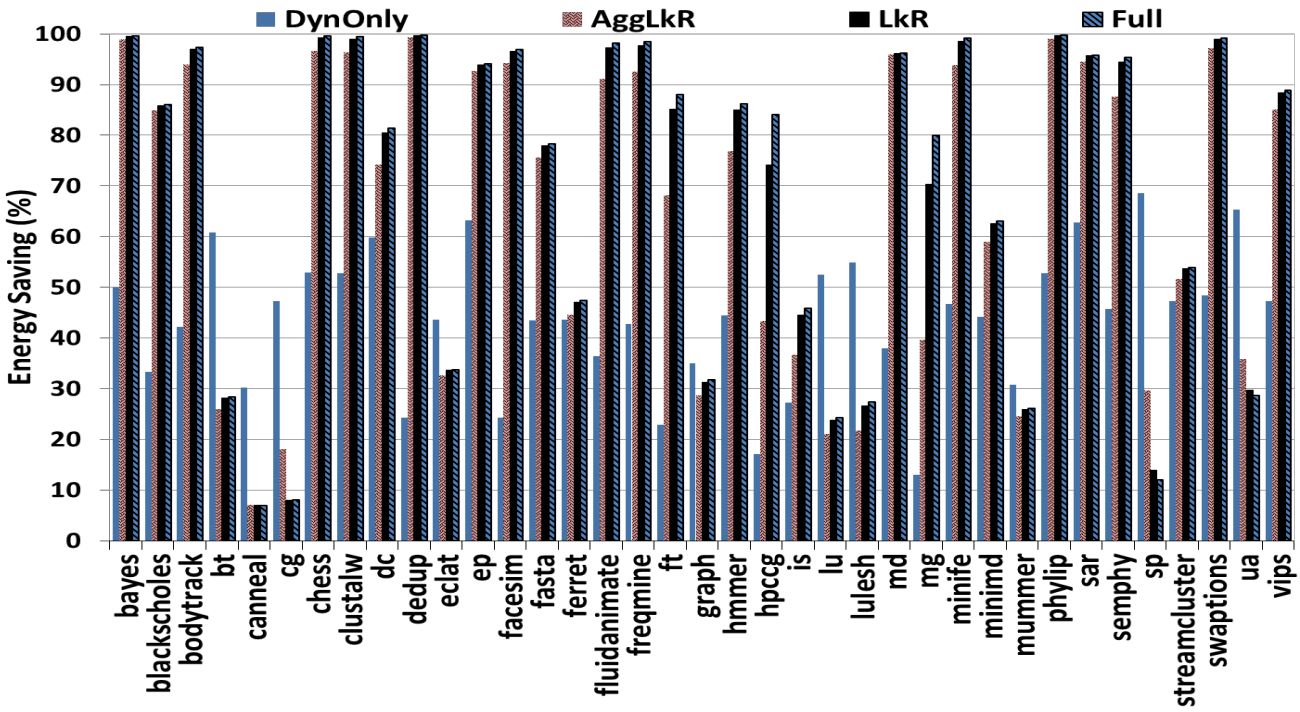

Figure 6. Energy efficiency opportunity as leakage model is varied, starting with no leakage to full 32nm leakage without special compensation techniques.

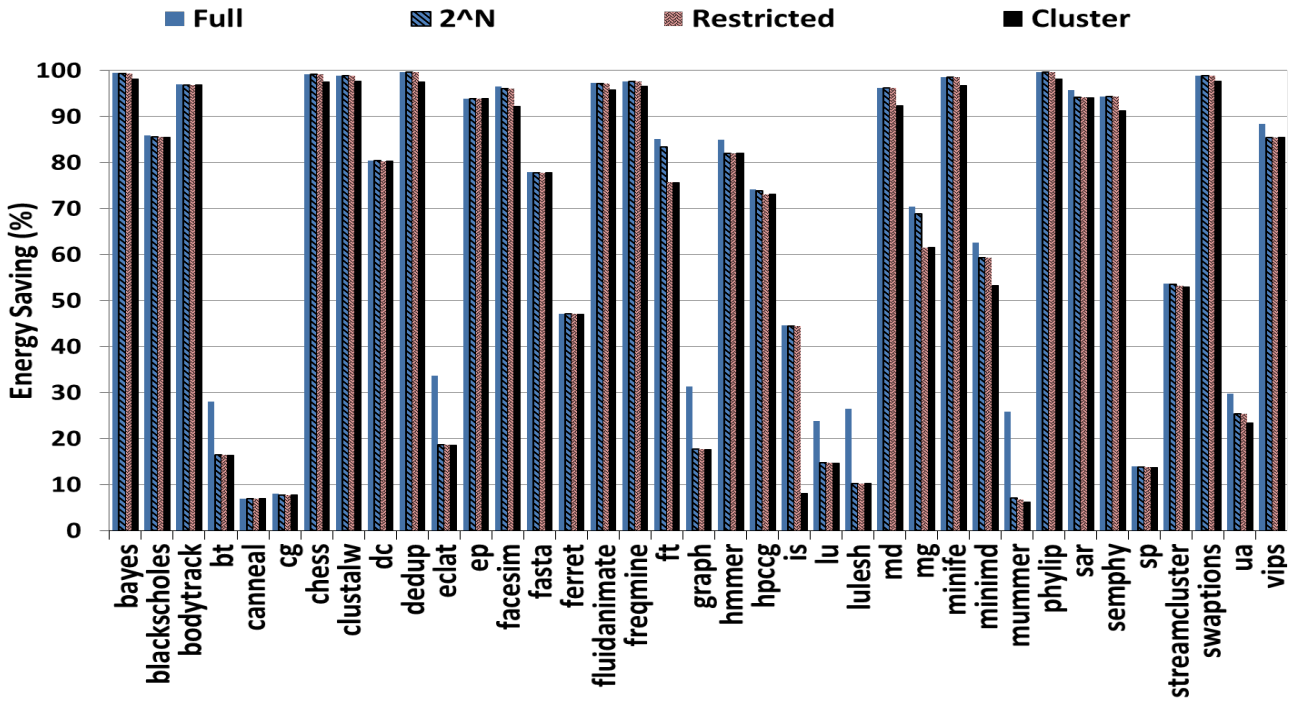

Figure 7. Energy saving in restricted configuration spaces.

As shown in Figure 7, a small amount of configuration flexibility can garner a large benefit in energy efficiency. With our clustered technique perhaps obtaining the largest benefit at a manageable configuration space cost. We believe this demonstrates the effectiveness of our clustering.

The slow progressing decrease in energy saving is also shown in Table 5. The average energy saving slowly decreases from 68.8% to 63.7% percent, which is small especially considered that it corresponds to a 260x reduction in the configuration space.

Table 5 also shows that even within restricted space applications continue to select unique configurations, for the most part, and span the whole 10 configurations in the most *Cluster* space. This indicates that the workload selected represents different locality patterns, and that it is important to maintain a diverse set of configuration to satisfy the diversity in locality patterns.

Table 6: Cache configuration search spaces.

Space Name	# Configurations in search space	# Unique Configs. selected	Avg. Energy Savings (%)
Full	2652	33	68.8
2^N	469	26	66.0
Restricted	224	23	65.5
Cluster	10	10	63.7

Finally, we observe that even in the *Cluster* space there is an even distribution of applications to configurations, with two most popular outliers. The two outliers have a 16KB L1 and 256KB L2, and a 128KB L1, 128KB L2, and 16MB L3. These represent recurring patterns with very high locality in one case, and relatively low locality in the other, and in general, half of the applications have good locality and gravitate towards *L3-less* configurations. The distribution is shown in Figure 8.

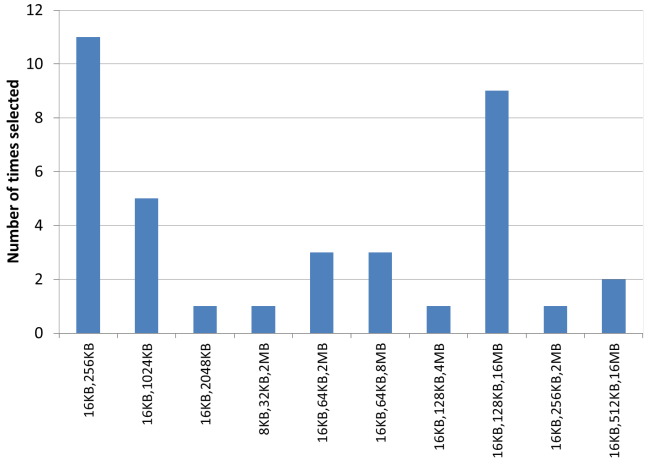

Figure 8. Histogram of selected cache configurations in clustering.

Further reducing the space reveals how far we can reduce the space while saving energy. First we reduce the clustering to 5 clusters, then we select the best 2 configurations, and finally converge to a single configuration. During the convergence process it appears that with fewer than 5 configurations, in order to preserve performance, the reference configuration must be part of the configuration space. That means that for certain applications, energy savings can be realized only with a certain degree of reconfigurability to improve over the reference. Finally, with one configuration the space converges to the reference, which is the configuration that guarantees no performance loss, indicating that the reference itself strikes a good balance between performance and energy consumption for the generic case. Figure 9 shows how energy saving increases from the reference towards larger configurations spaces. In addition, Figure 9 shows how energy saving improves rapidly to 45% with 2 configurations, 53% with 5, and up to 64% with the 10 configurations selected (*Cluster)*, and more gradually afterward, as previously observed (see Table 6).

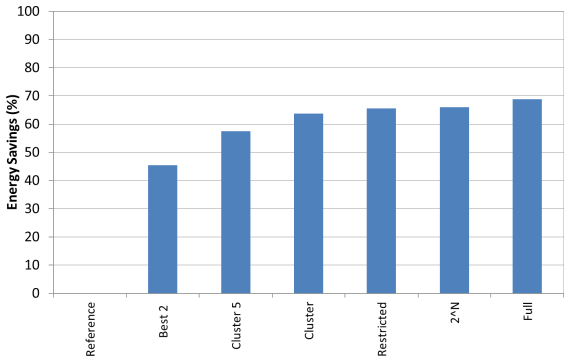

Figure 9. Energy savings vs. Memory hierarchy configuration space.

5 Towards Automatic Memory Hierarchy Configuration Selection

To be practical, the selection of the optimal configuration needs to be simplified and reformulated to the extent that it can be automated. In particular, it cannot depend on costly simulation-based models. As a first step in this direction, we explore reuse distance a tool to characterize application and that can be used to model energy consumption.

Reuse distance has been studied and utilized as a platform independent metric representative of cache behavior. We explore the use of reuse distance as an application characterization tool to select memory hierarchy configuration.

Assuming 64B cache lines, we use the following definition of reuse distance: *A 64B-aligned memory line is referenced with reuse distance d, where d is the number of unique memory lines referenced since the last reference to that memory line [30]*. From this definition it follows that a reference must be satisfied by a fully-associative LRU cache with capacity greater than or equal to the reuse distance (a compulsory miss has infinite reuse distance).

We use a Pin tool [31] to collect reuse distance data for each application in the workload. During execution Pin supplies the address stream of an application, gathered similarly to PEBIL, to a library that collects a histogram of reuse distances. Each element of the histogram represents the number of references within a reuse distance range that corresponds to a cache size in our design space. For convenience, we used bins with ranges matching the points of our cache configuration space with an additional "*anything greater than"* bin to capture distances outside the design space (beyond 64MB and to infinity for compulsory misses).

We apply reuse distance by estimating performance and energy usage for a given cache configuration based on what it would predict as hits and misses at each level of cache and memory based on the reuse distance histogram. From the histogram we infer hit counts assuming that references in a bin for distances up to *d* will hit the smallest cache with size greater than or equal to *d*; if no such cache level exists then those are counted as references to DRAM. Equation (3) defines hits by level for caches and DRAM, with the number of bins, bin upper bound, and bin reference count represented by B, U(j), and R(j) respectively [30]. For simplicity, in Equation (3) it is implied that h(0)=0 and that S(L+1)=∞

$$(3)\quad h(i) = \sum_{\{j \mid U(j) \le S(i) \wedge U(j) > S(i-1)\}} R(j)$$

5.1 Selecting in the Full Configuration Space

The methodology that we propose uses reuse distance data to model energy consumption and drive the selection of the configuration. We evaluate the selection by comparing the energy savings to the savings previously obtained while preserving performance, and by assuming leakage reduction.

Over the full configuration space, reuse distance based selection achieves comparable savings in most applications, and sometime even higher savings. While this is a successful result, it also shows an unexpected behavior. We use simulation as the underlying reference for both energy and time in the comparison, but the selection based on reuse distance has its own time estimate. As a result, it may select configurations that were not otherwise considered due to performance degradation, save more energy, but effectively degrade performance. In fact, the reuse distance based selection results in slightly lower average speedup (1.19 vs. 1.20) and a slowdown in some of the applications. The energy savings with reuse distance based selection are shown in Figure 10.

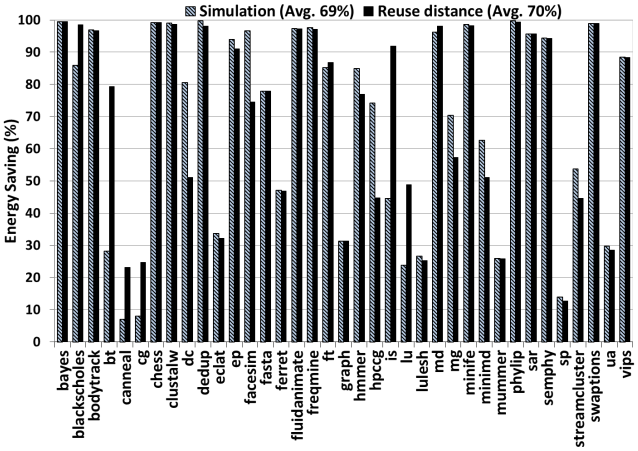

Figure 10. Simulation vs. Reuse distance (full space).

5.2 Selecting in the Constrained Space

Reducing the configuration space simplifies the selection of the optimal configurations because there are fewer points to choose from, and because the variation in energy consumption between points is greater.

For the *Cluster* configuration space, the reuse distance based selection successfully selects the optimal configuration for every single application. This is a very positive result indicating that, despite the inaccuracy due to the inherent approximation of the model, when the search space is reduced reuse distance can effectively selects the optimal configuration. This success is illustrated in Figure 11, which shows the reuse distance-based selection matching the potential benefit for all 37 applications, and delivering 64% energy reduction.

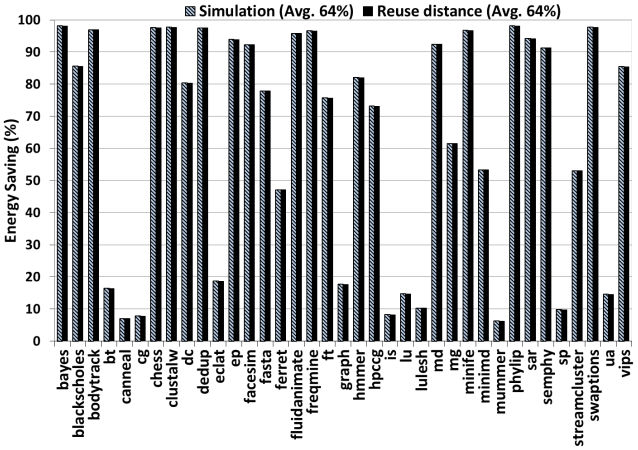

Figure 11. Comparing the selection based on exhaustive search vs. Reuse distance-based selection.

6 Discussion and Related Work

Our results show that significant energy savings can be achieved if the memory hierarchy design can be matched to the application's needs. The design of reconfigurable memory hierarchies aim to deliver these potential benefits as the energy consumed by memory hierarchies is an increasingly important element of overall system energy efficiency. As such, our results cannot be fairly compared to the wealth of research on memory hierarchy optimization that targets a single fixed design and aspects of organization and cache management policies to increase performance and many other attributes. This rich vein of research underlies the design of nearly all microprocessor designs today.

Numerous elements of memory hierarchy research employ elements of adaptation, a close relative to reconfiguration, to customize their behavior on a per-cache line basis or over short periods of time to better match application needs. Examples include adaptive caches [17, 18], adaptive protocols [32], and prefetching engines [12-14]. These systems typically focus on improving execution performance and perhaps secondarily reducing miss traffic. These contrast to our goal of exploiting reconfiguration in a memory hierarchy – per application -- to improve energy efficiency.

Work in the early 2000's [15, 16] explored processor, TLB, and cache structures to optimize energy efficiency, focusing on dynamic energy, in an era of higher clock rates, much smaller caches, and less leakage. As a result, reconfiguration focused on access method (sequential way access, tags then array, and even TLB energy optimization. As outlined in the introduction, these studies typically assumed constant total cache capacity, rather than our unconstrained dark silicon framework. Further, we focus on cache block/array level reconfiguration, enabling us to optimize dynamic energy, leakage, and even the remaining leakage after aggressive leakage reduction has been pursued. In short, for current device technology (billions of transistors operating at a few GHz) and chip-scale (16MB caches), access latencies are less critical and not using large sections of a chip ("dark silicon") is conceivable. Remarkably, turning entire sections of caches off in many cases reduces overall energy consumption.

The feasibility of cache reconfiguration is well-established with studies going back over a dozen years [15-17]and the well-known work on smart memories as a landmark effort [33]. In fact, recent proposals [19] go even further, proposing unifying register file, scratch, and cache memories that require configuration deep into the core, substantiating the viability of reconfigurable structures at even the highest levels of the memory hierarchy.

Reuse distance is a well-established and widely-used platform-independent approach to characterizing application data locality and reference structure [30, 32, 34-43]. It has been used to model program locality [34, 36], to select ISA-specific hints for locality [37], to predict performance [38], and to improve locality by code reordering [35, 39, 41]. More recently, reuse distance has been extended to model cache sharing and interference in multi-core processors [40, 42]. Finally, as a way to drive the design of performance optimal caches, reuse distance has been used to select sharing and size of LLC [43]. Ours is yet another use of reuse distance – to select the configuration of a reconfigurable memory hierarchy.

Modern memory hierarchies have grown to megabytes [29] and combined with deep submicron CMOS technologies, leakage (or static) power has become an important concern. As we discussed in Section 3.3, a wealth of circuit techniques have been developed to reduce the leakage energy penalty in large on-chip caches, including drowsy caches and transistor engineering [17], cache decay [18], etc. These techniques have been shown to reduce leakage energy by 50-70%, so in Section 3.3 we have explored a set of possible design points exploiting these techniques. These studies show that such reductions in leakage power can affect the energy savings modestly, but they do not strongly affect the selection of optimal hierarchy, nor the relative benefit. Given the poorer voltage scaling of SRAM cells due to instability, we expect leakage power is likely to remain an important factor in memory hierarchy energy.

7 Summary and Future work

Our results show that the energy benefits of reconfigurable memory hierarchies can be significant, ranging from 95% for extreme reconfigurability to 64% for simple reconfigurability; all while maintaining performance. Further, simulations show that these benefits are robust across a range of cache energy models, and that amongst the constrained memory hierarchy space, characteristics such as reuse distance can reliably select a good memory hierarchy configuration.

While we have studied the potential energy benefits of a reconfigurable memory hierarchy, much work remains to explore the detailed design and implementation of such memory hierarchies. For example, studies of the interplay between a wide range of cache features (sophisticated replacement, associativity, write policy, etc.) and the reconfigurable structure should be studied. Further, study of the impact of parallel (multi-core) access is clearly important. Finally, while our results show the promise of reuse distance to automatically select cache configuration, a wealth of static and dynamic techniques bear investigation to reduce the cost and increase the accuracy of automatic configuration.

8 Acknowledgements

This work supported in part by the National Science Foundation under award NSF OCI-10-1057921 and the

Defense Advanced Research Projects Agency under award HR0011-13-2-0014. The contents do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

9 References

[1] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design Of Ion-implanted MOSFET's with Very Small Physical Dimensions," *Proceedings of the IEEE,* vol. 87, pp. 668-678, 1999.

[2] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger, "Power Limitations and Dark Silicon Challenge the Future of Multicore," *ACM Transactions on Computer Systems (TOCS),* vol. 30, p. 11, 2012.

[3] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM,* vol. 54, pp. 67-77, 2011.

[4] ITRS. (2012). *2012 International Technology Roadmap for Semiconductors*. Available: http://www.itrs.net/Links/2012ITRS/Home2012.htm

[5] S. Gupta, S. Feng, A. Ansari, S. Mahlke, and D. August, "Bundled execution of recurring traces for energy-efficient general purpose processing," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 12-23.

[6] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam*, et al.*, "Dyser: Unifying functionality and parallelism specialization for energy efficient computing," 2012.

[7] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, "QsCores: trading dark silicon for scalable energy efficiency with quasi-specific cores," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 163-174.

[8] "OMAP 5 Mobile Application Platform," *Texas Instruments*.

[9] "Introducing Tegra 4, Worlds Fastest Mobile Processor," *NVIDIA*.

[10] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *Computer Architecture News,* vol. 23, pp. 20-24, 1995.

[11] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*: Morgan Kaufmann, 2011.

[12] I. Chung, C. Kim, H.-F. Wen, and G. Cong, "Application data prefetching on the IBM blue gene/Q supercomputer," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, 2012, pp. 1-8.

[13] S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Computing Surveys (CSUR),* vol. 32, pp. 174-199, 2000.

[14] T.-F. Chen, "An effective programmable prefetch engine for on-chip caches," in *Proceedings of the 28th annual international symposium on Microarchitecture*, 1995, pp. 237-242.

[15] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, 1999, pp. 248-259.

[16] 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 ACM/IEEE international symposium on Microarchitecture*, 2000, pp. 245-257.

[17] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Single-v DD and single-v T super-drowsy techniques for low-leakage high-performance instruction caches," in *Proceedings of the 2004 international symposium on Low power electronics and design*, 2004, pp. 54-57.

[18] Z. Hu, S. Kaxiras, and M. Martonosi, "Let caches decay: reducing leakage energy via exploitation of cache generational behavior," *ACM Transactions on Computer Systems (TOCS),* vol. 20, pp. 161-190, 2002.

[19] M. Gebhart, S. W. Keckler, B. Khailany, R. Krashinsky, and W. J. Dally, "Unifying Primary Cache, Scratch, and Register File Memories in a Throughput Processor."

[20] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," presented at the Proceedings of the 35th Annual International Symposium on Computer Architecture, 2008.

[21] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, L. R. Carter, L. Dagum*, et al.*, "NAS Parallel Benchmark Results," *International Journal of Supercomputing Applications,* vol. 5, 1991.

[22] C. Bienia, "Benchmarking Modern Multiprocessors," Princeton University, 2011.

[23] M. A. Heroux, D. W. Doefler, P. S. Crozier, J. Willenbring, M., C. H. Edwards, A. Williams*, et al.*, "Improving Performance via Mini-applications," Sandia National Laboratory2009.

[24] DARPA, "Ubiquitous High Performance Computing (UHPC)."

[25] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "MineBench: A Benchmark Suite for Data Mining Workloads," in *Workload Characterization, 2006 IEEE International Symposium on*, 2006, pp. 182-188.

[26] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C. W. Tseng*, et al.*, "BioBench: A Benchmark Suite of Bioinformatics Applications," in *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, 2005, pp. 2-9.

[27] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, 2010, pp. 363-374.

[28] M. Laurenzano, M. Tikir, L. Carrington, and A. Snavely, "PEBIL: Efficient Static Binary Instrumentation for Linux.," presented at the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), White Plains, NY, 2010.

[29] "The Sandy Bridge E Processor," *Intel*.

[30] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.,* vol. 9, pp. 78-117, 1970.

[31] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney*, et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.,* vol. 40, pp. 190-200, 2005.

[32] G. Kurian, O. Khan, and S. Devadas, "The Locality-Aware Adaptive Cache Coherence Protocol," in *International Symposium on Computer Architecture (ISCA) (to appear)*, 2013.

[33] M. Ken, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: a modular reconfigurable architecture," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, 2000, pp. 161-171.

[34] C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," presented at the Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, San Diego, California, USA, 2003.

[35] K. Beyls and E. H. D'Hollander, "Platform-independent cache optimization by pinpointing low-locality reuse," in *IN PROCEEDINGS OF INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE*, ed, 2004, pp. 463-470.

[36] K. Beyls and E. H. D'Hollander, "Reuse Distance as a Metric for Cache Behavior," in *In Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, ed, 2001, pp. 617-662.

[37] K. Beyls and E. H. D'Hollander, "Reuse Distance-Based Cache Hint Selection," presented at the Proceedings of the 8th International Euro-Par Conference on Parallel Processing, 2002.

[38] G. Marin and J. Mellor-Crummey, "Cross-architecture performance predictions for scientific applications using parameterized models," *SIGMETRICS Perform. Eval. Rev.,* vol. 32, pp. 2-13, 2004.

[39] G. Marin and J. Mellor-Crummey, "Pinpointing and Exploiting Opportunities for Enhancing Data Reuse," in *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*, 2008, pp. 115-126.

[40] X. Xiang, B. Bao, T. Bai, C. Ding, and T. Chilimbi, "All-window profiling and composable models of cache sharing," presented at the

Proceedings of the 16th ACM symposium on Principles and practice of parallel programming, San Antonio, TX, USA, 2011.

[41] C. Ding and K. Kennedy, "Improving cache performance in dynamic applications through data and computation reorganization at run time," *SIGPLAN Not.,* vol. 34, pp. 229-241, 1999.

[42] D. L. Schuff, B. S. Parsons, and V. S. Pai, "Multicore-aware reuse distance analysis," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010, pp. 1-8.

[43] M.-J. Wu and D. Yeung, "Identifying optimal multicore cache hierarchies for loop-based parallel programs via reuse distance analysis," presented at the Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness, Beijing, China, 2012.