University of Maryland Technical Report UMIACS-TR-2010-10

Memory Performance Analysis for Parallel Programs Using Concurrent Reuse Distance

Meng-Ju Wu and Donald Yeung
Department of Electrical and Computer Engineering
University of Maryland at College Park
{mjwu,yeung}@umd.edu

Abstract

Performance on multicore processors is determined largely by on-chip cache. Computer architects have conducted numerous studies in the past that vary core count and cache capacity as well as problem size to understand impact on cache behavior. These studies are very costly due to the combinatorial design spaces they must explore.

Reuse distance (RD) analysis *can help architects explore multicore cache performance more efficiently. One problem, however, is multicore RD analysis requires measuring* concurrent reuse distance (CRD) profiles *across thread-interleaved memory reference streams. Sensitivity to memory interleaving makes CRD profiles architecture dependent, undermining RD analysis benefits. But for parallel programs with symmetric threads, CRD profiles vary with architecture tractably: they change only slightly with cache capacity scaling, and shift predictably to larger CRD values with core count scaling. This enables analysis of a large number of multicore configurations from a small set of measured CRD profiles.*

This paper investigates using RD analysis to efficiently analyze multicore cache performance for parallel programs, making several contributions. First, we characterize how CRD profiles change with core count and cache capacity. One of our findings is core count scaling degrades locality, but

the degradation only impacts last-level caches (LLCs) below 16MB for our benchmarks and problem sizes, increasing to 128MB if problem size scales by 64x. Second, we apply reference groups [1] to predict CRD profiles across core count scaling, and evaluate prediction accuracy. Finally, we use CRD profiles to analyze multicore cache performance. We find predicted CRD profiles can estimate LLC MPKI within 76% of simulation for configurations without pathologic cache conflicts in $\frac{1}{1200}^{th}$ *the time to perform simulation of the full design space.*

1 Introduction

Practically all high-performance commercial CPUs today integrate multiple cores on a single chip. Given their ubiquity, achieving high performance on multicore processors is an important goal. One key factor determining multicore performance is the memory system. In particular, a crucial issue is how effectively programs can utilize the on-chip cache to mitigate off-chip memory accesses.

Numerous studies have investigated this multicore memory bottleneck [2, 3, 4, 5, 6, 7, 8, 9]. These studies all conduct detailed simulation that vary architecture parameters, usually *core count* and *cache capacity*, to quantify how different designs impact memory traffic and overall performance. A significant problem is the large number of configurations that must be simulated due to the multi-dimensional nature of the design space. Worse yet, this multicore design space is becoming more complex as processors scale.

Today, 4–8 state-of-the-art cores or 10s of smaller cores [10, 11] along with 10s of MBs of cache can fit on a single die. Since Moore's law scaling is expected to continue at historic rates for the foreseeable future [12], multicore processors with 100s of cores and 100+ MB of cache–*i.e.* large-scale chip multiprocessors (LCMPs) [3, 9]–are conceivable after only 2 or 3 generations. As processors scale to the LCMP level, studying multicore memory behavior will become extremely challenging.

A powerful tool that can potentially help architects evaluate multicore memory systems is *reuse*

distance (RD) analysis. RD analysis measures a program's memory reuse distance histogram, or *RD profile*, capturing the program-level locality that is responsible for cache performance. A key feature of RD profiles is *architecture independence*: they can be acquired on one architecture, and then used to predict performance across a large number of cache configurations without additional simulations.

RD analysis is well established for uniprocessors [13, 14, 15, 1], but its use for multicore processors is limited. The problem is locality in multicore processors depends not only on per-thread locality, but also on how simultaneous threads' memory references interleave. To apply RD analysis for multicore processors, the *concurrent reuse distance (CRD) profile* [16] must be acquired across threads' interleaved memory reference streams. Unfortunately, memory interleaving is *architecture dependent*, exhibiting sensitivity to both core count and cache capacity scaling. Hence, it may not be possible to analyze different multicore configurations from a common CRD profile, defeating a key benefit of RD analysis.

Despite their architecture dependence, this paper shows CRD profiles can still be very useful for a specific but important case: parallel programs. Parallel programs usually exploit loop-level parallelism, giving rise to *symmetric threads* that exhibit very similar locality characteristics (*i.e.*, their RD profiles are essentially identical). Even though CRD profiles are architecture dependent, it is possible to reason about how profiles change with architecture scaling when per-thread memory reference streams are similar.

In particular, CRD profiles change only slightly with cache capacity scaling. While cache size can affect performance significantly, symmetric threads tend to speedup or slow down by the same amount as capacity is varied.[1] So, inter-thread performance remains about the same, largely preserving memory interleaving. This suggests CRD profiles acquired at one cache capacity can be

[1]This assumes homogeneous cores with symmetric cache hierarchies.

used to analyze memory behavior across a range of cache capacities.

In contrast, CRD profiles change significantly with core count scaling. Adding cores creates more memory interleaving, degrading locality and shifting portions of CRD profiles to larger CRD values. However, for symmetric threads, this locality degradation is often systematic. In particular, interleaved memory references at the same RD value usually shift by the same amount due to threads' similar reuse patterns. Hence, affected portions of CRD profiles tend to shift in a *shape-preserving fashion*.

Coincidentally, a similar shape-preserving shift has been observed for RD profiles when scaling problem size in sequential programs [1]. Moreover, previous research has shown such profile shifting is predictable. By comparing RD profiles at different problem sizes, the shift can be gauged and used to predict scaled RD profiles [1]. Our work shows such "profile diffing" can predict the impact of core count scaling on CRD profiles as well. This suggests CRD profiles acquired at a small number of core counts can (after prediction) analyze memory behavior across a large number of machine sizes.

Our research investigates RD analysis for parallel programs on multicore processors, making several contributions. First, we characterize how CRD profiles change with last-level cache (LLC) capacity and core count. Our characterization considers processors with 1-256 cores and 4-128MB LLCs. We show 99% of CRD profiles are very similar across LLC sizes, exhibiting *profile window error (PWE)* (a profile similarity metric) of 20% or less. This confirms CRD profiles have low sensitivity to LLC scaling. We also demonstrate profile shift due to core count scaling varies with CRD. Shifting occurs for smaller CRD values, but stops beyond a certain CRD which we call C_{stop}. This implies locality degradation due to core count scaling only impacts LLCs with capacity $< C_{stop}$. For our benchmarks and problem sizes, C_{stop} is between 136KB–22MB. However, if problem size scales by a factor 64x (beyond what we can simulate), C_{stop} can increase to 128MB.

Second, we study CRD profile prediction across core count scaling. Specifically, we employ reference groups [1]–a technique proposed for predicting problem scaling–and evaluate its accuracy for predicting core count scaling. Our results show 46%, 73%, 83%, and 93% of core-count predicted CRD profiles exhibit up to 20%, 40%, 70%, and 100% PWE, respectively. (We also tried predicting problem scaling, and found similar prediction errors). These results show reference groups cannot predict core count scaling perfectly. Nevertheless, a large number of profiles are still predicted with good accuracy.

Finally, to demonstrate their utility, we use CRD profiles to analyze cache performance across our multicore design space and input problems. We find CRD profiles are unable to effectively predict performance in 36% of the configurations due to unanticipated cache conflicts. For the remaining configurations, measured CRD profiles can predict the LLC MPKI to within 69% of simulation, while predicted CRD profiles can predict MPKI to within 76% of simulation. In the latter case, the predicted MPKI is obtained in $\frac{1}{1200}^{th}$ of the time required to simulate the entire design space.

The rest of this paper is organized as follows. Section 2 discusses CRD profiles, and Section 3 characterizes their sensitivity to architectural scaling. Then, Section 4 studies predicting core count scaling, and evaluates prediction accuracy. Next, Section 5 applies our CRD profiles to estimate cache performance. Finally, Sections 6 and 7 discuss related work and conclusions, respectively.

2 Concurrent Reuse Distance

Reuse distance measures the number of unique memory references performed between two references to the same data block,[2] quantifying program-level locality in an architecture-independent fashion. RD profiles–*i.e.*, the distribution of RD values for all memory references in a sequential program–are useful for analyzing uniprocessor cache performance. Because caches can satisfy mem-

[2]Sometimes, reuse distance refers to the *total* memory references between two uses of a data block rather than the *unique* memory references. In this paper, we adopt the latter definition which has also been called stack distance [17].

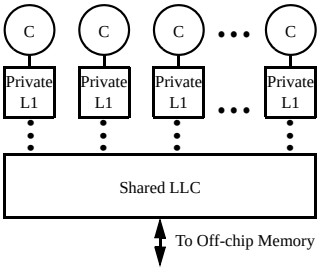

Figure 1. Multicore processor cache hierarchy.

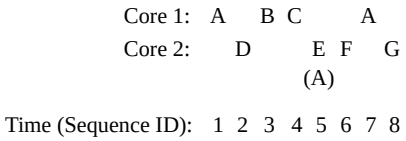

Figure 2. Interleaved memory reference streams from two cores.

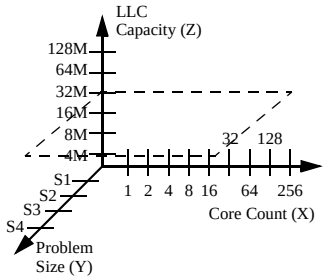

Figure 3. 3-D architecture-problem space (APS). $APS_{x,y,32}$ is highlighted.

ory references with reuse distance $\leq$ the cache size, CS (assuming LRU management), the number of cache misses is simply the sum of all reference counts in a profile above the RD value for capacity CS.

This paper investigates applying RD analysis for shared caches in multicore processors. As illustrated in Figure 1, a typical multicore cache hierarchy integrates multiple levels of cache on chip. Often, caches near the cores are private while caches near the off-chip interface are shared. The LLC, which is the focus of our work, is usually shared by *all* cores.

RD analysis can be extended to handle shared LLCs by computing reuse distance across the interleaved memory reference streams from all on-chip cores. This is known as the *concurrent reuse distance (CRD)* [16]. Figure 2 illustrates CRD for a sequence of interleaved memory references from two cores. In Figure 2, Core 1 references blocks A–C, and then re-references block A, while core 2 references blocks D–G. Core 1's reuse of A has RD = 2, but its CRD, which takes into account the interleaved references from core 2, is 5. In this case, CRD > RD because core 2's references are distinct from core 1's references. The opposite can occur when cores *share data*. For example, if core 2 references block A instead of block E at time 5, then core 1's reuse of A would have CRD = 1, so CRD < RD. Similar to RD profiles, CRD profiles present the distribution of CRD values across all memory references in a parallel program.

As Figure 2 shows, CRD is defined for a particular memory interleaving only. If interleaving

changes, so will the CRD profile. Unfortunately, memory interleaving is sensitive to architecture scaling. In particular, varying LLC capacity changes per-thread execution speed. This implies no single CRD profile can capture locality precisely across multiple LLCs. In addition, varying core count changes the number of interleaved memory streams. Certainly, core count scaling will affect CRD profiles significantly. The key question is given CRD profiles' architecture dependence, can we reason about the memory performance of different multicore configurations from a single (or small number of) CRD profiles? The answer depends on exactly how CRD profiles vary with architecture scaling.

3 Scaling Characterization

This section characterizes how architecture scaling impacts CRD profiles. Our study proceeds in four parts. First, Section 3.1 describes experimental methodology. Then, Sections 3.2 and 3.3 present the LLC capacity and core count scaling analyses, respectively. Finally, Section 3.4 discusses problem scaling.

3.1 Experimental Methodology

We use simulation to characterize the impact of varying LLC capacity and core count on CRD profiles. We also study the impact of varying problem size. (As processors scale to the LCMP level, they will be used to execute larger problems, so understanding problem scaling in the context of machine scaling is crucial). Together, these scaling dimensions form the 3-D architecture-problem space (APS) illustrated in Figure 3. We simulate all points in APS, acquiring the CRD profile at each point, $APS_{x,y,z}$. By comparing profiles across any axis, we can ascertain sensitivity along the corresponding scaling dimension.

The multicore architecture assumed in our study is the *tiled CMP* [18], illustrated in Figure 4. In a tiled CMP, each tile contains a core, a private L1 cache, an L2 cache and directory "slice,"

Number of Tiles	1, 2, 4, 8, 16, 32, 64, 128, 256
Core Type	Single issue, In-order, CPI = 1, clock speed = 2GHz
IL1/DL1	32KB/32KB, 64B block, 8-way, 1 cycle
Total L2 Cache Size	4MB, 8MB, 16MB, 32MB, 64MB, 128MB
L2 Slice	64B blocks, 32-way, 10 cycles
2-D Mesh	3 cycles per-hop, bi-directional channels, 256-bit wide links
Memory channels	latency: 200-CPU cycles, bandwidth: 32GB(1-16cores) and 64GB(32-256cores)

Table 1. Simulator parameters used in the experiments.

and a switch with point-to-point interconnect for a 2-D on-chip mesh network. Tiles are replicated, which increases *all* resources proportionally (*e.g.*, cores, cache, and network). Hence, tiled CMPs are considered scalable [19, 18]. This enables exploring CRD profiles across a large design space on a single architectural platform.

A key part of tiled CMPs are the L2 slices. We manage the aggregate L2 slices as a single logically shared LLC, with no replication or migration across slices. Each cache block is always placed in the same L2 slice, known as the cache block's "home." We assume cache block homes are chunk-interleaved across L2 slices according to their physical address, with chunking factor equal to one way of an L2 slice. This simple organization permits us to study CRD profiles across a large range of shared LLC capacities.

We modified the M5 simulator [20] to model a tiled CMP. Our simulator permits replication at the private L1 caches, so we also added a directory-based MESI cache coherence protocol. We assume full-map directories collocated with their associated cache blocks on the home tile. We also modified M5's memory sub-system to support multiple DRAM channels, each associated with a memory controller on a special "memory tile." In total, we simulate 4 memory tiles evenly spaced on the north and south faces of the chip, providing an aggregate off-chip bandwidth of 32 GB/sec. Lastly, during simulation, our simulator records the CRD profile for the pre-L1 memory reference stream across all cores. CRD is computed at the granularity of 64 bytes, the block size for both the L1 and L2 caches.

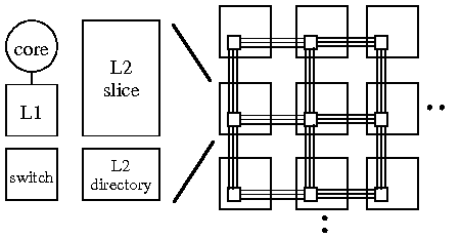

Figure 4. Tiled CMP. Each tile contains a core+L1 cache, an L2 cache and directory "slice," and an on-chip network switch.

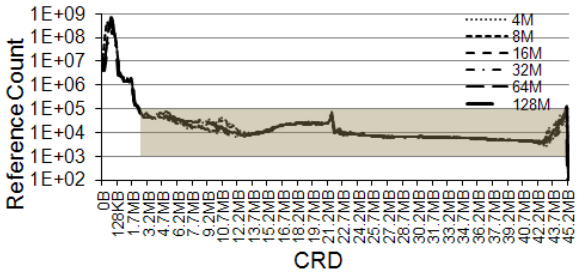

Figure 5. CRD profiles from the Water benchmark across LLC capacity at $APS_{64,S4,z}$.

Table 1 lists the parameters we use. We simulate 1–256 cores and 4–128MB of total L2 cache (LLC). The cores are very simple, each executing one instruction per cycle (in the absence of memory stalls) in program order. However, we model the memory hierarchy accurately, including L1 access, hops through the network, L2 slice access, and DRAM access. We also account for queuing at the on-chip network switches as well as the memory controllers. When scaling the LLC, we always assume the total capacity is evenly divided amongst tiles, so the size of each L2 slice is always the total LLC size divided by the number of tiles.

Table 2 lists our benchmarks: FFT, LU, Radix, Barnes, FMM, Water, and Ocean from the SPLASH2 suite [21], KMeans from the MineBench suite [22], and BlackScholes from the PARSEC suite [23]. For each benchmark, we employ 4 problem sizes, S1-S4, (2^{nd} column of Table 2). We always execute initialization code on a single core, then turn on CRD profiling (and other statistics) and simulate the parallel region for some number of instructions (3^{rd} column of Table 2). For FFT, LU, and Radix, the parallel region represents the entire program, whereas for the remaining benchmarks, it represents 1 timestep of the algorithm.

Figure 3 is labeled with the core counts, LLC capacities, and problem sizes from Tables 1 and 2. There are 216 configurations per benchmark in APS. Across all 9 benchmarks, there are 1,944 configurations.

Benchmark	Problem Sizes(S1/S2/S3/S4)	Insts(M)(S1/S2/S3/S4)	Sim Region
FFT	$2^{16}/2^{18}/2^{20}/2^{22}$ elements	32/139/605/2,610	whole program
LU	$256^2/512^2/1024^2/2048^2$ elements	72/577/4,620/36,990	whole program
RADIX	$2^{17}/2^{19}/2^{21}/2^{23}$ elements	23/90/432/1,728	whole program
Barnes	$2^{13}/2^{15}/2^{17}/2^{19}$ particles	614/2,908/12,796/55,222	1 timestep
FMM	$2^{13}/2^{15}/2^{17}/2^{19}$ particles	217/928/3,793/15,301	1 timestep
Ocean	$130^2/258^2/514^2/1026^2$ Grid	50/200/783/2,879	1 timestep
Water	$10^3/16^3/25^3/40^3$ mols	115/431/1,826/9,564	1 timestep
KMeans	$2^{16}/2^{18}/2^{20}/2^{22}$ Objects, 16 features	492/1,970/7,880/31,499	1 timestep
BlackScholes	$2^{16}/2^{18}/2^{20}/2^{22}$ options	94/377/1,507/6,027	1 timestep

Table 2. Parallel benchmarks used to drive the simulations.

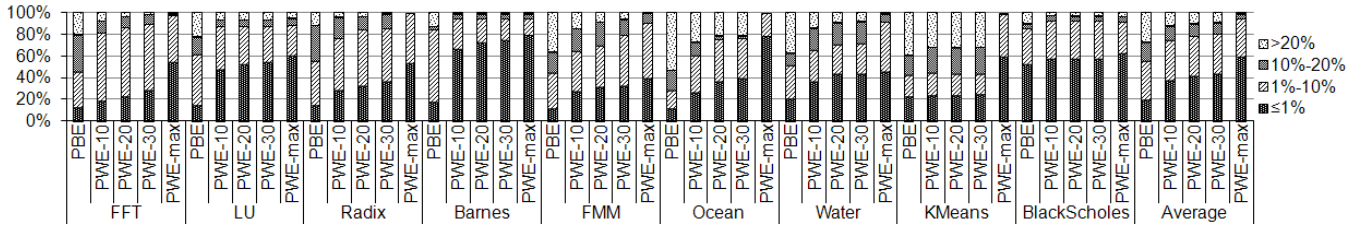

Figure 6. Percentage of CRD profiles whose PBE(C), PWE(C,10), PWE(C,20), PWE(C,30), and PWE(C,$maxbin$) are $\leq$ 1%, 1-10%, 10-20%, and $>$ 20%.

3.2 LLC Capacity Scaling

Figure 5 shows an example of CRD profile variation across LLC capacity scaling. It plots CRD profiles from the Water benchmark all running on 64 cores with the S4 problem size, but varies the LLC size between 4 and 128MB (*i.e.*, profiles at $APS_{64,S4,z}$). Each CRD profile plots reference count along the Y-axis versus CRD along the X-axis. CRD values are multiplied by 64 bytes, the granularity for computing CRD, so the X-axis reports distance in terms of capacity. To enhance readability, reference counts from multiple adjacent CRD values are summed into a single CRD bin, and plotted as a single Y value. For capacities between 0 bytes and 128KB, bin size grows logarithmically; beyond 128KB, bins are fixed at 128KB each.

As Figure 5 shows, CRD profiles change with LLC capacity, so they are indeed architecture dependent. However, the profiles are *very similar*, with significant overlap. While the CRD profiles in Figure 5 are not identical, they exhibit *low sensitivity* to LLC scaling. This is because LLC

scaling speeds up or slows down symmetric threads by similar amounts, as discussed in Section 1. So, profiles tend to remain the same.

To quantify this similarity, we compare CRD profiles across the entire APS. For each benchmark, core count, and problem size, we compare the CRD profiles at capacities C = 4, 8, 16, 64, and 128MB (CRD_C at $APS_{x,y,C}$) against the CRD profile at capacity 32MB (CRD_{32} at $APS_{x,y,32}$ in the dotted plane of Figure 3). For each pairwise profile comparison, we compute the absolute error between reference counts at all pairwise CRD bins (CRD_C[i] and CRD_{32}[i]), and then average the errors from the first to last bin ($maxbin$). We call this the *profile bin error for capacity C, PBE(C).*

$$PBE(C) = \frac{1}{maxbin} \sum_{i=1}^{maxbin} \frac{|CRD_C[i] - CRD_{32}[i]|}{CRD_{32}[i]} \quad (1)$$

Across all $APS_{x,y,z}$, there are 180 CRD profile comparisons per benchmark. In Figure 6, the bars labeled "PBE" report the percentage of profiles with $\leq$ 1%, 1–10%, 10–20%, and $>$ 20% PBE(C). The rightmost "PBE" bar reports the average across all benchmarks. From our experience, a PBE(C) of 1% implies the profiles are essentially identical while a PBE(C) of 1–20% implies the profiles are very similar. (For comparison, profiles in Figure 5 exhibit PBE(C) = 23.2% on average). As the average PBE bar in Figure 6 shows, 19% of the profiles exhibit PBE(C) $\leq$ 1%, and 73% exhibit PBE(C) $\leq$ 20%.

The remaining 27% of CRD profile comparisons belong to the "$>$ 20%" category. In most cases, these profiles have higher PBE(C) due to benign non-systematic errors. Such errors are illustrated by the shaded region in Figure 5: CRD profiles fluctuate above and below each other frequently, resulting in errors at individual CRD bins. But these errors do not accumulate, so the profiles are still very similar.

To factor out non-systematic errors, we compare the *sum of reference counts* (*i.e.*, area) across windows of W bins. We call this the *profile window error for capacity C over window W, PWE(C,W).*

PWE(C,W) is computed using Equation 1 except the $CRD_C[i]$ and $CRD_{32}[i]$ terms are replaced by their summing versions, $\sum_{j=i}^{i+W} CRD_C[j]$ and $\sum_{j=i}^{i+W} CRD_{32}[j]$, respectively. (To handle summing past *maxbin*, we assume $CRD_C[j] = 0$ for $j > maxbin$). The outer sum in Equation 1 doesn't change, so we still perform *maxbin* comparisons–*i.e.*, the summing window slides across all CRD bins. PWE(C,W) measures by how much the W-bin area under a CRD profile changes when the LLC varies from 32MB to capacity C. In the limit, when $W = maxbin$, PWE(C,W) becomes the cache-miss count error.

In Figure 6, the bars labeled "PWE" report the percentage of profile comparisons with $\leq 1\%$, 1–10%, 10–20%, and $> 20\%$ PWE(C,W) for $W = 10$, 20, 30, and *maxbin*. The rightmost set of "PWE" bars report the average across all benchmarks. From our experience, a PWE(C,W) of 1–20% implies the profiles are very similar. (For comparison, the 8MB LLC profile in Figure 5 exhibits PWE(8,20) = 22.7%). As the average PWE bars in Figure 6 show, 90% of the profiles exhibit PWE(C,20) under 20%, and 99% of the profiles exhibit PWE(C,*maxbin*) under 20%. These results demonstrate the vast majority of CRD profiles exhibit low sensitivity to LLC capacity scaling.

3.3 Core Count Scaling

Unlike LLC capacity scaling, CRD profiles change significantly under core count scaling. Figure 7 shows an example. In Figure 7, we plot CRD profiles from the Barnes benchmark running on a 32MB LLC using the S4 problem size, but vary the machine size to use 1, 4, 16, 64, and 256 cores (*i.e.*, profiles at $APS_{x,S4,32}$). The graph uses the same format as Figure 5, with an inset magnifying the region below CRD = 8MB.

Figure 7 shows CRD profiles *shift to larger CRD values with increasing core count* (especially visible in the inset). But the shifting is non-uniform: initially it is significant, then it slows down, and eventually it stops. CRD profile shift is due to increasing memory reference interleaving.

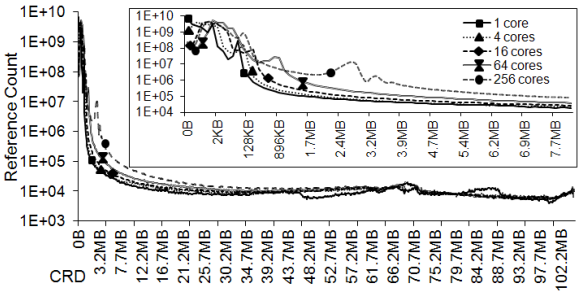

Figure 7. CRD profiles for Barnes on 4, 16, 64, and 256 cores at $APS_{x,S4,32}$.

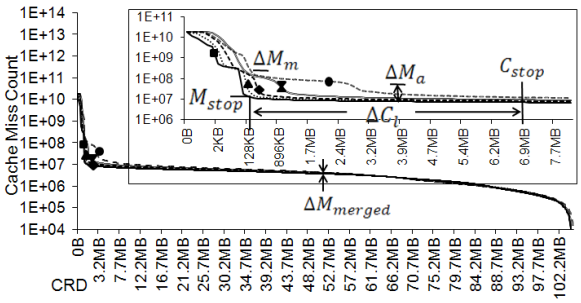

Figure 8. CMC profiles for each CRD profile in Figure 7 with C_{stop}, ΔC_l, and ΔM labeled.

Interleaving tends to dilate intra-thread reuse distances (see Section 2). As core count increases, so does interleaving, causing memory references to move to larger CRD values.[3] In addition to core count, profile shifting also tends to increase with CRD value as well (up to C_{stop}). This is due to the fact that larger intra-thread reuse distances are more likely to experience inter-thread interleaving compared to smaller reuse distances, and hence, experience greater dilation. Most of our benchmarks exhibit two shifting regions: one that shifts a little followed by another that shifts a lot. These small- and large-shift regions are visible in Figure 7's inset.

The reason CRD profiles eventually stop shifting is because memory interleaving is a localized phenomenon, occurring only within parallelized loops. This limits the maximum reuse distance dilation. More distant reuses occurring across loops do not experience increased interleaving. Instead, because parallelization usually does not change per-loop work much, distant inter-loop reuses tend to maintain their RD values.

The impact of core count scaling on CRD profiles has implications for cache performance. To see this, we compare cache-miss counts derived from the shifted profiles. As discussed in Section 3.2, the number of cache misses incurred at capacity i in profile CRD_C can be written as $\sum_{j=i}^{maxbin} CRD_C[j]$. Figure 8 plots the *cache-miss count (CMC) profile*, *i.e.* the number of cache misses at every cache capacity up to *maxbin*. Five CMC profiles are plotted, one for each of the CRD profiles in Figure 7

[3]Interleaving can also reduce reuse distance if threads share data, but dilation is more prevalent as we find CRD profiles almost always shift to the right whenever core count increases.

(an inset magnifies the < 8MB CRD region). Comparing the curves in Figure 8, we see CMC profiles exhibit similar shifting as CRD profiles. More importantly, this profile shift causes an increase in cache misses with core count scaling. The increase is large below C_{stop}, and very small (or non-existent) above C_{stop}. In other words, *core count scaling degrades locality, but the impact is confined to smaller CRD values*. This implies LLCs smaller than C_{stop} will incur a significant cache miss increase with core count scaling, but LLCs larger than C_{stop} will not.

Most of our benchmarks exhibit the behaviors shown in Figure 8, though the exact locality impact differs. To compare benchmarks' response to core count scaling, we use three parameters. The first is C_{stop}. The other two are ΔC_l, the capacity spanned by the large-shift region, and ΔM, the miss-count increase due to shifting. These parameters are labeled in Figure 8, and measured as follows. For every LLC and problem size in APS, we derive the CMC profiles for 1–256 cores (*i.e.*, the profiles at the same Y-Z coordinate in APS, as in Figure 8), and define ΔM to be the ratio of cache-miss counts between the 256- and 1-core CMC profiles at a particular CRD value. We consider the ΔM at a CRD well beyond C_{stop} (we use CRD = $\frac{maxbin}{2}$) where the CMC profiles have practically merged and ΔM is close to one. We call this ΔM_{merged}. Then, we identify the CRD closest to $\frac{maxbin}{2}$ where $\Delta M = 1.5 \times \Delta M_{merged}$, *i.e.* the tail-end of the large-shift region where very large ΔM transition to ΔM_{merge}. This CRD value is C_{stop}. Lastly, we identify the number of cache misses at C_{stop}, called M_{stop} (see Figure 8), and determine the CRD values where the 1- and 256-core CMC profiles intercept M_{stop}. The difference between these two CRD values is ΔC_l.

Table 3 reports C_{stop}, ΔC_l, and ΔM for each benchmark and problem size averaged across different LLC capacities. For ΔM, we report both the maximum and the average ΔM across the large-shift region, ΔM_m and ΔM_a, respectively. The lower-right corner of Table 3 reports averages across all benchmarks. Some cases could not be analyzed. Blackscholes and KMeans exhibit extremely small working sets that fit in the L1 cache. For these benchmarks, there is no

Benchmark	C_{stop}	ΔC_l	ΔM_m / ΔM_a	$maxbin$	Benchmark	C_{stop}	ΔC_l	ΔM_m / ΔM_a	$maxbin$
FFT S1	781.4KB	761.4KB	19.4 / 4.7	4.3MB	Ocean S1	707.0KB	494.5KB	13.0 / 8.8	6.5MB
S2	1.9MB	1.9MB	18.3 / 4.6	14.3MB	S2	1.5MB	500.1KB	4.0 / 3.5	19.0MB
S3	4.7MB	4.7MB	13.7 / 4.2	52.3MB	S3	4.7MB	615.4KB	2.2 / 2.1	64.4MB
S4	10.8MB	10.8MB	7.8 / 3.8	200.3MB	S4	17.2MB	1.2MB	1.9 / 1.8	238.0MB
LU S1	135.7KB	108.2KB	48.1 / 11.9	685.1kB	Water S1	228.5KB	222.4KB	70.3 / 34.1	1.3MB
S2	194.3KB	95.3KB	3.3 / 2.3	2.3MB	S2	655.6KB	531.6KB	16.9 / 5.7	3.4MB
S3	309.1KB	98.1KB	2.5 / 1.7	8.3MB	S3	1.7MB	1.2MB	4.4 / 2.6	11.6MB
S4	314.3KB	167.0KB	42.6 / 17.2	32.4MB	S4	3.4MB	1.4MB	2.3 / 1.8	45.5MB
Radix S1	-	-	-	30.3MB	KMeans S1	-	-	-	5.3MB
S2	-	-	-	36.3MB	S2	-	-	-	19.5MB
S3	7.7MB	7.7MB	10.8 / 5.4	60.3MB	S3	-	-	-	76.5MB
S4	22.0MB	21.9MB	6.9 / 3.3	156.45MB	S4	-	-	-	304.5MB
Barnes S1	465.5KB	427.0KB	86.2 / 21.2	2.1MB	Black- S1	-	-	-	1.6MB
S2	1.9MB	1.8MB	24.1 / 6.8	7.0MB	Scholes S2	-	-	-	6.1MB
S3	4.2MB	4.0MB	14.2 / 5.3	26.5MB	S3	-	-	-	24.1MB
S4	6.9MB	6.7MB	14.7 / 4.8	105.4MB	S4	-	-	-	96.1MB
FMM S1	649.9KB	618.0KB	29.4 / 9.4	4.1MB	Average S1	494.7KB	438.6KB	44.4 / 15.0	6.2MB
S2	1.4MB	1.4MB	32.4 / 6.5	12.4MB	S2	1.3MB	1.0MB	16.5 / 4.9	13.3MB
S3	7.2MB	7.2MB	37.6 / 3.8	50.1MB	S3	4.3MB	3.6MB	12.2 / 3.6	41.5MB
S4	9.2MB	9.1MB	34.5 / 3.0	163.1MB	S4	10.0MB	7.3MB	15.8 / 5.1	149.0MB

Table 3. C_{stop}, ΔC_l, ΔM_m, and ΔM_a at different problem sizes for our benchmarks.

detectable large-shift region. Radix employs a large amount of per-core private data that dominates shared data for the S1 and S2 problems, causing *maxbin* to increase significantly with core count in these cases. The *maxbin* variation is so great that $\frac{maxbin}{2}$ is not well defined across different core counts, again preventing our detection of a large-shift region.

For the remaining benchmarks and problem sizes in Table 3, C_{stop} varies between 136KB and 21.9MB. As the average bars show, C_{stop} is between 400KB and 10MB across different problem sizes. These results demonstrate the locality degradation due to core count scaling in our benchmarks and problem sizes only impacts smaller LLCs ($\leq$ 16MB). The larger LLCs in our study are all beyond C_{stop}, and hence, are largely insensitive to core count scaling's locality impact. (We note, however, C_{stop} increases with problem size in Table 3. Section 3.4 will discuss this further.)

Table 3 also shows ΔC_l is between 95KB and 21.9MB. On average, ΔC_l is 80% of C_{stop}, so the shifting region below C_{stop} is spanned almost entirely by the large-shift region. We find the small-shift region is very small, usually spanning the first 20KB in CMC profiles, so it never impacts LLC performance.

Finally, the average bars in Table 3 show ΔM_m varies between 12.2 and 44.4 while ΔM_a varies

between 3.6 and 15.0. For LLC sizes that fall within the large-shift region, ΔC_l, our results show core count scaling can increase cache misses significantly. For example, consider the fact that scaling core count by 256x creates a 2 orders of magnitude increase in off-chip bandwidth due to parallelism (assuming linear bandwidth increase with core count for symmetric threads). Table 3 shows the same 256x increase in cores can potentially add another order of magnitude in cache misses (and hence, total memory traffic) due to locality degradation.

3.4 Problem Scaling Impact

We now briefly describe problem scaling's impact on CRD profiles (see [1] for a more detailed treatment). Figure 9 shows an example of CRD profile variation across problem scaling. It plots CRD profiles from the Barnes benchmark running on 16 cores and a 32MB LLC, but varies problem size from S1–S4 (*i.e.*, profiles at $APS_{16,y,32}$). The profiles are presented in the same format as Figures 7 and 8.

Interestingly, problem scaling has a similar effect on CRD profiles as core count scaling. As Figure 9 shows, problem scaling also shifts CRD profiles to larger CRD values. In problem scaling, this shifting is due to additional memory references that arise from accessing scaled data structures (as opposed to memory interleaving in core count scaling). But problem scaling shifts larger CRD values, with no shifting at small CRD–the exact opposite compared to core count scaling. The reason shifting stops below a certain CRD is because frequently referenced objects that account for small CRD (*e.g.*, temporary variables) often do not scale with respect to input problem size. These observations are consistent with Zhong's results [1].

As with core count scaling, the shifting caused by problem scaling also degrades locality and increases cache misses. While the two forms of scaling shift complementary portions of the CRD profile, we find the shifting regions overlap. In particular, the large-shift region in core count scaling

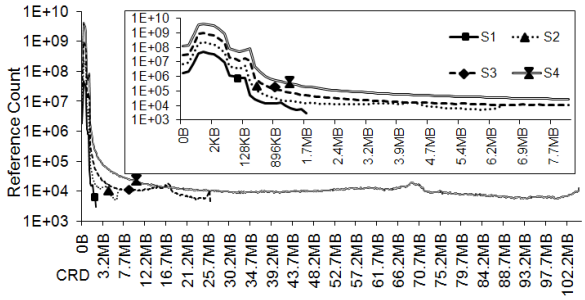

Figure 9. CRD profiles for Barnes using S1–S4 on 16 cores and a 32MB LLC.

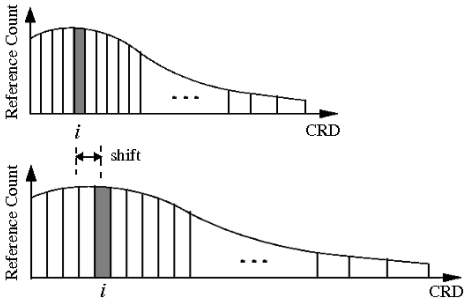

Figure 10. Detecting alignment and shifting using reference groups.

usually shifts with problem scaling. So, as problem size increases, core count scaling will tend to affect a wider range of LLCs.

To illustrate this, we study C_{stop} sensitivity to problem scaling. In Table 3, the columns labeled "*maxbin*" report the maximum CRD value for different benchmarks and problem sizes. Each *maxbin* value is averaged across different core counts and LLC sizes. As Table 3 shows, *maxbin* increases by roughly 4x with each problem size increment–*i.e.*, linearly with problem size. In contrast, C_{stop} shifts at a slower rate because it occurs at smaller CRD where sensitivity to problem scaling is less. Nevertheless, C_{stop} still shifts roughly as the square root of problem size. Extrapolating to larger problems, we see that another 64x increase in problem size will cause C_{stop} to grow to 128MB. For such larger problems (which are not unreasonably large for LCMPs), core count scaling would impact the entire range of LLC capacities in our study.

4 Profile Prediction

Zhong *et al* [1] shows that profile shifting caused by problem scaling (which core count scaling resembles) is predictable. In fact, this previous work has already proposed techniques to predict profiles of scaled configurations from profiles acquired on smaller configurations. In this section, we review these techniques, apply them to predict core count scaling, and evaluate the prediction accuracy.

4.1 Reference Groups

Zhong's technique samples two profiles from different problem sizes to determine the amount of shift as a function of the problem size increase. Then, they apply a scaled shift to predict profiles for larger problem sizes. Zhong's work assumes sequential programs, so they predict RD profiles. However, we find their approach can predict CRD profiles for parallel programs as well.

A key issue addressed by Zhong's technique is the non-uniformity of profile shift. As discussed in Section 3, core count scaling shifts small CRD values more than large CRD values, and vice versa for problem scaling. To handle variable shift across CRD values, Zhong's technique uses *reference groups*, illustrated in Figure 10. Zhong divides sampled CRD profiles into groups along the CRD axis, each containing an equal fraction of the program's total references. Reference groups are *aligned* via association: the i^{th} group in the first profile is aligned to the i^{th} group in the second profile (as Figure 10 shows).

Zhong's technique assumes aligned reference groups "correspond" to each other across the shift, and maintain a fixed shift rate dependence on problem scaling. This dependence can be at the least constant–*i.e.*, no shift with problem size–and at most linear shift with problem size.[4] In addition to constant and linear, Zhong's technique also allows intermediate shift rates: cube root, square root, and cube-root squared. The amount of shift between pairs of reference groups is measured, and compared against each allowed shift rate. The one with the closest match is assigned to the reference group. To predict CRD profiles for larger problems, each reference group is shifted according to its shift rate and the desired problem scaling factor.

We apply Zhong's technique to predict core count scaling as follows. For a given LLC and problem size, we use the 2- and 4-core CRD profiles as samples to predict the CRD profiles at the remaining core counts, 8 to 256–*i.e.*, we predict along the X-axis in $APS_{x,y,z}$. (For core count

[4]Group shift rate cannot be greater than linear because reuse distance cannot increase by more than the number of unique memory locations in memory, which is proportional to problem size.

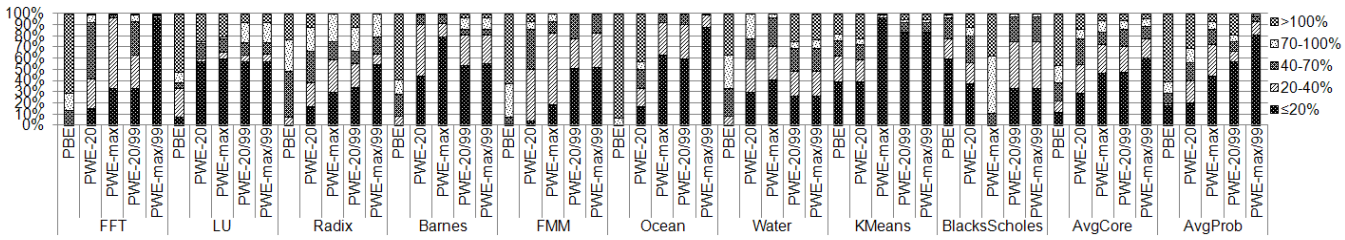

Figure 11. Percentage of CRD profiles predicted across core count scaling whose PBE and PWE with window size 10, 20, 30, and $maxbin$ are $\leq$ 20%, 20-40%, 40-70%, 70-100%, and $>$ 100%.

prediction, we do not consider the 1-core CRD profiles). We detect the inter-group shift as discussed above, but instead of multiplying this shift rate by the problem scaling factor, we multiply by the core count scaling factor. Lastly, while Zhong originally divided each profile into 1,000 reference groups, we use 100,000 reference groups. We find the increased resolution provides slightly better accuracy for core count scaling.

4.2 Prediction Accuracy

This section evaluates the accuracy of predicting CRD profiles when each scaling dimension–either core count or problem size–is predicted separately. (Later, in Section 5, we will predict both core count and problem scaling together). Our study considers core count scaling first. We repeat prediction along the core count scaling dimension for every LLC capacity and problem size in APS. Then, we compare each predicted CRD profile at $APS_{x,y,z}$ against the measured CRD profile at the same $APS_{x,y,z}$ using the PBE and PWE metrics from Section 3.2. (Note, this is a different use of PBE/PWE compared to Section 3.2 which employed the metrics to compare measured CRD profiles at *different* $APS_{x,y,z}$).

Figure 11 reports prediction error. In Figure 11, groups of bars break down for each benchmark the number of predicted CRD profiles that exhibit $\leq$ 20%, 20–40%, 40–70%, 70–100%, and $>$ 100% error compared to their corresponding measured profiles. The "PBE" bars report breakdowns under

the PBE metric while the "PWE-20" and "PWE-max" bars report breakdowns under the PWE metric for window sizes 20 and *maxbin*, respectively. The group of bars labeled "AvgCore" report averages across all benchmarks.

The prediction errors are significant. As the AvgCore PBE bars show, only 12% of predicted CRD profiles exhibit $\leq$ 20% PBE. The other predicted profiles exhibit larger error: 22%, 38%, and 53% of the predictions exhibit up to 40%, 70%, and 100% PBE, respectively, with the remaining predictions exhibiting $>$ 100% PBE. Similar to Figure 6, the PWE bars report lower error, indicating some of the PBE cancels across windows. As the AvgCore PWE-20 bars show, 29%, 54%, 77%, and 86% of predicted profiles exhibit up to 20%, 40%, 70%, and 100% PWE, respectively. For PWE-max, the predicted profiles that fall within the same error ranges increases to 46%, 73%, 83%, and 93%, respectively. But still, even PWE is significant.

Although predicted CRD profiles exhibit elevated PBE/PWE, we find a significant part of the error is from large CRD values where the impact on RD analysis is minor. At large CRD values, reference counts can be very small (10,000 or less) which tends to magnify PBE/PWE. At the same time, because these reference counts are small, they never have a large impact on cache performance analyses (*e.g.*, CMC profiles).

To illustrate this, the PWE-20/99 and PWE-max/99 bars in Figure 11 report PWE averaged across the leftmost CRD bins in each CRD profile accounting for 99% of the total memory references with window size 20 and *maxbin*, respectively (the CRD bins accounting for the remaining 1% of references are excluded from the average). PWE reduces noticeably when excluding large CRD values. As the AvgCore PWE-20/99 bars in Figure 11 show, 71% and 84% of predicted profiles exhibit $\leq$ 40% and $\leq$ 70% PWE, respectively, with window size 20. With window size *maxbin*, 77% and 88% of predicted profiles exhibit $\leq$ 40% and $\leq$ 70% PWE, respectively. Our results show reference groups cannot predict core count scaling perfectly, but a large number of profiles are

predicted with good accuracy.

In addition to evaluating core count prediction, we also evaluate predicting problem scaling (the original use of reference groups). For a given core count and LLC capacity, we use the S1 and S2 CRD profiles as samples to predict the CRD profiles at the S3 and S4 problem sizes–*i.e.*, we predict along the Y-axis in $APS_{x,y,z}$. We repeat prediction along the problem scaling dimension for every core count and LLC size, and compare each predicted CRD profile against its corresponding measured CRD profile.

To conserve space, we only report the average results across all benchmarks, appearing in the last group of bars labeled "AvgProb" in Figure 11. As the AvgProb PBE bars show, 17%, 19%, 30%, and 40% of predicted profiles exhibit up to 20%, 40%, 70%, and 100% PBE, respectively, with the remaining predictions exceeding 100% PBE. Errors become smaller under the PWE metric. As the AvgProb PWE-20 bars show, 20%, 40%, 56%, and 69% of predicted profiles exhibit up to 20%, 40%, 70%, and 100% PWE, respectively. For PWE-max, the predicted profiles that fall within the same error ranges increases to 44%, 73%, 86%, and 93%, respectively. Finally, when excluding small CRD values, the errors reduce even further. As the AvgProb PWE-20/99 bars show, 66% and 74% of predicted profiles exhibit $\leq$ 40% and $\leq$ 70% PWE, respectively, for window size 20. With window size *maxbin*, 92% and 98% of predicted profiles exhibit $\leq$ 40% and $\leq$ 70% PWE, respectively.

The prediction errors for the "AvgProb" and "AvgCore" bars in Figure 11 are comparable. Based on these results, we find reference groups are about as effective for predicting core count scaling as they are for predicting problem scaling.

5 Performance Prediction

Having characterized and predicted CRD profiles for parallel programs, we now demonstrate their utility. This section uses the CRD profiles from Sections 3 and 4 to predict cache performance

in our tiled CMP. We first discuss methodology, and then present results.

5.1 Prediction Methodology

When acquiring CRD profiles, our simulations also record cache performance, in particular L2 MPKI, at every $APS_{x,y,z}$. This section uses CRD profiles to predict the measured cache performance. Specifically, we use the CRD profiles acquired at $APS_{x,y,32}$ (*i.e.*, the dotted plane in Figure 3) to predict performance for the same core count and problem size (*i.e.*, same X-Y coordinate). Prediction across LLC capacity is done by examining the capacities of interest (4–128MB) along the CRD axis in the profile at $APS_{x,y,32}$.

To predict L2 MPKI at a particular $APS_{x,y,C}$, we derive the CMC profile from the CRD profile at $APS_{x,y,32}$, and extract the cache-miss count at the desired capacity, C. This predicts the number of misses assuming a fully-associative cache (*i.e.*, capacity misses). Since our tiled CMP employs set-associative caches, we use Qasem and Kennedy's model [24] to account for cache conflicts. This model takes our CRD profiles as input, and uses a binomial distribution to estimate conflict misses for a given LLC capacity and associativity. We divide the sum of predicted conflict and capacity misses by instruction count (IC) to derive MPKI. We assume knowledge of IC at every $APS_{x,y,32}$, and further assume IC is constant across LLC capacity.

In addition to predicting MPKI using measured CRD profiles, we also do the same with predicted CRD profiles. We consider predicting across core count alone as well as across core count and problem size together. In the former, we use the profiles at $APS_{2,y,32}$ and $APS_{4,y,32}$ to predict the remaining core counts at $APS_{x,y,32}$. In the latter, we use the profiles at $APS_{2,S1,32}$, $APS_{2,S2,32}$, $APS_{4,S1,32}$, and $APS_{4,S2,32}$ to predict the profiles at $APS_{2,S3,32}$, $APS_{2,S4,32}$, $APS_{4,S3,32}$, and $APS_{4,S4,32}$ across the problem size dimension. Then, we use these predicted profiles to predict across core count as already described. We also predict IC. We assume knowledge of IC at $APS_{1,S1,32}$ and

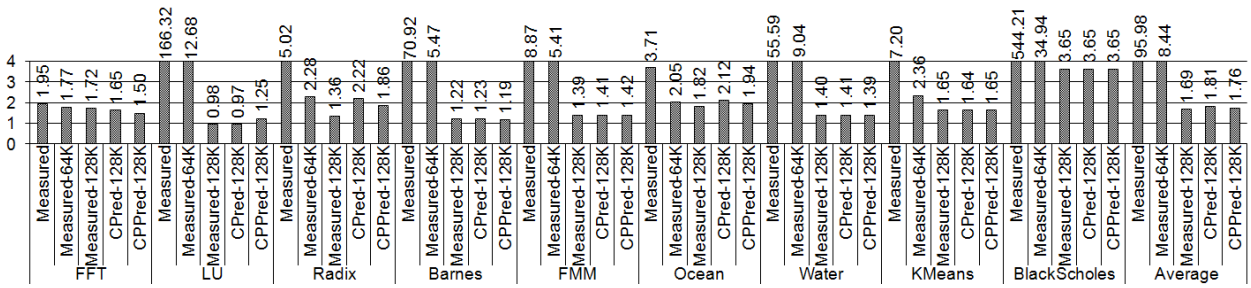

Figure 12. Ratio of measured-to-predicted L2 MPKI.

$APS_{1,S2,32}$, and derive all other ICs by assuming IC doesn't change across core count, but changes linearly with problem size at the same rate observed from S1 to S2.

Finally, we predict L2 MPKI for the rectangular sub-space in APS excluding configurations used for CRD profile prediction (since we simulate many of these configurations to acquire their CRD profiles anyways). We exclude all configurations with 1, 2, and 4 cores and the S1 and S2 problems, resulting in a sub-space with 72 configurations instead of 216. Also, for a particular CRD profile, we do not predict L2 MPKI at capacities beyond $maxbin$. For such large LLCs, cold misses dominate which our profiles cannot predict.

5.2 Performance Prediction Accuracy

Figure 12 reports the accuracy of L2 MPKI prediction. The Y-axis in Figure 12 plots accuracy as the ratio of simulated to predicted L2 MPKI (*i.e.*, 1.0 implies perfect prediction). When actual L2 MPKI is near zero, this ratio blows up if predicted MPKI approaches zero. To address this, we add a small MPKI value, 0.1, to both the simulated and predicted values before taking the ratio. For each benchmark, Figure 12 reports 5 bars which we explain below. The rightmost group of bars reports averages across all 9 benchmarks.

The bars labeled "Measured" in Figure 12 show accuracy when performance prediction is driven by the measured CRD profiles in the APS sub-space. These Measured bars are always above 1.0, indicating CRD profiles under-predict actual performance. More problematic, the under-prediction

is huge, between 1.9–9.0x for 5 benchmarks, and between 55.6–544.2x for the other 4 benchmarks.

These huge errors are due to configurations with large core counts and small LLCs. In such processors, LLC thrashing often occurs due to pathologic cache conflicts, which the simple conflict model cannot predict. To show the impact of these cases, the bars labeled "Measured-64K" in Figure 12 report accuracy without configurations using $\leq$ 64KB L2 slices (*i.e.*, 256 cores with $\leq$ 16MB LLCs, 128 cores with $\leq$ 8MB LLCs, and 64 cores with 4MB LLCs), and the bars labeled "Measured-128K" report the same without configurations using $\leq$ 128KB L2 slices. The former removes 22% of configurations in the *APS* sub-space, while the latter removes 36% of configurations. As the average bars show, without sub-64KB slices, the under-prediction reduces from 96.0 to 8.4, and without sub-128KB slices, the under-prediction further reduces to 1.7.

The bars labeled "CPred-128K" and "CPPred-128K" in Figure 12 show accuracy when performance prediction is driven by the predicted CRD profiles. CPred-128K uses prediction across core count scaling alone while CPPred-128K uses prediction across both core count and problem scaling. These bars all exclude sub-128K L2 slices. As the average bars in Figure 12 show, predicted CRD profiles add more error: under-prediction increases to 1.8 for both CPred-128K and CPPred-128K. But the additional performance prediction error is only 12%, which is less than the profile prediction error reported in Section 4.2. This is because the accuracy of performance prediction depends on both CRD profiles and the cache conflict model, which tends to reduce the impact of profile errors.

Although CRD profiles cannot predict the simulated cache performance exactly, they yield predictions in a timely fashion. The Measured, CPred-128K, and CPPred-128K bars in Figure 12 only require $\frac{1}{6}^{th}$, $\frac{1}{8}^{th}$, and $\frac{1}{8}^{th}$ of the simulations, respectively, from the *APS* sub-space to predict the entire sub-space. The savings in simulation time is even greater because the omitted simulations are for the larger core counts and problem sizes. Roughly $\frac{1}{4}^{th}$, $\frac{1}{11}^{th}$, and $\frac{1}{1200}^{th}$ of the full simulation

time, respectively, is required. These results confirm RD analysis enables efficient exploration of multicore processor design spaces.

6 Related Work

Several researchers have recently developed RD analysis techniques for multicore processors [25, 16, 26]. Ding and Chilimbi [25] analyze multithreaded traces to extract statistics on per-thread locality, data sharing, and interleaving to drive a model for predicting locality and cache performance on scaled systems. Their approach is more general than ours, as it can handle non-symmetric threads. However, their trace analyses are very expensive, incurring space and time complexity quadratic and exponential, respectively, with the number of threads. So, they cannot study large design spaces. In contrast, we only acquire CRD profiles on small-scale multicore configurations (no trace acquisition or analysis). Predicting scaled CRD profiles from the sampled profiles is computationally trivial, enabling our technique to analyze LCMPs.

Jiang *et al* [16] propose a model for deriving CRD profiles from per-thread RD profiles. Jiang's model requires knowing all per-thread RD profiles a priori. It cannot explore multicore configurations that it has not yet profiled, limiting its use for scaling studies (the focus of our work). One key point they make is that for the special case of parallel programs, changes in core architecture do not change the relative execution speed of threads, allowing CRD profiles to remain the same across different multicore architectures. Our finding that CRD profiles exhibit low sensitivity to LLC capacity scaling is a very similar observation.

Schuff *et al* [26] also investigate using CRD profiles to analyze shared caches. In addition, they propose using per-thread RD profiles to analyze private caches, taking into consideration write-sharing in order to predict invalidations. However, their work only applies RD analysis to LLC capacity scaling. In contrast, we look at a much larger design space that includes core count and problem scaling by predicting the impact these forms of scaling have on CRD profiles.

Chandra *et al* [27] and Suh *et al* [28] have also developed locality models for multicore processors, but they focus on multiprogrammed workloads whereas our work focuses on parallel programs. RD analysis has also been used to analyze uniprocessor caches [13, 14, 15, 1]. Of these, our work is most similar to Zhong *et al* [1]. Section 4.1 has already discussed Zhong's approach in detail.

Finally, our work is related to all previous research on multicore design space exploration [2, 3, 4, 5, 6, 7, 8, 9]. Like them, we characterize the impact of processor scaling on memory system performance. However, previous studies use simulation. To our knowledge, we are the first to use RD analysis.

7 Conclusion

This paper tackles the architecture dependence of CRD profiles, showing the problem is tractable for parallel programs. In particular, we show CRD profiles change very little with LLC capacity scaling, so profiles acquired at one capacity can be used to accurately analyze different LLC sizes. We also show core count scaling shifts CRD profiles to larger CRD values. Shifting occurs below a certain CRD value, C_{stop}, so locality degradation only impacts LLC capacities $< C_{stop}$. Furthermore, we show shifting is predictable via profile "diffing." This means a small set of CRD profiles can (after prediction) analyze a large number of machine sizes. To demonstrate benefits, we use CRD profiles to predict LLC performance across a large tiled CMP design space. We find CRD profiles can predict LLC MPKI within 76% of simulation for configurations without pathologic cache conflicts in $\frac{1}{1200}^{th}$ the time to perform simulation of the full design space.

References

[1] Y. Zhong, S. G. Dropsho, and C. Ding, "Miss Rate Prediction across All Program Inputs," in *Proc. of PACT*, 2003.

[2] J. Davis, J. Laudon, and K. Olukotun, "Maximizing CMP Throughput with Mediocre Cores," in *Proc. of PACT*, 2006.

[3] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell, "Exploring the Cache Design Space for Large Scale CMPs," *SIGARCH Comp. Arch. News*, vol. 33, 2005.

[4] J. Huh, S. W. Keckler, and D. Burger, "Exploring the Design Space of Future CMPs," in *Proc. of PACT*, 2001.

[5] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proc. of the Int'l Symp. on Microarchitecture*, 2009.

[6] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP Design Space Exploration Subject to Physical Constraints," in *Proc. of HPCA*, 2006.

[7] J. Li and J. F. Martinez, "Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors," in *Proc. of ISPASS*, 2005.

[8] B. Rogers, A. Krishna, G. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling," in *Proc. of the Int'l Symp. on Comp. Arch.*, 2009.

[9] L. Zhao, R. Iyer, S. Makineni, J. Moses, R. Illikkal, and D. Newell, "Performance, Area and Bandwidth Implications on Large-Scale CMP Cache Design," in *Proc. of the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnect*, 2007.

[10] A. Agarwal, L. Bao, J. Brown, B. Edwards, M. Mattina, C.-C. Miao, C. Ramey, and D. Wentzlaff, "Tile Processor: Embedded Multicore for Networking and Multimedia," in *Proc. of the 19th Symp. on High Performance Chips*, 2007.

[11] Y. Hoskote, S. Vangal, N. Borkar, and S. Borkar, "Teraflop Prototype Processor with 80 Cores," in *Proc. of the Symp. on High Performance Chips*, 2007.

[12] "Silicon Industry Association Technology Roadmap." 2009.

[13] 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*, pp. 617–662, 2001.

[14] C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," *SIGPLAN Not.*, vol. 38, no. 5, pp. 245–257, 2003.

[15] Y. Zhong, X. Shen, and C. Ding, "Program locality analysis using reuse distance," *ACM Trans. Program. Lang. Syst.*, vol. 31, no. 6, pp. 1–39, 2009.

[16] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen, "Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors?." 2010.

[17] J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, 1970.

[18] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proc. of ISCA*, 2005.

[19] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches," in *Proc. of the Int'l Symp. on Comp. Arch.*, 2009.

[20] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The M5 Simulator: Modeling Networked Systems," vol. 26, no. 4, 2006.

[21] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the 22nd International Symposium on Computer Architecture*, (Santa Margherita Ligure, Italy), June 1995.

[22] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "MineBench: A Benchmark Suite for Data Mining Workloads," in *Proceedings of the International Symposium on Workload Characterization*, October 2006.

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proc. of PACT*, 2008.

[24] A. Qasem and K. Kennedy, "Evaluating a model for cache conflict miss prediction," tech. rep., Rice University, 2005.

[25] C. Ding and T. Chilimbi, "A Composable Model for Analyzing Locality of Multi-threaded Programs," MSR-TR 2009-107, Microsoft Research, 2009.

[26] D. L. Schuff, B. S. Parsons, and J. S. Pai, "Multicore-Aware Reuse Distance Analysis," TR 09-07, Purdue University School of Electrical and Computer Engineering, August 2009.

[27] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture," in *Proceedings of the International Symposium on High Performance Computer Architecture*, February 2005.

[28] G. E. Suh, S. Devadas, and L. Rudolph, "Analytical Cache Models with Applications to Cache Partitioning," in *Proceedings of ICS*, 2001.