Identifying Optimal Multicore Cache Hierarchies for Loop-based Parallel Programs via Reuse Distance Analysis

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

ABSTRACT

Understanding multicore memory behavior is crucial, but can be challenging due to the complex cache hierarchies employed in modern CPUs. In today's hierarchies, performance is determined by complicated thread interactions, such as interference in shared caches and replication and communication in private caches. Researchers normally perform extensive simulations to study these interactions, but this can be costly and not very insightful. An alternative is multicore reuse distance (RD) analysis, which can provide extremely rich information about multicore memory behavior. In this paper, we apply multicore RD analysis to better understand cache system design. We focus on loop-based parallel programs, an important class of programs for which RD analysis provides high accuracy. We propose a novel framework to identify optimal multicore cache hierarchies, and extract several new insights. We also characterize how the optimal cache hierarchies vary with core count and problem size.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies, Modeling techniques; B.3.2 [**Design Styles**]: Cache memories; C.1.4 [**Parallel Architectures**]: Distributed architectures

General Terms

Design, Performance

Keywords

Reuse Distance, Chip Multiprocessors, Cache Performance

1. INTRODUCTION

Memory performance is a major determiner of overall system performance and power consumption in multicore processors. For this reason, understanding how applications utilize the on-chip cache hierarchies of multicore processors is extremely important for architects, programmers, and compiler designers alike. But gaining deep insights into multicore memory behavior can be difficult due to the complex cache organizations employed in modern multicore CPUs.

Today, a typical multicore processor integrates multiple levels of cache on-chip with varying capacity, block size, and/or set associativity across levels. In addition, cores may physically share cache. Each core normally has its own dedicated L1 cache, but the caches further down the hierarchy can be either private or shared. Moreover, data replication is usually permitted across the private caches, with hardware cache coherence used to track and manage replicas.

In such cache hierarchies, memory performance depends not only on how well per-thread working sets fit in the caches, but also on how threads' memory references interact. Many complex thread interactions can occur. For instance, inter-thread memory reference interleaving leads to *interference* in shared caches. Also, *data sharing* may reduce aggregate working set sizes in shared caches, partially offsetting the performance-degrading interference effects. But in private caches, data sharing causes *replication*, increasing capacity pressure, as well as *communication*.

To study these complex effects, researchers normally rely on architectural simulation [3, 5, 6, 9, 10, 21], but this can be very costly given the large number of simulations required to explore on-chip cache design spaces. A more efficient approach that can potentially yield deeper insights is *reuse distance (RD) analysis*. RD analysis measures a program's memory reuse distance histogram–*i.e.*, its locality profile–directly quantifying the application-level locality responsible for cache performance. Hence, a single locality profile can predict the cache-miss count at every possible cache capacity without having to simulate all of the configurations.

Multicore RD analysis is relatively new, but is becoming a viable tool due to recent work. In particular, researchers have developed new notions of reuse distance that account for thread interactions. *Concurrent reuse distance* (CRD) [4, 7, 12, 16, 17, 19, 20] measures RD across thread-interleaved memory reference streams, and accounts for interference and data sharing between threads accessing shared caches. On the other hand, *Private-stack reuse distance* (PRD) [12, 16, 17] measures RD separately for individual threads using per-thread coherent stacks. PRD accounts for replication and communication between threads accessing private caches.

One problem with multicore RD is that it is sensitive to memory interleaving, and hence, *architecture dependent*. However, such profile instability is minimal when threads

To Appear in *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness (MSPC-2012). Beijing, China. June 2012.*

exhibit similar access patterns [7, 17, 19]. For example, programs exploiting loop-level parallelism contain *symmetric threads* that tend to execute at the same rate regardless of architectural assumptions. For such programs, CRD and PRD profiles are essentially architecture independent and provide accurate analysis.

Despite this limitation, RD analysis can provide extremely rich information about multicore memory behavior. But surprisingly, there has been very little work on extracting insights from this information, as current research has focused primarily on developing techniques and verifying their accuracy. In this paper, we use RD analysis to extract new insights that can potentially help architects optimize multicore cache hierarchies. For accuracy, we focus on loop-based parallel programs. Though somewhat restrictive, our study is still fairly general given the pervasiveness of parallel loops. They dominate all data parallel codes. They also account for most programs written in OpenMP, one of the main parallel programming environments. And while our results come from parallel loops, we believe many of our insights apply to parallel programs in general.

In this work, we study two classic cache-hierarchy optimization problems, and we identify the configuration which can provide the lowest average memory access time (AMAT) for a three-level cache hierarchy. The first optimization problem is the selection between shared and private last level caches (LLCs). In the past, researchers rely on using detailed simulations to study the performance difference between shared and private caches at some cache sizes. In our analysis, we find that the selection between shared and private LLCs depends on two factors: (1) the degree of data sharing, which varies with reuse distance, and (2) the communication cost. Hence, the preference of shared/private LLC is a function of LLC capacity and the LLC access frequency. Because LLC access frequency plays an important role for cache performance, the L2 capacity is a very important design parameter. Hence, the second optimization problem that we study is the L2/LLC capacity partition when the total on-chip cache capacity is fixed. We identify the optimal L2/LLC capacity partition and the performance variation under the cache capacity constraint.

Our work makes several contributions. First, we propose a novel framework which employs multicore RD profiles to identify optimal multicore cache hierarchies for loop-based parallel programs. Our framework can analyze and quantify the performance differences for different cache hierarchies easily. Second, for a fixed on-chip cache capacity, we find that the optimal cache hierarchy exists when *the change of on-chip memory stall equals to the change of off-chip memory stall.* Hence, the key to optimize multicore cache hierarchies is *balancing the on-chip and off-chip memory stalls.* We find that the capacity of the last private cache above the last level cache must exceed the region in the PRD profile where significant data locality degradation happens. Third, shared LLC can outperform private LLC when *the total off-chip memory stall saved in shared LLC is larger than the total on-chip memory stall saved in private LLC.* At the optimal LLC size, the average performance difference between private LLCs and shared LLCs is only within 11%, which is smaller than the performance difference caused by the L2/LLC partitioning (76% in shared LLCs, and 30% in private LLCs). This suggests that physical data locality is very important for multicore cache design.

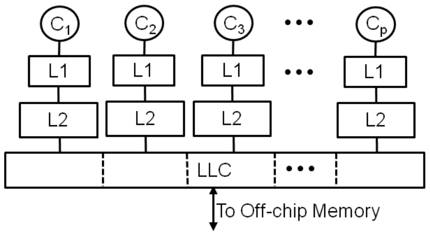


Figure 1: Multicore cache hierarchy.

Time:	1	2	3	4	5	6	7	8	9	10
Core C_1:	A	B	C		D		E			A
Core C_2:				F		C		G	H	

Figure 2: Two interleaved memory reference streams, illustrating different thread interactions.

The rest of this paper is organized as follows. Section 2 reviews CRD/PRD, and introduces our methodology and framework. Then, Section 3 analyzes private vs. shared cache performance. Section 4 studies the trade-offs between L2 and LLC capacities. Finally, Sections 5 and 6 discuss related work and conclusions.

2. MULTICORE RD ANALYSIS

Reuse distance is the number of unique memory references performed between two references to the same data block. An RD profile–*i.e.*, the histogram of RD values–can analyze uniprocessor cache performance. Because a cache of capacity C can satisfy references with RD $< C$ (assuming LRU), the cache-miss count (CMC) is the sum of all reference counts in an RD profile above the RD value for capacity C. In multicore CPUs, RD analysis must consider memory references from simultaneously executing threads. Below, we first review the multicore RD profiles. Then, we introduce our methodology and performance models.

2.1 CRD and PRD Profiles

Multicore processors often contain both shared and private caches. For example, Figure 1 illustrates a typical CPU consisting of 2 levels of private cache backed by a shared or private LLC. Threads interact very differently in each type of cache, requiring separate locality profiles. Multicore RD analysis also takes data sharing into account.

CRD profiles report locality across thread-interleaved memory reference streams, thus capturing interference in shared caches. CRD profiles can be measured by applying the interleaved stream on a single (global) LRU stack [7,12,16,17,19]. For example, Figure 2 illustrates a 2-thread interleaving in which core C_1 touches blocks A–E, and then re-references A, while core C_2 touches blocks C and F–H. Figure 3(a) shows the state of the global LRU stack when C_1 re-references A. While C_2's reference to C interleaves with C_1's reuse of A, this does not increase A's CRD because C_1 already references C in the reuse interval. So only three of C_2's references (F–H) are distinct from C_1's references between C_1's reuse of A. In this case, CRD is equal to 7, instead of 8. Hence, data sharing can reduce the impact of data interleaving in shared caches.

PRD profiles report locality across per-thread memory reference streams that access coherent private caches. PRD

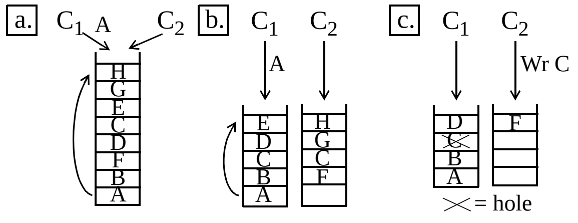


Figure 3: LRU stacks showing (a) interleave and overlap for CRD , (b) scaling, replication, and (c) holes for PRD.

profiles can be measured by applying threads' references on private LRU stacks that are kept coherent. Without writes, the private stacks do not interact. For example, Figure 3(b) shows the PRD stacks corresponding to Figure 3(a) assuming all references are reads. For C_1's reuse of A, PRD = 4. Note, however, the multiple private stacks still contribute to increased cache capacity. (Here, the capacity needed to satisfy PRD = 4 is 10, assuming 2 caches with 5 cache blocks each). To capture this effect, we compute the *scaled PRD* (sPRD) which equals $T \times PRD$, where T is the number of threads. Hence, for C_1's reuse of A, sPRD = 8.

In PRD profiles, read sharing causes *replication*, increasing overall capacity pressure. Figure 3(b) shows duplication of C in the private stacks. In contrast, write sharing causes *invalidation*. For example, if C_2's reference to C is a write instead of a read, then invalidation would occur in C_1's stack, as shown in Figure 3(c). To prevent invalidations from promoting blocks further down the LRU stack, invalidated blocks become *holes* rather than being removed from the stack [17]. Holes are unaffected by references to blocks above the hole, but a reference to a block below the hole moves the hole to where the referenced block was found.

Invalidations reduce locality for victimized data blocks since reuse of these blocks will always cause coherence misses. But invalidations can also improve locality because the holes they leave behind eventually absorb stack demotions. For example, in Figure 3(c), C_1's subsequent reference to E does not increase A's stack depth because D will move into the hole. Hence, C_1's reuse of A has PRD (sPRD) = 3 (6) instead of 4 (8).

2.2 Methodology

We use Intel PIN [11] to acquire whole-program locality profiles. Our PIN tool maintains a global LRU stack to compute a CRD profile, and coherent private LRU stacks to compute an sPRD profile. All stacks employ 64-byte memory blocks, the cache block size we assume. Our PIN tool follows McCurdy and Fischer's method [13], performing functional execution with context switching between threads every memory reference, which interleaves threads' references uniformly in time. While this does not account for timing, it is accurate for loop-based parallel programs [7, 17, 19].

Our study employs nine benchmarks, each running two problem sizes and eight core counts. Table 1 lists the benchmarks: FFT, LU, RADIX, Barnes, FMM, Ocean, and Water from SPLASH2 [18], KMeans from MineBench [14], and BlackScholes from PARSEC [2]. The second column of Table 1 specifies the problem sizes, S1 and S2, in terms of data elements. We acquire profiles in each benchmark's parallel region. The third column of Table 1 reports the number of instructions in the parallel phases studied. For FFT, LU, and RADIX, these regions are the entire parallel phase. For the other benchmarks, profiles are acquired for only one timestep of the parallel phase, so we skip the first timestep and profile the second timestep.

Table 1: Parallel benchmarks used in our study.

Benchmark	Problem Sizes(S1/S2)	Insts(M)(S1/S2)
FFT	$2^{20}/2^{22}$ elements	560/2,420
LU	$1024^2/2048^2$ elements	2,752/22,007
RADIX	$2^{22}/2^{24}$ keys	843/3,372
Barnes	$2^{17}/2^{19}$ particles	4,438/19,145
FMM	$2^{17}/2^{19}$ particles	4,109/16,570
Ocean	$514^2/1026^2$ grid	420/1,636
Water	$25^3/40^3$ molecules	553/2,099
KMeans	$2^{20}/2^{22}$ objects	2,967/11,874
BlackS.	$2^{20}/2^{22}$ options	967/3,867

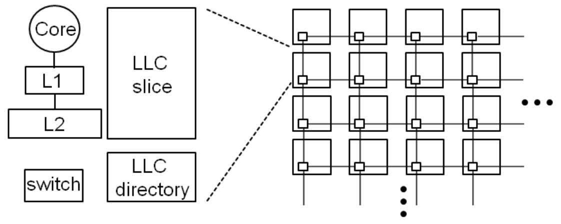


Figure 4: Tiled CMP.

2.3 Performance Model

CRD and PRD profiles provide rich information for shared and private caches, that are two basic components for CMPs. In this work, although we only study the tiled architecture due to its scalability, the same methodology can be extended to other architectures. Figure 4 depicts a tiled CMP example. Each tile contains a core, a private L1 cache, a private L2 cache, and an LLC module. The LLC module can either be a private cache, or a slice of a shared cache. Tiles are connected by a 2D mesh network.

In our study, we assume fully-associative LRU caches, because CRD and PRD profiles can provide accurate program's memory behavior for fully-associative LRU caches. Researchers [1, 15, 22] have shown that RD profiles can be used to approximate the performance of set-associative caches with good accuracy. However, the impact of cache associativity is beyond the scope of this study. We also assume that caches are inclusive, and data blocks can be replicated in private caches. Hence, when a cache miss happens in the L1 and L2 caches, the cache sends a request to the next level cache directly. However, when a cache miss happens in the private LLC, the coherence protocol first checks the on-chip directory. If the cache block resides in the private LLC of another tile, the cache block will be forwarded to the

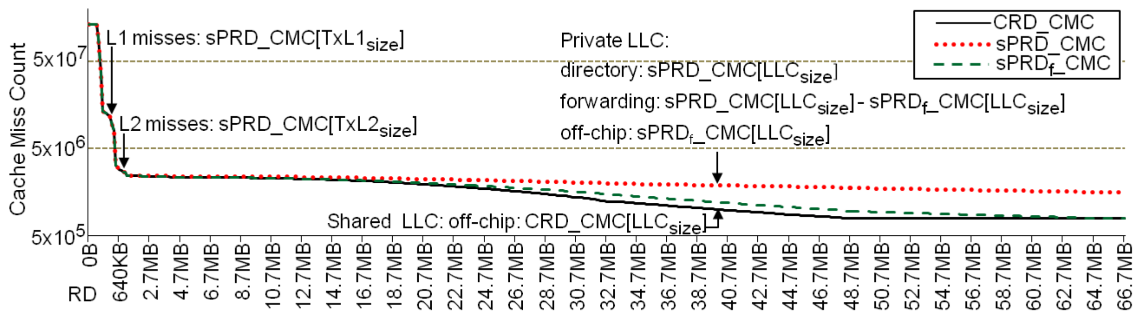


Figure 5: CRD_CMC, sPRD_CMC, and sPRD$_f$_CMC profiles for FFT running on 16 cores at the S1 problem size, indicating the number of cache misses at each cache level.

requesting tile. Otherwise, the request will be sent to the off-chip DRAM. To model the data-forwarding in private LLC, we need another profile (sPRD$_f$) to indicate if the cache block resides in the other tiles or not. To compute the sPRD$_f$ profile, we find the minimum RD among per-thread stacks for a given memory access. We query the depth of the accessed data block in each core's private stack without updating the stacks. The intuition is that this minimum RD is the smallest cache size which can contain a replica of the given cache block.

Figure 5 uses FFT running on 16 cores at S1 to summarize how we compute the cache misses at each cache level. There are three cache-miss count (CMC) profiles, CRD_CMC, sPRD_CMC, and sPRD$_f$_CMC. Each profile plots cache miss count (Y-axis) versus RD (X-axis). RD values are multiplied by the block size, 64 bytes, so the X-axis reports distance as capacity. For capacities 0–128KB, cache capacity grows logarithmically; beyond 128KB, capacity grows in 128KB increments.

To evaluate cache performance, we compute average memory access time (AMAT). By multiplying the cache misses of each level with the access time of the next level cache, we can compute the memory stall at each cache level. Equation 1 computes AMAT for CMP configurations using private LLC. When CPU has a data access, it will access the data cache (L1 cache). The access latency is $L1_{lat}$, and the total number of data references is $sPRD\_CMC[0]$. For private L1 cache, the total number of cache misses (L2 cache accesses) is $sPRD\_CMC[T \times L1_{size}]$, where $L1_{size}$ is the private L1 cache size per core and T is the number of tiles. The L2 access latency is $L2_{lat}$. Next, for private L2 cache, the total number of cache misses (LLC accesses) is $sPRD\_CMC[T \times L2_{size}]$, where $L2_{size}$ is the private L2 cache size per core. The LLC access latency is LLC_{lat}. Finally, for private LLC with total capacity of LLC_{size}, the directory-access traffic is $sPRD\_CMC[LLC_{size}]$, and the access latency is $(DIR_{lat} + HOP_{lat})$. We assume data blocks are distributed uniformly on the LLC slices, so network messages incur $(\sqrt{T}+1)$ hops on average. Hence, HOP_{lat} equals the product of the latency per-hop and the average number of hops per a network message. In other words, $HOP_{lat} = (latency\ per-hop) \times (\sqrt{T}+1)$. After checking the directory, the data-forwarding traffic is $(sPRD\_CMC\ [LLC_{size}] - sPRD_f\_CMC[LLC_{size}])$, and the access latency is $(LLC_{lat} + 2 \times HOP_{lat})$ which contains 2-way data forwarding communication and one cache access to acquire the data. The off-chip memory accesses are $sPRD_f\_CMC[LLC_{size}]$. We also model the 2-way communication when accessing memory controllers, so the off-chip memory access latency is $(DRAM_{lat} + 2 \times HOP_{lat})$. Summing the above terms gets us the total memory access/stall time. Dividing by the total references ($sPRD\_CMC[0]$), we get the average memory access time as in Equation 1.

$$\begin{aligned}
&AMAT_p = (L1_{lat} \times sPRD\_CMC[0] \\
&+ L2_{lat} \times sPRD\_CMC[T \times L1_{size}] \\
&+ LLC_{lat} \times sPRD\_CMC[T \times L2_{size}] \\
&+ (DIR_{lat} + HOP_{lat}) \times sPRD\_CMC[LLC_{size}] \\
&+ (LLC_{lat} + 2 \times HOP_{lat}) \times \\
&\quad (sPRD\_CMC[LLC_{size}] - sPRD_f\_CMC[LLC_{size}]) \\
&+ (DRAM_{lat} + 2 \times HOP_{lat}) \times sPRD_f\_CMC[LLC_{size}]) \\
&/sPRD\_CMC[0]
\end{aligned} \tag{1}$$

For a shared LLC, the L1 and L2 caches above the LLC are the same as in the private LLC case. Thus, the first two terms in Equation 1 carry over to the shared LLC case. The LLC accesses are $sPRD\_CMC[T \times L2_{size}]$ with access latency of $(LLC_{lat} + 2 \times HOP_{lat})$. In shared LLC, the data always resides on the home tile, therefore there is two-way communication. The off-chip memory accesses are $CRD\_CMC[LLC_{size}]$. The DRAM access latency is also $(DRAM_{lat} + 2 \times HOP_{lat})$. Adding the terms up and dividing by the total references, the average memory access time for shared LLC can be modeled as in Equation 2.

$$\begin{aligned}
&AMAT_s = (L1_{lat} \times sPRD\_CMC[0] \\
&+ L2_{lat} \times sPRD\_CMC[T \times L1_{size}] \\
&+ (LLC_{lat} + 2 \times HOP_{lat}) \times sPRD\_CMC[T \times L2_{size}] \\
&+ (DRAM_{lat} + 2 \times HOP_{lat}) \times CRD\_CMC[LLC_{size}]) \\
&/sPRD\_CMC[0]
\end{aligned} \tag{2}$$

These two simple performance models do not consider queuing in the on-chip and off-chip networks. However, they capture the first order effects in cache design (capacity and sharing), and can provide rich insights about optimizing cache hierarchies. In this paper, we study two optimization problems. First, we investigate when shared LLC outperforms private LLC. Then we study the trade-offs between L2 and LLC capacity partition when the total on-chip cache capacity is fixed.

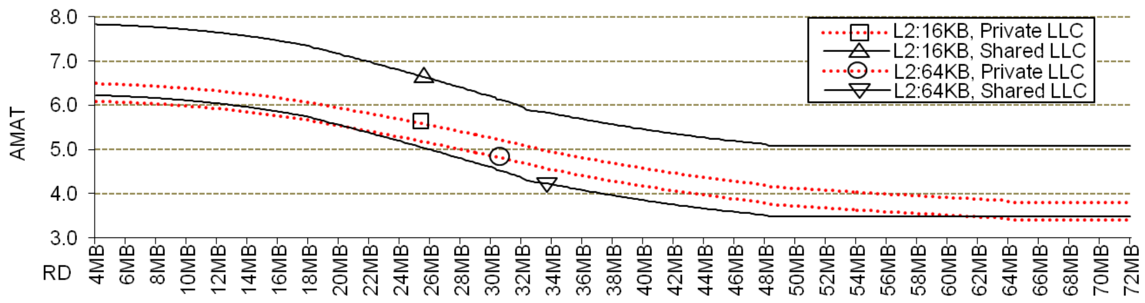


Figure 6: FFT's AMAT$_p$ and AMAT$_s$ for different L2 and LLC capacities.

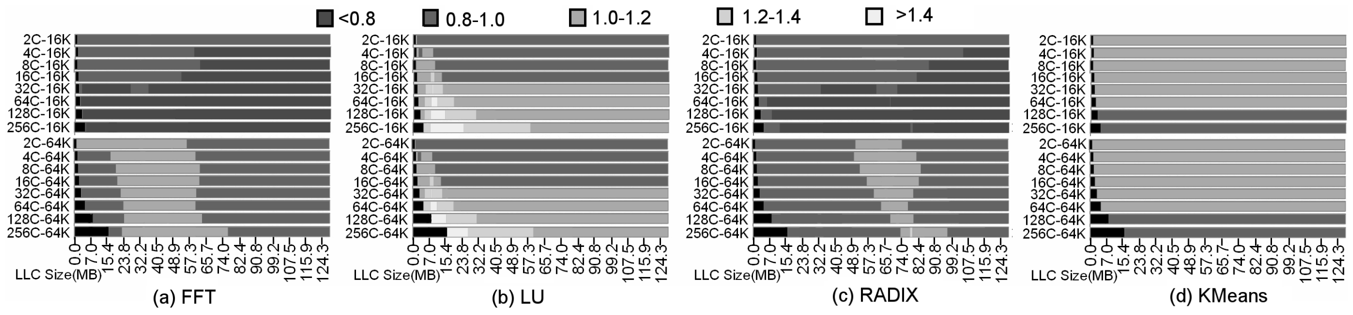


Figure 7: Private vs. shared performance across L2 capacity, LLC capacity, and machine scaling.

3. PRIVATE LLC VERSUS SHARED LLC

Shared LLC is better than private LLC when $AMAT_p > AMAT_s$. Plugging Equations 1 and 2 into this inequality and simplifying yields Equation 3. Equation 3 shows that shared LLC can outperform private LLC when the total off-chip memory stall saved in shared LLC exceeds the total on-chip memory stall saved in private LLC.

$$\begin{aligned}&(DRAM_{lat} + 2 \times HOP_{lat})\times\\&\quad(sPRD_f\_CMC[LLC_{size}] - CRD\_CMC[LLC_{size}]) >\\&(2 \times HOP_{lat}) \times sPRD\_CMC[T \times L2_{size}]\\&- (DIR_{lat} + HOP_{lat}) \times sPRD\_CMC[LLC_{size}]\\&- (LLC_{lat} + 2 \times HOP_{lat})\times\\&\quad(sPRD\_CMC[LLC_{size}] - sPRD_f\_CMC[LLC_{size}])\end{aligned} \tag{3}$$

Figure 6 plots AMAT$_p$ (dotted lines) and AMAT$_s$ (solid lines) as a function of total LLC size for the FFT benchmark running on 16 cores at the S1 problem size. Different pairs of curves show results for different L2 sizes. When computing AMAT, we assume 8KB L1 cache with 1-cycle latency, 4-cycle L2 latency, 10-cycle LLC latency, 10-cycle directory latency, 200-cycle DRAM latency, and 3-cycle per-hop network latency.

Figure 6 shows several insights. At small L2 capacities, the first term in the RHS of Equation 3 ($(2 \times HOP_{lat}) \times sPRD\_CMC[T \times L2_{size}]$) always dominates due to the high shared LLC accesses. So private LLC is always better. This occurs in Figure 6 when the L2 is 16KB. Second, for larger L2 caches, the RHS of Equation 3 reduces, allowing the LHS to dominate if the sPRD$_f$_CMC and CRD_CMC gap is sufficiently large. This occurs in Figure 6 for the 64KB L2 with a private-to-shared LLC cross-over at 20.4MB. Finally, at large private LLC capacities which can contain significant replications, the off-chip traffic of shared and private caches (*i.e.*, CRD_CMC[LLC$_{size}$] and sPRD$_f$_CMC[LLC$_{size}$]) are almost identical. As a result, the LHS of Equation 3 is close to 0. This diminishes the advantage of shared LLC. In fact, private LLC may regain a performance advantage when the total on-chip memory stall in shared LLC is higher than the total on-chip memory stall in private LLC. This occurs in Figure 6 for the 64KB L2 with a shared-to-private LLC cross-over at 62.0MB LLC capacity.

Figure 7 extends the architectural insights discussed above by incorporating machine scaling effects. We select four representative benchmarks (FFT, LU, RADIX, and KMeans) to represent our results. We plot two graphs per benchmark: the top graph shows tiled CPUs with 16KB L2s while the bottom graph shows tiled CPUs with 64KB L2s. Within each graph, LLC capacity is varied from 0–128MB along the X-axis, and machine scaling is studied along the Y-axis for 2–256 cores. For each CPU configuration, the ratio $\frac{AMAT_p}{AMAT_s}$ is plotted as a gray scale. The legend reports the gray scale assignments. In particular, the 3 lightest gray scales (ratio > 1.0) indicate shared caches are best, while the two darkest gray scales (ratio < 1.0) indicate private caches are best. We do not consider CPUs with less total LLC than total L2 cache; these cases are shaded black in Figure 7. All benchmarks run the S1 problem, and each benchmark has 32,166 configurations.

Figure 7 shows a 16KB L2 is not large enough for FFT and RADIX, so significant traffic reaches the LLC in these cases. Hence, private LLCs are always best. Second, when L2 size is beyond the high reference counts region, shared LLC can outperform private LLC when the LHS of Equation 3 dominates. However, the gap between sPRD_CMC$_f$

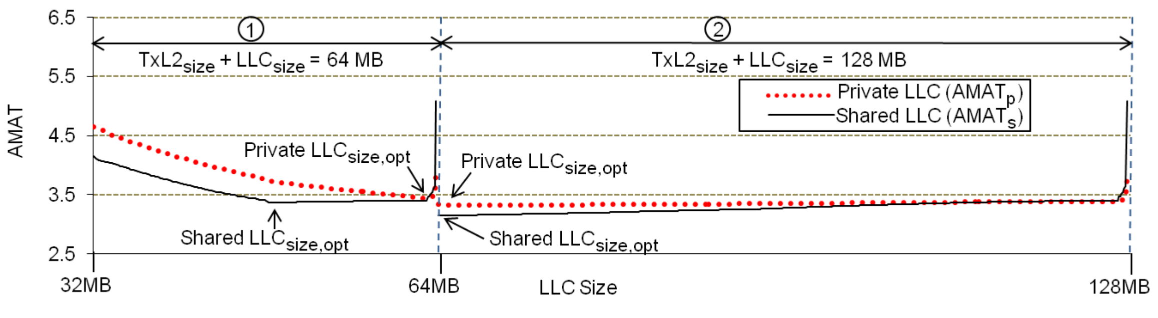


Figure 8: The trade-off between L2$_{size}$ and LLC$_{size}$.

and CRD_CMC varies across LLC capacity, so preference changes from private to shared and back to private again. Figure 7 shows this effect in FFT and RADIX with 64KB L2. LU and KMeans have good data locality, and shared caches are usually best.

Figure 7 also shows machine scaling's impact on private vs. shared cache selection. Machine scaling increases the physical extent of the CPU, making shared LLC accesses more costly. The scaling trend tends to make private LLC more desirable at larger core counts when on-chip memory stall in shared LLC dominates. Figure 7 shows this effect in RADIX and KMeans. LU has a large number of replications which grows with core count. Hence, the scaling trends tend to make shared LLC more desirable at larger core counts in LU due to the big sPRD_CMC/CRD_CMC gap.

4. TRADE-OFF BETWEEN L2 AND LLC CAPACITY

In the previous section, we find L2 capacity has significant impact on cache performance. In this section, we study the trade-off between L2 and LLC capacity when the total on-chip cache capacity (C) is fixed. We assume that L1 is closely coupled with the core, so L1 size is fixed. We can write $T \times L2_{size} + LLC_{size} = C$ and $T \times L2_{size} \leq LLC_{size}$. Substituting from the first equation into the second, the following holds: $LLC_{size} \geq 0.5C$ and $T \times L2_{size} \leq 0.5C$.

Figure 8 plots AMAT$_p$ (dotted lines) and AMAT$_s$ (solid lines) as a function of LLC$_{size}$ for the FFT benchmark running on 16 cores at the S1 problem size. Figure 8 is divided into two regions to represent two total cache capacities ($T \times L2_{size} + LLC_{size}$), marked as ①$64MB$, and ②$128MB$. For each of these two regions, the left-most point represents $T \times L2_{size} = LLC_{size} = 0.5C$. As LLC$_{size}$ increases along X-axis, L2$_{size}$ decreases as $T \times L2_{size} = C - LLC_{size}$. The right-most point represents $L2_{size} = 2 \times L1_{size}$ and $LLC_{size} = C - T \times L2_{size}$. Thus, in region ① at the left edge, the cache sizes are as follows: $L1 = 8KB$, $L2 = 2MB$, and $LLC = 32MB$. Whereas at the right edge, the cache sizes are as follows: $L1 = 8KB$, $L2 = 16KB$, and $LLC = 63.75MB$. In region ② at the left edge, the cache sizes are as follows: $L1 = 8KB$, $L2 = 4MB$, and $LLC = 64MB$. Whereas at the right edge, the cache sizes are as follows: $L1 = 8KB$, $L2 = 16KB$, and $LLC = 127.75MB$.

Figure 8 provides three major insights that are valid for all of our benchmarks. First, when L2$_{size}$ is close to L1$_{size}$ (8KB in our study and marked in Figure 5 by the first arrow to the left of figure close to the Y-axis), the high LLC accesses cause high AMAT. AMAT$_s$ is higher than AMAT$_p$ at small L2$_{size}$ because shared LLC has higher on-chip communication cost than private LLC.

Second, as L2$_{size}$ increases (as we move from right to left in each region), the AMAT drops rapidly. When L2$_{size}$ is large enough to capture the major working set (>640KB, illustrated in Figure 5), further increasing L2$_{size}$ does not reduce LLC accesses significantly. Hence, off-chip traffic grows as L2$_{size}$ keeps increasing (and LLC$_{size}$ decreases), so the AMAT goes up again. The optimal LLC$_{size}$ (LLC$_{size,opt}$) occurs when the change of on-chip memory stall equals to the change of off-chip memory stall–*i.e.*, the total on-chip and off-chip memory stalls are balanced. In Figure 8, region ① shows this behavior, and the LLC$_{size,opt}$ is 48.4MB (62.9MB) for AMAT$_s$ (AMAT$_p$). In region ①, shared LLC outperforms private LLC between 32MB and 62.9MB due to the gap between CRD_CMC and sPRD$_f$_CMC profiles. Because CRD_CMC also decreases faster than sPRD_CMC inside this range, increasing L2$_{size}$ has more benefit in shared LLC configuration. Hence, the shared LLC$_{size,opt}$ is smaller than the private LLC$_{size,opt}$.

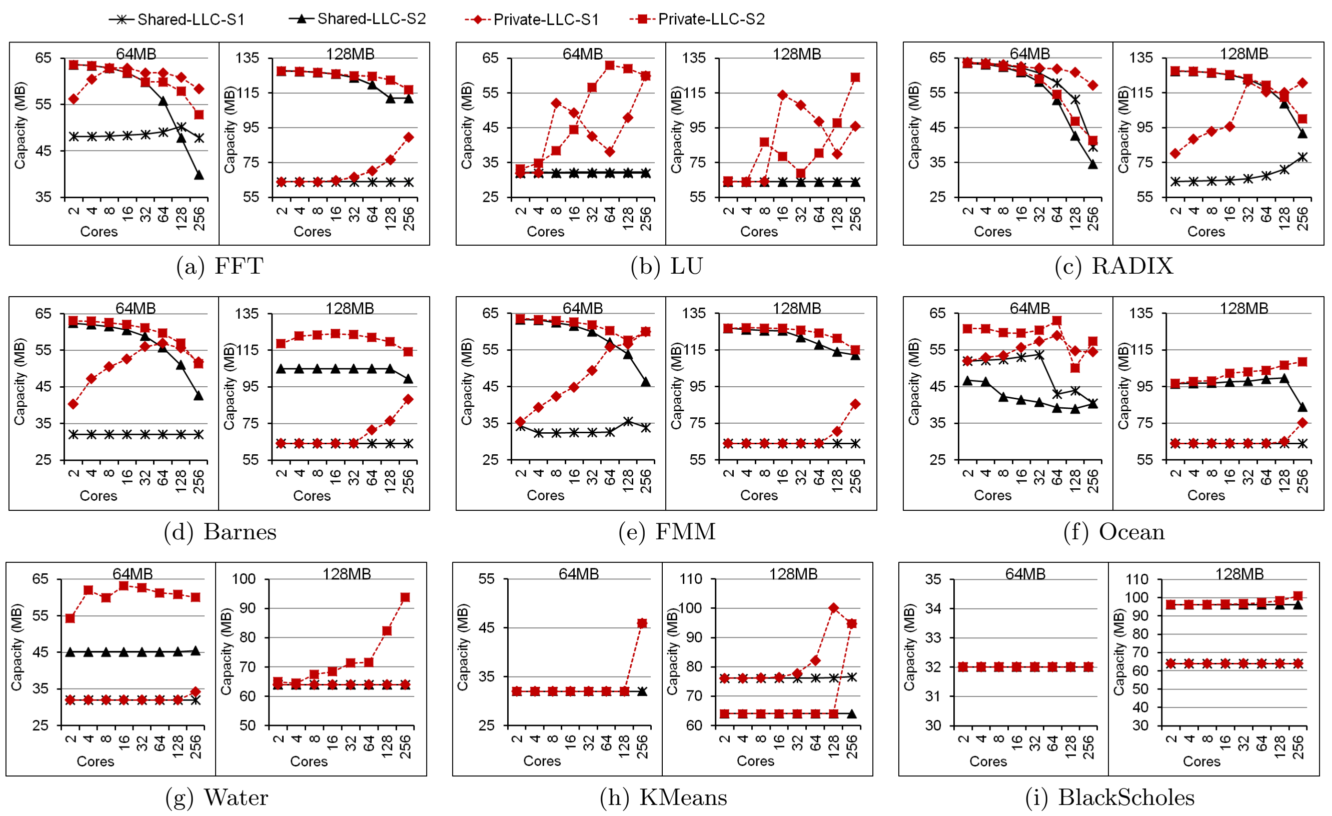
Lastly, when LLC capacity is large enough (region ② in Figure 8), the LLC misses only change slightly with respect to LLC$_{size}$, as illustrated in Figure 5. The decrease of LLC$_{size}$ does not impact off-chip traffic significantly, but the increase of L2$_{size}$ keeps reducing the LLC accesses. Hence, the LLC$_{size,opt}$ is close to $LLC_{size} = T \times L2_{size} = 0.5C$. For AMAT$_s$ (AMAT$_p$), the LLC$_{size,opt}$ is 64.0MB (64.7MB) in Figure 8. Due to the sPRD_CMC/CRD_CMC gap, shared LLC also outperforms private LLC between 64.0MB and 115.5MB.

Figure 9 reports the LLC$_{size,opt}$ for our 9 benchmarks across 24,120 configurations per benchmark. In Figure 9, we plot two capacities, 64MB and 128MB, per benchmark. For each graph, the X-axis is the core counts, and the Y-axis is the corresponding LLC$_{size,opt}$. We report LLC$_{size,opt}$ for two problem sizes, S1 and S2.

All basic insights from Figure 8 are valid in Figure 9. First, for both shared and private LLC$_{size,opt}$, the total L2$_{size}$ must be sufficiently large to reduce LLC access frequency. Second, the LLC$_{size,opt}$ depends on the balance between on-chip and off-chip memory stall. We discuss shared LLC and private LLC separately.

4.1 Shared LLC

Core count scaling only impacts cache performance significantly below a particular cache size for shared caches, and this cache capacity grows with core count [19]. When core count scales, the routing distance also increases in shared LLC. To offset these effects, the optimal $T \times L2_{size}$ (LLC_{size})



(a) FFT (b) LU (c) RADIX

(d) Barnes (e) FMM (f) Ocean

(g) Water (h) KMeans (i) BlackScholes

Figure 9: The private LLC$_{size,opt}$ (dotted lines) and shared LLC$_{size,opt}$ (solid lines) at different problem sizes (S1,S2) and total cache sizes (64M,128M) for our benchmarks.

must grow (decrease) with core count scaling. Figure 9 confirms this across 5 benchmarks (FFT, RADIX, Barnes, FMM, and Ocean) at S2. For LU, KMeans, and BlackScholes, these programs have good data locality, so the optimal LLC$_{size}$ is almost constant across core counts.

Problem scaling affects the LLC$_{size,opt}$ selection in two ways. First, when the problem size is small compared to the total cache capacity, the optimal cache configuration usually happens at $T \times L2_{size} = LLC_{size}$ across different core counts. For example, when total cache capacity is 128MB, the LLC$_{size,opt}$ for the S1 problem size is around half of the total cache sizes–*i.e.*, 64MB –for our 9 benchmarks. Second, because problem scaling increases memory footprint, the LLC$_{size,opt}$ grows as problem scales to reduce the expensive off-chip access. Figure 9 confirms this in FFT, RADIX, Barnes, FMM, and Ocean at the 128MB graphs.

4.2 Private LLC

In private LLC, the minimal $T \times L2_{size}$ should also contain the major working set which grows as core count scales. However, core count scaling also degrades data locality at large RD values due to increased replications and invalidations, which prefer larger LLC. The combined effects make the optimal LLC$_{size}$ behavior complicated.

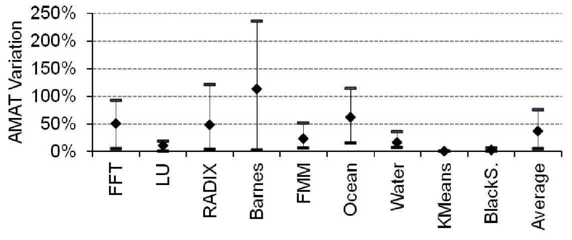
We use the FFT's graph at 128MB in Figure 9 as an example to explain this complicated behavior. For the S1 problem size, the major working set is small and can be contained within small $T \times L2_{size}$. Therefore, the LLC$_{size,opt}$ usually increases with core count scaling to reduce the directory accesses and off-chip traffic. However, for the S2 problem size, the major working set shifts to larger RD values, so reducing private LLC accesses is more critical (*i.e.*, the decreasing rate of on-chip memory stall is faster than the increasing rate of off-chip memory stall). The LLC$_{size,opt}$ usually decreases as core count scales. We see these behaviors in FFT, RADIX, Barnes, and FMM. The private LLC$_{size,opt}$ is also larger than the shared LLC$_{size,opt}$ due to the higher cache misses in private LLCs.

For benchmarks with good data locality (*i.e.*, KMeans and BlackScholes), the LLC$_{size,opt}$ is almost constant across core counts, and the values are very similar to the shared LLC$_{size,opt}$. LU has good data locality in shared caches, but it has bad locality in private caches due to massive replications. Hence, the private LLC$_{size,opt}$ of LU usually grows with core count.

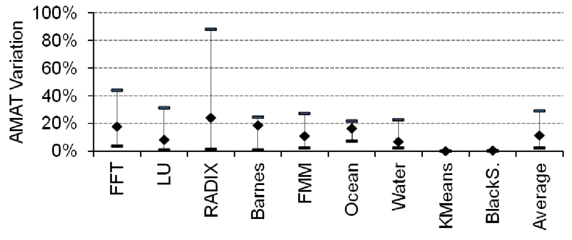
4.3 AMAT Variation of L2/LLC Partitioning

In the previous section, we studied the optimal L2/LLC capacity partition and the impact of scaling. In this section, we quantify the impact of L2/LLC partitioning on overall performance, in order to understand the importance of capacity partitioning between L2 and LLC.

For each benchmark, there are 8 core counts, 2 problem sizes, 2 total cache sizes, and the selection of shared/private LLCs. Hence, there are a total of 64 combinations per benchmark. When we search the design space of L2/LLC partitioning for each combination, we record the lowest AMAT and the highest AMAT values. Then, the AMAT variation of L2/LLC capacity partitioning for each combination is calculated as $\frac{highest\ AMAT - lowest\ AMAT}{lowest\ AMAT} \times 100\%$.

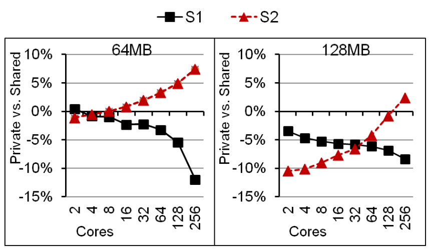


(a) Shared LLC.

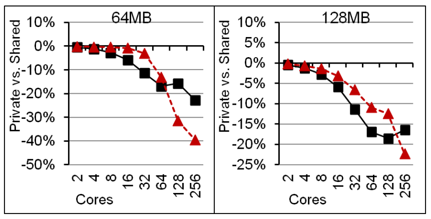


(b) Private LLC.

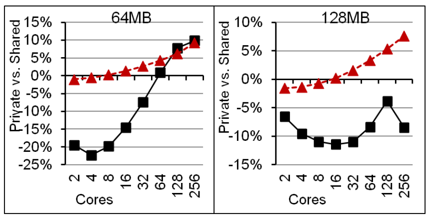
Figure 10: AMAT variation of L2/LLC capacity partitioning.



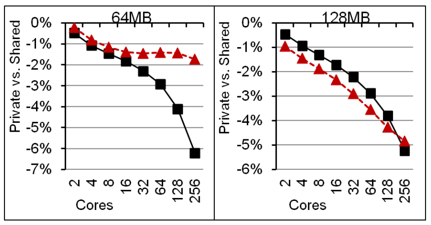
(a) FFT



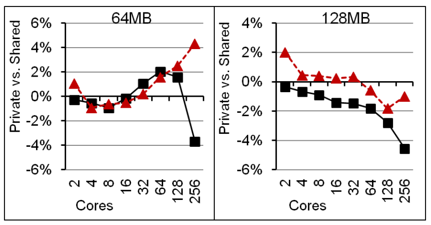
(b) LU



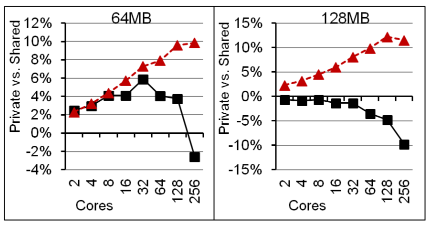
(c) RADIX



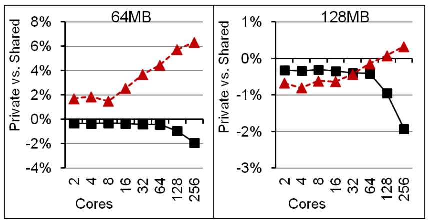
(d) Barnes



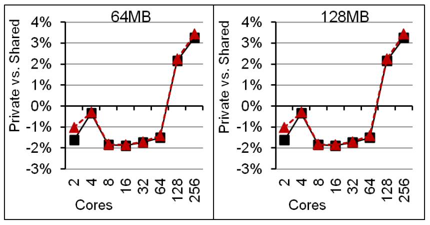
(e) FMM



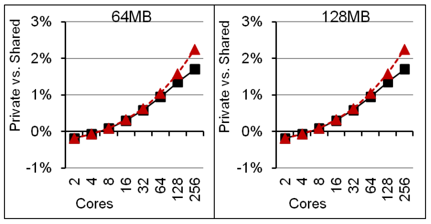
(f) Ocean



(g) Water



(h) KMeans



(i) BlackScholes

Figure 11: AMAT difference between private and shared LLCs at $LLC_{size,opt}$. Private LLC outperforms shared LLC when Y-value > 0.

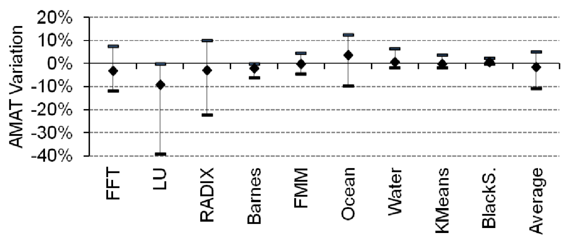
Figure 10(a,b) reports the AMAT variation across our 9 benchmarks for shared and private LLCs, respectively. For each benchmark, the bar shows a range of AMAT variation across 2–256 cores, S1/S2 problem size, and 64MB/128MB total cache size. The diamond marker of each bar is the average AMAT variation. The last bar is the average across all benchmarks.

When applications have very good locality (*i.e.*, KMeans and BlackScholes), the AMAT variation is close to 0 in both shared and private LLCs. Because the small $L1_{size}$ (8KB per core) can capture the main working set. For shared LLCs, the variation can reach 236% in Barnes, as illustrated in Figure 10(a). On average, the variation is between 4% and 76%. Figure 10(b) reports the AMAT variation for private LLCs. The largest variation is 88% in RADIX. On average, the variation is between 2% and 30%. Private LLCs have smaller AMAT variation than shared LLCs. This is because $L2_{size}$ has bigger impact in shared LLC's AMAT due to the high on-chip communication cost.

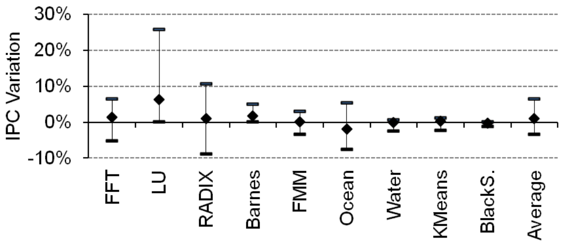
4.4 Private vs. Shared LLC

Lastly, we measure the impact of private-vs-shared LLC on overall performance at the optimal L2/LLC capacity partition. By comparing with the AMAT variation results of the previous section, we can quantify the importance of each optimization choice (private-vs-shared LLC and L2/LLC capacity partitioning).

Figure 11 reports the AMAT variation between private and shared LLCs (*i.e.*, $\frac{AMAT_s - AMAT_p}{AMAT_s} \times 100\%$) at the optimal LLC_{size} for 8 core counts, 2 problem sizes, and 2 total cache capacities. Private LLC outperforms shared LLC when the Y-axis value > 0. Although the preference of shared LLC or private LLC depends on core counts and problem sizes, there are two important trends for cache design. First, when problem size is small compared to on-chip cache capacity, the on-chip communication dominates. The on-chip memory stall (directory access + data forwarding) in private LLC becomes more expensive with core count scal-



(a) AMAT variation.



(b) IPC variation.

Figure 12: Private vs. shared LLC at LLC$_{size,opt}$.

ing. Hence, core count scaling prefers shared LLC. We see this effect in FFT, LU, BARNES, FMM, Ocean, and Water at S1 with 128MB cache. Second, at the same cache capacity, continued problem scaling increases LLC accesses, and physical data locality becomes the major determiner of performance. Furthermore, core count scaling also increases shared LLC's access latency. Hence, we see core count scaling prefer private LLC at S2 with 64MB cache in FFT, RADIX, FMM, Ocean, and Water.

Figure 12(a) summarizes the highest, the lowest, and the average AMAT variation between private and shared LLCs in Figure 11. For each benchmark, the bar shows a range of AMAT variation across core counts, problem sizes and total cache sizes. The diamond marker of each bar is the average AMAT variation. Shared LLC can outperform private LLC by 39% in LU, and private LLC can outperform shared LLC by 12% in Ocean. On average, the variation is between -11% and 5%. We can also model the IPC as $1/(1 + \frac{memory\ accesses}{instructions} \times AMAT)$ by assuming $CPI = 1$ in the absence of memory stall. Figure 12(b) reports the IPC variation–*i.e.*, $\frac{IPC_s - IPC_p}{IPC_s} \times 100\%$. Private LLC outperforms shared LLC when the Y-axis value < 0. The shared LLC's IPC can outperform private LLC's IPC by 26% in LU, and the private LLC's IPC can outperform shared LLC's IPC by 9% in RADIX. On average, the IPC variation is between -3% and 7%. Comparing from Figures 10 and 12, we find that the L2/LLC capacity partitioning has larger impact than private-vs-shared-LLC selection on cache performance. This suggests that physical data locality is very important for future CMP designs.

5. RELATED WORK

There has been significant research on acquiring multicore locality profiles. Ding and Chilimbi [4], Jiang *et al* [7], and Xiang *et al* [20] present techniques to construct CRD profiles from per-thread RD profiles. However, analysis cost can be significant as the numerous ways in which threads' memory references can interleave must be considered. Schuff *et al* [17] acquire locality profiles by simulating uniform memory interleaving of simultaneous threads (much like our approach), and evaluate cache-miss prediction accuracy using the acquired profiles. Schuff predicts shared (private) cache performance using CRD (PRD) profiles. In subsequent work, Schuff speeds up profile acquisition via sampling and parallelization techniques [16]. They also propose using multicore RD analysis and communication analysis to model memory system [8]. However, they do not provide detailed methods and analyses.

We leverage these previous techniques to acquire our CRD and PRD profiles. This work is closely related to our previous work on isolating inter-thread interactions for CRD/PRD profiles across machine scaling [12, 19]. However, this paper focuses on the insights and implications for multicore cache hierarchy design. In particular, to our knowledge, we are the first to identify optimal multicore cache hierarchies via reuse distance analysis, and study the scaling impact on cache design.

6. CONCLUSIONS

Multicore RD profiles provide detailed memory behavior which can help architects understand multicore memory systems. In this work, we create a novel framework based on multicore RD profiles to study multicore cache hierarchies. The key to optimize multicore cache hierarchies is *balancing the total on-chip and off-chip memory stalls.* Hence, the capacity of the last private cache above the last level cache must exceed the region in the PRD profile where significant data locality degradation happens. Shared LLC can outperform private LLC when *the total off-chip memory stall saved in shared LLC is larger than the total on-chip memory stall saved in private LLC.*

We also study how core count scaling and problem size scaling impact the optimal cache design, revealing several new insights. When problem size is large compared to cache capacity, core count scaling usually prefers private LLC. Continued problem size scaling also prefers private LLC. Hence, shared LLC only shows a benefit within a capacity range. At the optimal LLC size, the performance difference between private and shared LLCs is smaller than the performance difference caused by L2/LLC partitioning. This suggests physical data locality is very important for multicore cache system design.

In this study, we assume fixed block size (64 bytes), fully-associative and inclusive caches. The impact of these design parameters requires further investigation. Using detailed simulations to validate the insights and results from reuse distance analysis is also important future work. However, our current study demonstrates that multicore RD analysis is a powerful tool that can help computer architects improve future CMP designs.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments, and Abdel-Hameed Badawy for insightful discussions about the AMAT models and for reviewing the paper.

7. REFERENCES

[1] K. Beyls and E. H. D'Hollander. Reuse distance as a metric for cache behavior. In *Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, 2001.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[3] J. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, 2005.

[4] C. Ding and T. Chilimbi. A composable model for analyzing locality of multi-threaded programs. Technical Report MSR-TR-2009-107, Microsoft Research, 2009.

[5] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell. Exploring the cache design space for large scale CMPs. *ACM SIGARCH Computer Architecture News*, 2005.

[6] J. Huh, S. W. Keckler, and D. Burger. Exploring the design space of future CMPs. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, 2001.

[7] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen. Is reuse distance applicable to data locality analysis on chip multiprocessors? In *Proceeding of Compiler Construction*, 2010.

[8] M. Kulkarni, V. S. Pai, and D. L. Schuff. Towards architecture independent metrics for multicore performance analysis. *ACM SIGMETRICS Performance Evaluation Review*, 2010.

[9] J. Li and J. F. Martinez. Power-performance implications of thread-level parallelism on chip multiprocessors. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2005.

[10] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP design space exploration subject to physical constraints. In *Proceedings of the 12th International Symposium on High Performance Computer Architecture*, 2006.

[11] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Programming Language Design and Implementation*, 2005.

[12] M. Wu and D. Yeung. Understanding multicore cache behavior of loop-based parallel programs via reuse distance analysis. Technical Report UMIACS-TR-2012-1, University of Maryland, 2012.

[13] C. McCurdy and C. Fischer. Using pin as a memory reference generator for multiprocessor simulation. *ACM SIGARCH Computer Architecture News*, 2005.

[14] 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*, 2006.

[15] A. Qasem and K. Kennedy. Evaluating a model for cache conflict miss prediction. Technical Report CS-TR05-457, Rice University, 2005.

[16] D. L. Schuff, M. Kulkarni, and V. S. Pai. Accelerating multicore reuse distance analysis with sampling and parallelization. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, 2010.

[17] D. L. Schuff, B. S. Parsons, and V. S. Pai. Multicore-aware reuse distance analysis. Technical Report TR-ECE-09-08, Purdue University, 2009.

[18] 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*, 1995.

[19] M. Wu and D. Yeung. Coherent profiles: enabling efficient reuse distance analysis of multicore scaling for loop-based parallel programs. In *Proceedings of the 20th International Symposium on Parallel Architectures and Compilation Techniques*, 2011.

[20] X. Xiang, B. Bao, C. Ding, and Y. Gao. Linear-time modeling of program working set in shared cache. In *Proceedings of the 20th International Symposium on Parallel Architectures and Compilation Techniques*, 2011.

[21] 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 *Proceedings of the Workshop on Chip Multiprocessor Memory Systems and Interconnect*, 2007.

[22] Y. Zhong, S. G. Dropsho, and C. Ding. Miss rate prediction across all program inputs. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, 2003.