In the proceedings of the 39th ACM/IEEE International Symposium on Computer Architecture (ISCA), June 2012

FLEXclusion: Balancing Cache Capacity and On-chip Bandwidth via Flexible Exclusion

Jaewoong Sim Jaekyu Lee Moinuddin K. Qureshi Hyesoon Kim

Georgia Institute of Technology
{jaewoong.sim, jaekyu.lee, moin, hyesoon.kim}@gatech.edu

Abstract

Exclusive last-level caches (LLCs) reduce memory accesses by effectively utilizing cache capacity. However, they require excessive on-chip bandwidth to support frequent insertions of cache lines on eviction from upper-level caches. Non-inclusive caches, on the other hand, have the advantage of using the on-chip bandwidth more effectively but suffer from a higher miss rate. Traditionally, the decision to use the cache as exclusive or non-inclusive is made at design time. However, the best option for a cache organization depends on application characteristics, such as working set size and the amount of traffic consumed by LLC insertions.

This paper proposes FLEXclusion, a design that dynamically selects between exclusion and non-inclusion depending on workload behavior. With FLEXclusion, the cache behaves like an exclusive cache when the application benefits from extra cache capacity, and it acts as a non-inclusive cache when additional cache capacity is not useful, so that it can reduce on-chip bandwidth. FLEXclusion leverages the observation that both non-inclusion and exclusion rely on similar hardware support, so our proposal can be implemented with negligible hardware changes. Our evaluations show that a FLEXclusive cache reduces the on-chip LLC insertion traffic by 72.6% compared to an exclusive design and improves performance by 5.9% compared to a non-inclusive design.

1. Introduction

As modern processors step from the multi-core era to the many-core era, the design of memory hierarchies has become even more important. The capacity of the last-level cache (LLC) primarily dictates the amount of the working set that can be stored on-chip. The LLC capacity is unlikely to grow at the same rate as the number of cores on-chip. This means that the LLC capacity per core is not expected to increase for future processor generations; instead, the ratio of the non-LLCs to the LLC is likely to become larger as we go towards many-core on the same die. Figure 1 shows the ratio of the capacity of the on-chip non-LLCs to the total capacity of the LLC for Intel processors over the past 10 years. A lower ratio means that the LLC capacity is much larger than the non-LLC capacity. For example, a ratio of 0.1 indicates that the LLC is 10 times larger than the non-LLCs. Until 2006, this ratio was steadily decreasing as processor designs focused on increasing LLC capacity in the single-core era. However, with the introduction of multi-core processors, the ratio has not scaled down further since 2006. Instead, the ratio has significantly increased in recent micro-architectures that adopt the L3 cache as the LLC. For example, AMD's recent processor, Phenom II, has a very aggressive non-LLC-to-LLC ratio of 0.5, a design in which a 6 MB LLC is shared among six cores that each have a private 512KB L2 cache core [1]. In this case, an inclusive design for the L3 cache would end up consuming half of the LLC capacity for simply replicating the blocks that are already resident in the L2 cache. Therefore, an exclusive cache becomes an attractive design choice to increase effective cache capacity for such designs.

Inclusive LLCs have the advantage of simplifying the cache coherence protocol [3] by avoiding snoop traffic for inner-level caches. However, inclusion requires that the same cache block be duplicated in multiple cache levels, which reduces the effective LLC size, thus increasing the number of off-chip accesses. On the other hand, exclusive LLCs, which are preferred by AMD and VIA processors [1, 20], fully utilize the cache capacity by avoiding duplication of cache blocks and allowing the snoop traffic to go to the inner level caches. Unfortunately, exclusive caches increase on-chip bandwidth requirements significantly due to the need to insert clean victims from the non-LLCs to the LLC [8, 24].

In fact, there is no cache configuration (inclusion, exclusion, or non-inclusion) that works best for all workloads. Each design has its own pros and cons, depending on the workload and the effective ratio of the non-LLCs to the LLC size. This motivates us to investigate dynamic designs that can configure the cache inclusion property on-the-fly depending on the workload requirements. To that end, we propose a flexible inclusion scheme, called *FLEXclusion,* that adaptively configures the LLC inclusion property depending on the cache capacity requirement and on-chip traffic consumption. With FLEXclusion, the LLC acts like an ex-

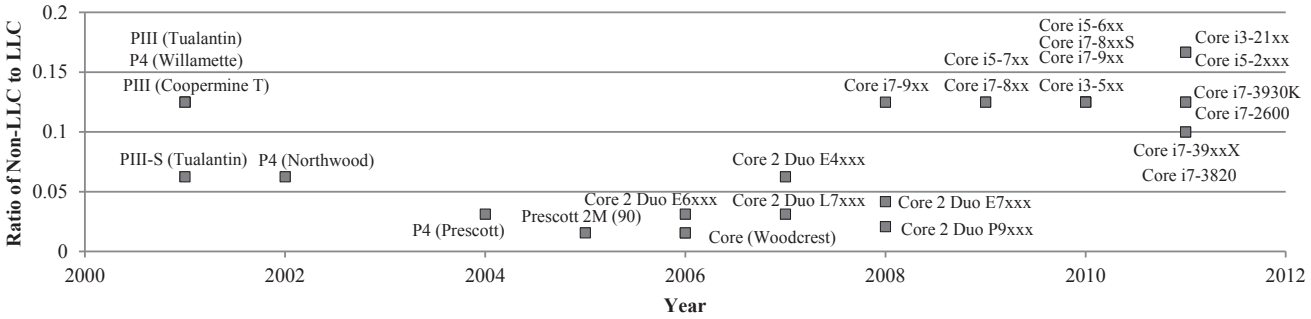


Figure 1. Ratio of cache capacity of the non-LLCs to the LLC for Intel processors over the past 10 years.

clusive cache for workloads that benefit from more cache capacity and behaves like a non-inclusive cache when extra cache capacity is not useful, thereby reducing the on-chip bandwidth and power consumption. The idea of FLEXclusion stems from the key observation that both non-inclusive and exclusive caches rely on similar hardware. Hence, FLEXclusion can easily switch the cache inclusion property between non-inclusion and exclusion without too much additional hardware. FLEXclusion only requires (1) a mechanism to detect workload behavior (fortunately such mechanism is already used in modern cache designs [9, 10, 16]), and (2) a few configuration registers and gates, which do not alter any address or data path in the cache hierarchy.

Compared to an exclusive cache, FLEXclusion reduces power consumed in the on-chip interconnect and cache insertions. And, compared to a non-inclusive cache, FLEXclusion improves performance by freeing up cache space when the workload needs cache capacity. Our evaluations show that FLEXclusion effectively reduces on-chip LLC insertion traffic over exclusive caches by 72.6% on average with only a 1.6% performance degradation. This helps FLEXclusion reduce the on-chip network power consumption by 20.5%. On the other hand, compared to non-inclusion, FLEXclusion improves performance by 5.9%. Thus, the dynamic selection of FLEXclusion between exclusion and non-inclusion can be used either to save power or to improve performance.

2. Background and Motivation

Based on the inclusion property, multi-level cache hierarchies can be classified into three designs: *inclusive*, *non-inclusive*, and *exclusive*. A cache is referred to as *inclusive* when a cache organization forces the cache to always hold cache blocks that are residing in the upper-level(s). An exclusive cache is designed not to cache any blocks that are already present in the upper-level cache(s). A non-inclusive cache is not forced by either inclusion or exclusion (i.e., it *may* contain the upper-level cache blocks).

In a three-level cache hierarchy, lower-level caches (L2 and L3) may have different inclusion policies. For example, L3 can be inclusive (i.e., cache blocks in L1 or L2 must exist in L3), while L2 is non-inclusive (i.e., cache blocks in L1 may exist in L2) as in Intel processors [6, 7]. In this paper, we focus on the inclusion property only between L2 and L3 caches. The next section describes the differences in data flows and relative on-chip traffic for the three designs.

2.1. On-chip/Off-chip Traffic in Multi-level Caches

Figure 2 illustrates the insertion/eviction flows of memory blocks for the three cache designs. In each design, the operation flows triggered by an L3 access are different, which results in the difference in traffic among the three cache hierarchies. We assume that the L2 cache is non-inclusive and the L3 caches follow typical designs for each inclusion property. Without loss of generality, we omit the L1 cache in the figure in order to focus on the flows of the L2 and L3 caches. Before discussing on-chip/off-chip traffic, note that the insertions of (❷-❻) consume on-chip bandwidth, while the others (❶,❼) consume off-chip bandwidth.

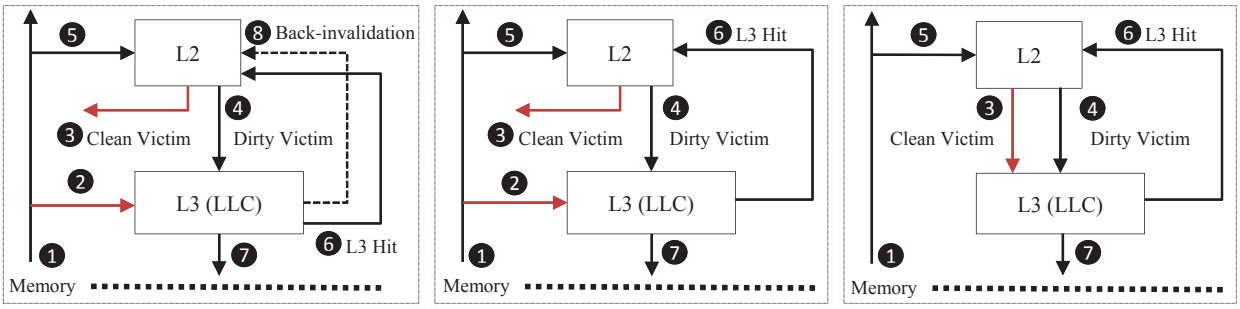
Inclusive/Non-Inclusive Caches: First, on an *L3 miss*, the requested block is brought into both the L2 (❺) and L3 (❷) caches. Thus, the traffic produced by the L3 miss is one on-chip (❷,❺) and one off-chip (❶) transfers. On an *L3 hit*, the line is brought into L2 while the line still remains in the L3 cache, which generates one on-chip transfer (❻).

Whenever there is an *L3 access* (hit or miss), one victim block should be evicted from the L2 cache (❸ or ❹). The data flow of this eviction is the same in the inclusive and non-inclusive caches as they silently drop the clean victims.[1] Dirty victims from L2 are inserted into the position where the same block resides. In the non-inclusive design, however, the block may not exist in the L3 cache. In this case, a victim is selected in L3, so an off-chip transfer (❼) may occur depending on the dirtiness of the L3 victim. In general, however, write-back traffic to memory results from traffic (❷) and $\%\text{dirty}_{\text{L3}}$ as in the inclusive caches.

From the perspective of block insertions/evictions, inclusive and non-inclusive caches have identical flows, and the difference in design only results from the back-invalidation[2], which invalidates the same line in the upper-level caches (L1 and L2) on L3 evictions to enforce the inclusion property (❽). In summary, for both designs, the on-chip ($T_{\text{on-chip}}$) and the off-chip traffic ($T_{\text{off-chip}}$) resulting

[1] We describe the most common implementation of non-inclusion that silently drops clean victims from L2 to L3. Alternative designs are possible but are outside the scope of our study.

[2] While there are several methods to enforce inclusion, the back invalidation method is the most popular one.



(a) Inclusive LLC (b) Non-Inclusive LLC (c) Exclusive LLC

Figure 2. Block insertion/eviction flows for three different multi-level cache hierarchies.

from L3 accesses can be expressed as follows:

$$\begin{aligned} T_{\text{on-chip}} &= T_{\rightarrow L2}\ (❺,❻) + T_{L2\rightarrow L3}\ (❹) \\ &= \#\text{miss}_{L3}\ (❺) + \#\text{hit}_{L3}\ (❻) + \#\text{miss}_{L2} \times \%\text{dirty}_{L2}\ (❹) \\ &= \#\text{miss}_{L2}\ (❺,❻) + \#\text{miss}_{L2} \times \%\text{dirty}_{L2}\ (❹) \end{aligned} \quad (1)$$

$$\begin{aligned} T_{\text{off-chip}} &= T_{\text{MEM}\rightarrow}\ (❶) + T_{L3\rightarrow\text{MEM}}\ (❼) \\ &= \#\text{miss}_{L3}\ (❶) + \#\text{miss}_{L3} \times \%\text{dirty}_{L3}\ (❼) \end{aligned} \quad (2)$$

Exclusive Caches: On an *L3 miss*, the requested block is inserted *only into the L2 cache* (❺), unlike in the other designs. However, note that one L3 miss generates the same amount of on/off-chip traffic as that in the inclusive/non-inclusive caches (❶,❺). On an *L3 hit*, the hitting line is inserted into the L2 cache while invalidating the line in L3. Thus, the traffic resulting from an L3 hit is the same as that in inclusive/non-inclusive caches.

Similar to the inclusive/non-inclusive caches, an L3 access (hit or miss) leads to an L2 eviction. However, in exclusive caches, the victim line is *always installed into the L3 cache regardless of its dirty status* (❸,❹). Note that clean victims (❸) in inclusive/non-inclusive caches are always dropped silently without the insertion into L3 caches. The victims (❸,❹) from L2 cause L3 evictions unless they are inserted into the position invalidated due to L3 hits. Note that, depending on cache designs, the L2 victim resulting from an L3 hit may also not be installed into the invalidated position (i.e., the hitting line) in L3. In summary, the traffic in the exclusive cache can be represented as follows:

$$\begin{aligned} T_{\text{on-chip}} &= T_{\rightarrow L2}\ (❺,❻) + T_{L2\rightarrow L3}\ (❸,❹) \\ &= \#\text{miss}_{L3}\ (❺) + \#\text{hit}_{L3}\ (❻) + (\#\text{hit}_{L3} + \#\text{miss}_{L3})\ (❸,❹) \\ &= \#\text{miss}_{L2}\ (❺,❻) + \#\text{miss}_{L2}\ (❸,❹) \end{aligned} \quad (3)$$

$$\begin{aligned} T_{\text{off-chip}} &= T_{\text{MEM}\rightarrow}\ (❶) + T_{L3\rightarrow\text{MEM}}\ (❼) \\ &= \#\text{miss}_{L3}\ (❶) + \#\text{miss}_{L2} \times \%\text{evict}_{L3} \times \%\text{dirty}_{L3}\ (❼) \end{aligned} \quad (4)$$

Traffic Comparison of Non-Inclusion and Exclusion: Using the above equations, the difference in traffic between the non-inclusive and exclusive designs can be expressed as

$$\begin{aligned} T_{\text{diff}} &= T_{\text{on-chip\_diff}} + T_{\text{off-chip\_diff}} \\ &= \#\text{miss}_{L2} - \#\text{miss}_{L2} \times \%\text{dirty}_{L2} + T_{\text{off-chip\_diff}} \\ &= \#\text{miss}_{L2} \times \%\text{clean}_{L2} + (T_{\text{off-chip\_EX}} - T_{\text{off-chip\_NICL}}). \end{aligned} \quad (5)$$

For on-chip traffic, note that $\#\text{miss}_{L2}$ is not affected by the L3 cache designs. Instead, it is a function of the parameters of the L2 cache and program behavior. Thus, the L2 insertion traffic (❺,❻) is the same between Eq. (1) and Eq. (3), and they cancel out. As a result, the difference in traffic only comes from the *on-chip L3 insertion traffic*, which is (❸,❹). We can also expect that the on-chip L3 insertion traffic remains the same, in both designs, even if the L3 size is changed since it is determined by $\#\text{miss}_{L2}$ and $\%\text{dirty}_{L2}$, which are independent of the L3 configurations.

We will not elaborate the difference in off-chip traffic as we focus on on-chip traffic.[3] In general, $T_{\text{off-chip\_EX}}$ is smaller than $T_{\text{off-chip\_NICL}}$ since the L3 miss ratio in exclusive caches is lower than that in non-inclusive caches. However, if running workloads fit in the L3 cache or have very large memory footprints, the difference in miss ratio could be insignificant. In this case, the difference in off-chip traffic between non-inclusive and exclusive designs is negligible.

2.2. Need for Dynamic Mechanism

Exclusive LLCs usually perform better than non-inclusive LLCs for the same cache size. Figure 3 shows that this is mainly due to the increase in effective cache capacity.

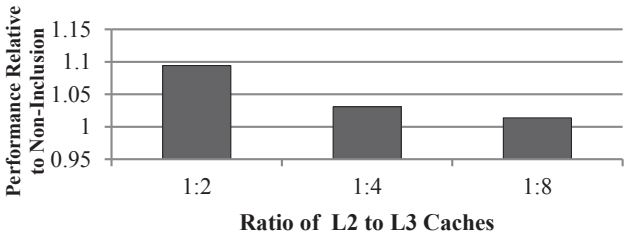


Figure 3. Performance of non-inclusive and exclusive LLCs varying the LLC size for SPEC2K6 benchmarks.

With a 1:2 ratio for the L2 and L3 caches, the increase in cache size is 50% over the non-inclusive design, which results in a performance difference of 9.4% between the two designs. However, as the ratio becomes smaller (1:4 and 1:8), the performance gain over non-inclusive LLCs

[3]The difference in off-chip transfers between non-inclusive and exclusive caches is much smaller than the difference in on-chip transfers.

becomes insignificant, thereby making non-inclusive LLCs perform comparably to exclusive LLCs.

On the other hand, the LLC insertion traffic in the exclusive hierarchy is typically much higher than in the non-inclusive one, and the traffic remains the same even though the L3 size is increased since it is determined by the L2 cache size for a given application (Eq. (3)). This higher amount of traffic can only be justified when there is a noticeable performance difference between the exclusive and non-inclusive LLCs. To provide a concrete example, we measure performance and L3 insertions per kilo instructions (L3 IPKI) of a dense matrix multiplication application whose working set size is 1MB (L2 cache size is 256KB). Figure 4 shows that if the L3 size is 512KB, the exclusive L3 performs 1.45 times better than the same size of the non-inclusive L3 cache. However, if the L3 size is greater than 1MB, the performance difference becomes smaller. Note that the L3 IPKI remains the same across all configurations. Thus, for this workload, considering the bandwidth consumption as well as performance, the 1MB and 2MB L3 should be non-inclusive, while the 512KB L3 should be exclusive. This implies that the best cache configuration depends on workload behavior, and a static policy is unable to adapt across workload changes.

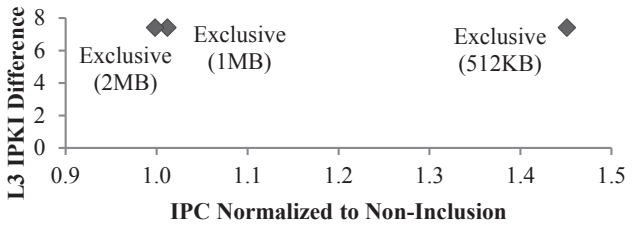


Figure 4. For dense matrix multiply, performance and L3 IPKI difference as the LLC size is varied for an exclusive design.

3. FLEXclusion

To avoid the disadvantages of a static cache organization, we propose *FLexible EXclusion (FLEXclusion)*. A cache that implements FLEXclusion is called a FLEXclusive cache. A FLEXclusive cache operates in one of two modes: *exclusive mode* or *non-inclusive mode*, which are determined at run-time using traffic monitoring and decision mechanisms. This section describes the proposed FLEXclusion scheme.

3.1. Operation of FLEXclusive cache

Figure 5 shows an overview of the FLEXclusive cache. The data flow is composed of both non-inclusion and exclusion data paths, and two additional logic/registers control the data flow based on operating mode: (1) *EXCL-REG*: one-bit register in the L2 cache controller to decide whether clean victims are inserted into L3 or silently dropped, (2) *NICL-GATE*: logic that controls the data flow for incoming blocks from DRAM to L3.

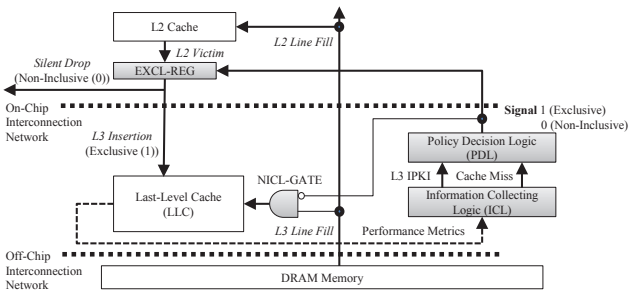


Figure 5. Hardware mechanisms and data flows in the FLEXclusive cache (shaded boxes indicate additional logic).

Exclusive Mode: Policy decision logic (PDL) (Section 3.2.2) sends the signal 1 (exclusion) to (1) NICL-GATE: in order not to insert incoming blocks from the DRAM to the L3 cache, (2) L3 insertion path: to invalidate the hitting lines in L3, and (3) EXCL-REG: to insert L2 clean victims to the L3 cache. The exclusive mode in FLEXclusion behaves the same as in typical exclusive caches except for the L3 insertion of L2 victims, which is performed in a similar way to write-back updates in non-inclusive caches.[4]

Non-Inclusive Mode: As opposed to the exclusive design, the PDL resets the EXCL-REG to prevent clean victim insertions into the L3 cache. Also, all incoming blocks from the DRAM are inserted into both L2 and L3 caches. Although there could be multiple design choices regarding non-inclusive caches, the non-inclusive mode in the FLEXclusive cache follows the typical non-inclusive cache behavior described in Section 2. The characteristics of the operating modes are summarized in Table 1.

3.2. Operating Mode Decision

3.2.1. Collecting Cache Information The information collecting logic (ICL) in Figure 6 gathers information from the L3 cache based on the set dueling method [16] but is extended to check cache misses as well as insertion traffic.

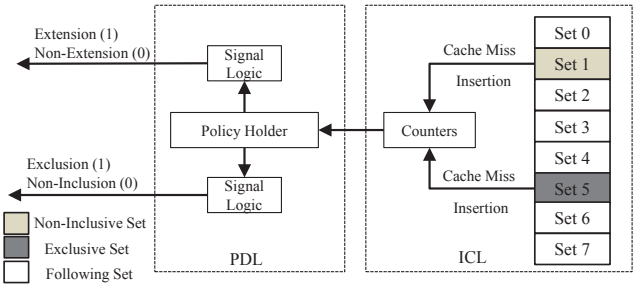


Figure 6. High-level block diagram for mode selection.

In our evaluations, we set 16 dedicated non-inclusive sets and 16 dedicated exclusive sets per 1024 L3 sets (the number of required sampling sets is well discussed in [16]),

[4]Note that this does not incur additional costs since a tag lookup needs to be performed in typical exclusive caches as well.

Operating Mode	Clean Victims (L2)	Line Fill (DRAM)	EXCL-REG	NICL-GATE (output)	Traffic ($T_{\text{on-chip}}$)
Non-inclusive mode	Silent drop	Both L2 and L3	Set to false (0)	True (Requested data)	Eq. (1)
Exclusive mode	Insert	Only L2	Set to true (1)	False (None)	Eq. (3)

Table 1. Characteristics of different operating modes in FLEXclusion.

and the other sets follow the winning policy. Each dedicated exclusive/non-inclusive set follows its exclusion/non-inclusion behaviors. For example, if there is a hit on one of the exclusive dedicated sets, the hitting line is invalidated, while if the hit is on one of the non-inclusive sets, the hitting line remains in the L3 cache. For the dedicated sets, the L3 insertion traffic and cache misses are monitored and then are fed to the policy decision logic periodically. We spread the dedicated sets across the L3 cache sets to improve the effectiveness of set sampling.

3.2.2. Setting Operating Mode Policy decision logic (PDL) decides which mode should be triggered for the FLEXclusive cache using the information from ICL. If the estimated performance of the exclusive mode is higher than that of the non-inclusive mode, PDL sends a signal to the L3 cache to run it as the exclusive mode. Otherwise, the FLEXclusive cache is configured as the non-inclusive mode. We used two metrics to determine the operating region of FLEXclusion: cache misses (performance) and L3 insertions per kilo instructions (traffic). Figure 7 illustrates the operating regions including extensions of FLEXclusion that will be explained in the following section.

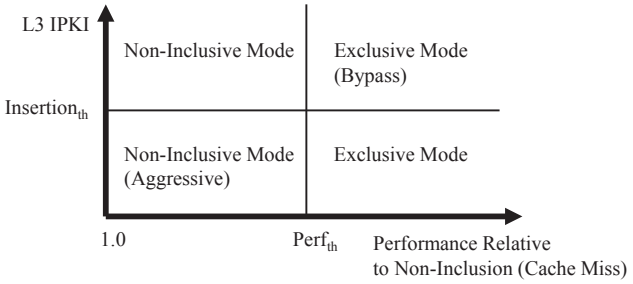


Figure 7. Operating regions for FLEXclusive caches.

FLEXclusion does not require any special actions on a transition from one mode to another. For example, if L3 is switched to non-inclusive mode, L2 clean victims are dropped even though they might not be in L3. However, this does not cause any correctness problems. On a switch to exclusive mode, the victims from L2 may already exist in the L3 cache. However, FLEXclusion does not cause any duplicate copies since the L2 victims are inserted into the position where the same block resides in such cases.

3.3. Extensions of the FLEXclusive Cache

3.3.1. Per-core Inclusion Policy (PER-CORE) When multiple applications are running on different cores, one application may undesirably determine the decision for an L3 organization. To address this problem, the FLEXclusive cache can be configured with a per-core policy. Now, it has distinct dedicated sets for different cores to detect application characteristics. Every L2 block also needs core identification information, and fortunately, it is already provided in most of the current processors to indicate cache affinity.

3.3.2. Aggressive Non-Inclusive Mode (AGG) In the non-inclusive cache, a cache block can exist only in the L2 cache, but not in the L3 cache. In the FLEXclusive cache, since the L3 cache can be switched between non-inclusive mode and exclusive mode, this situation can occur more often than other typical non-inclusive caches. In this extension, when a clean L2 victim is evicted, the L3 cache is checked to see if it contains the evicted line. If not, the victim will be installed into the L3 cache instead of being dropped, which effectively increases the L3 hit ratio at the cost of increases in insertion traffic and address bandwidth.

3.3.3. Operating with Other Cache Techniques (BYPASS) FLEXclusion is orthogonal to cache replacement, insertion, and bypass algorithms; thus, they can be synergistically applied together. One of the interesting combinations could be the integration with bypass techniques since the FLEXclusive cache requires the same amount of bandwidth as that of exclusive caches when it runs as the exclusive mode. Thus, we can employ bypass algorithms on FLEXclusion's exclusive mode to further reduce the on-chip bandwidth consumption. In Section 5.5, we show that the FLEXclusive cache seamlessly operates with one of the state-of-the-art bypass algorithms.

3.4. Hardware Overhead and Design Changes

FLEXclusive cache requires negligible hardware overhead and design changes, as shown in Figure 5. Compared to the baseline caches (both exclusive and non-inclusive caches), we need an additional four registers to record performance and IPKI information for each mode, one comparator, set dueling support[5], and a few configuration gates/registers to control the insertion/eviction flows of memory blocks. All the data flows required to implement the FLEXclusive cache are already necessary for both traditional exclusive and non-inclusive caches; thus, the FLEXclusive cache simply leverages these pre-existing data paths.

3.5. Impact on Cache Coherence

Both non-inclusive and exclusive caches do not guarantee the inclusion property; therefore they need to either have a coherence directory or support snooping of inner level caches. As exclusive and non-inclusive caches have similar flow of coherence traffic, we do not need to modify the coherence network in order to support FLEXclusion.

[5]Note that the overhead to support set dueling is negligible when every 32nd set is selected for set sampling. We only need one 5-input NOR gate.

4. Experimental Methodology

Simulation Infrastructure: We use MacSim [2], an x86 simulator, for our evaluations. Table 2 shows our baseline processor configurations. We keep the L1 and L2 cache parameters consistent for this study while varying the inclusion property and the size of the L3 cache.

Core	1-4 cores, 4.0 GHz out-of-order, 256 ROB 4 issue width, gshare branch predictor
L1 Cache	32KB I-Cache + 32KB D-Cache (3-cycle), 64B line size
L2 Cache	8-way 512KB (8-cycle), 64B line size
L3 Cache	16-way shared 4MB (4 tiles, 25-cycle), 64B line size
Interconnect	Ring (cores, L3, and a memory controller), each hop takes one cycle
DRAM	DDR3-1333, FR-FCFS, 9-9-9 (CL-tRCD-tRP)

Table 2. Baseline processor configuration.

Benchmarks: We use the SPEC CPU2006 benchmarks and sample 200M instructions using SimPoint [17]. Then, based on the misses per kilo instructions (MPKI) on the baseline processor configuration, the benchmarks are categorized into three different groups. Group A represents the benchmarks whose working set sizes are greater than the L2 cache but have benefits from the L3. Group B consists of the benchmarks whose working set sizes are much greater than the L3. Finally, the benchmarks in Group C either are non-memory intensive or have small working set sizes. Table 3 summarizes the classification of the benchmarks.

Group	Benchmarks (SPEC2006)
A: LLC Beneficial (6)	bzip2, gcc, hmmer, h264, xalancbmk calculix
B: LLC Less-beneficial (14)	mcf, libquantum, omnetpp, astar, bwaves milc, zeusmp, cactusADM, leslie3d soplex, gemsFDTD, lbm, wrf, sphinx3
C: Non-memory Intensive (9)	perlbench, gobmk, sjeng, gamess gromacs, namd, dealII, povray, tonto

Table 3. Benchmark classification.

Based on the classification, we select six benchmarks from Group A and seven benchmarks from Group B for the single-threaded experiments. The benchmarks in Group C neither cause L3 insertion traffic nor experience performance differences between non-inclusive and exclusive designs since they fit into the L2 cache; therefore, we exclude Group C. For the multi-programmed workload study, we select benchmarks from Groups A and B and mix the benchmarks. For the experiment, if a faster thread finishes its entire instructions, the thread continues to execute from the beginning to keep competing for cache resources, which is similar to the methodology used in [5,9,23]. Tables 4 and 5 summarize the benchmarks used for single-threaded and multi-programmed workloads in this study, respectively.

Energy Model: We use Orion [22] to discuss network power consumption in Section 5.6. We model an on-chip network with four virtual channels (VCs) per physical channel, where each VC has 5 FIFO buffers. The router operates with 2.0GHz clock frequency at 1.2V in a 45nm technology. The link width is 64b, so that a data message is decomposed into 9 flits (1 for address and 8 for data), while an address packet is composed of 1 flit. The link length is 5mm. With these parameters, dynamic and leakage power are calculated from Orion. Table 6 summarizes the modeling parameters in each router component for a flit.

	MPKI			MPKI	
Group A	L2	L3	Group B	L2	L3
bzip2	4.13	0.77	mcf	65.94	59.13
gcc	4.64	1.50	omnetpp	19.05	14.15
hmmer	3.28	0.19	bwaves	23.74	23.53
h264	1.52	0.19	soplex	35.16	23.39
xalancbmk	2.08	1.56	leslie3d	32.23	29.76
calculix	2.21	1.79	wrf	25.80	22.49
			sphinx3	19.61	16.71

Table 4. Single-threaded workloads and their MPKI values.

Type	Descriptions
2-MIX-S	combine the same two benchmarks in Groups A and B
2-MIX-A	combine all possible two benchmarks in Groups A and B
4-MIX-S	combine the same four benchmarks in Groups A and B

Table 5. Benchmarks for multi-programmed workloads.

Buffers	11.8 (pJ)	Crossbar	9.5 (pJ)	Arbiters	1.8 (pJ)

Table 6. Router energy breakdown per 64-bit flit.

5. Results and Analysis

FLEXclusion aims to be either exclusive or non-inclusive to get either the performance benefit in exclusion or the bandwidth savings in non-inclusion at run-time. Hence, we compare performance with exclusive caches and bandwidth consumption with non-inclusive caches when discussing the results of FLEXclusion (Sections 5.2 to 5.4).

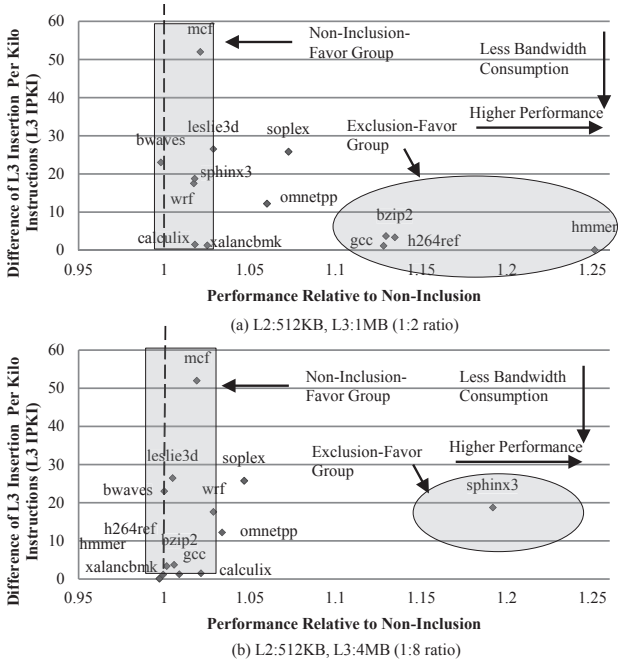


Figure 8. Performance and L3 IPKI difference between exclusion and non-inclusion (512KB L2, (a) 1MB L3, (b) 4MB L3).

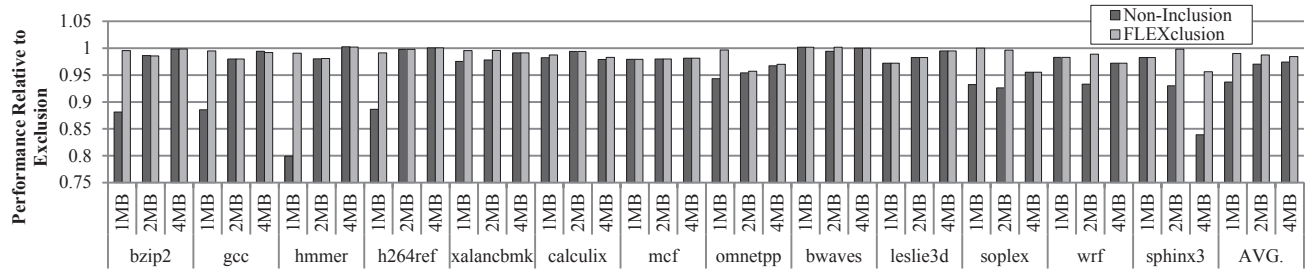


Figure 9. Performance of non-inclusion and FLEXclusion normalized to exclusion for the single-threaded workloads.

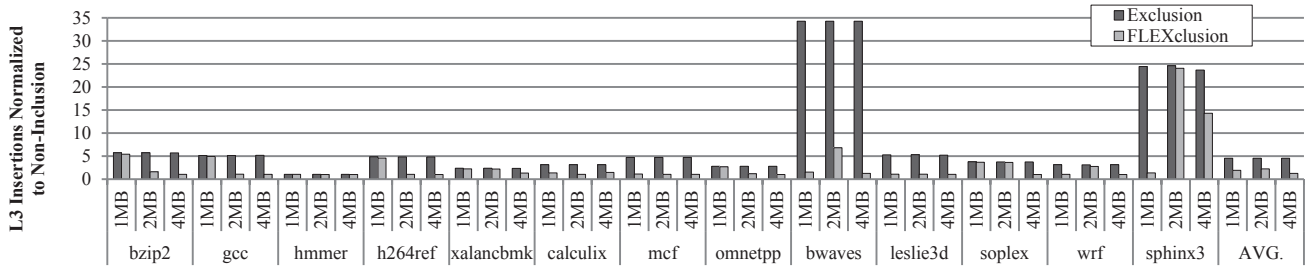


Figure 10. L3 insertions of exclusion and FLEXclusion normalized to non-inclusion for the single-threaded workloads.

5.1. Exclusive Caches vs. Non-inclusive Caches

First, we study the potential benefits of the FLEXclusive cache. Figure 8 shows the performance and L3 insertions per kilo instructions (L3 IPKI) between non-inclusive and exclusive L3 caches for 1MB and 4MB cache sizes. In both figures, the x-axis represents the performance of the exclusive L3 cache normalized to the non-inclusive L3 cache, and the y-axis shows the L3 IPKI difference between the non-inclusive and exclusive caches.

In Figure 8(a), with the 1MB L3 size, benchmarks such as *hmmer* (1.25), *gcc* (1.13), *bzip2* (1.13), and *h264* (1.13) show more than a 10% performance improvement over the non-inclusive cache. However, the other benchmarks actually do not benefit much from the exclusive cache even with a 1:2 ratio of the L2 and the L3 cache size while generating much more traffic than the non-inclusive cache. For example, *bwaves* does not benefit at all but generates 23.01 more L3 IPKI traffic than on the non-inclusive cache, so running the workload on the exclusive cache is not a good idea. This benchmark is a perfect streaming application whose L3 hit ratio is zero and 97% of L2 victim lines are clean (large memory footprint as well). Thus, from Eq. (5), we can infer that the traffic difference between exclusion and non-inclusion is significant for this workload.

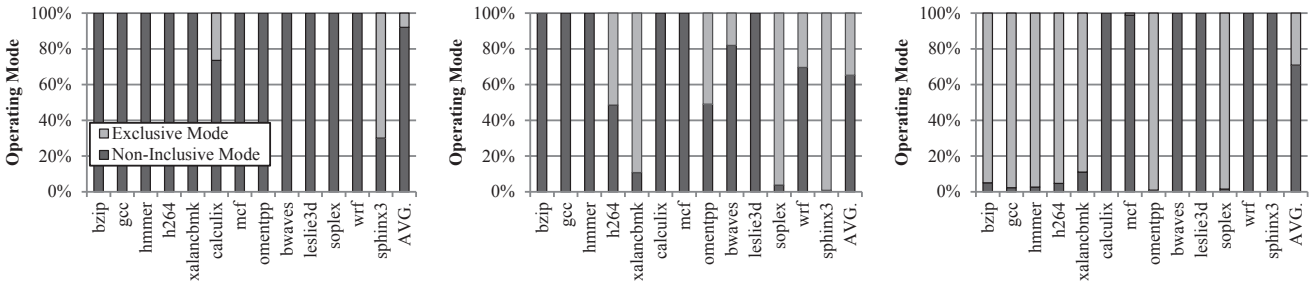
Figure 8(b) shows that, with the 4MB L3 cache, more benchmarks favor the non-inclusive cache since the maximum capacity that can benefit more from the exclusive cache is only 12.5%. Now, the benchmarks that previously favored the exclusive cache such as *hmmer* (0.997), *gcc* (1.006), and *bzip2* (1.001) have lost their advantage to run with the exclusive cache. Thus, these benchmarks actually consume more bandwidth compared to when they run with non-inclusive caches without achieving better performance.

5.2. FLEXclusion on Single-threaded Workloads

Figure 9 shows the performance of non-inclusion and FLEXclusion normalized to exclusion for L3 sizes from 1MB to 4MB. For each benchmark, there are three groups of bars, each of which corresponds to a different L3 cache size (1, 2, and 4MB). The left and right bars in each group show the performance of non-inclusion and FLEXclusion, respectively. Also, the bar labeled *AVG.* represents the geometric mean of normalized performance over all 13 benchmarks.

For benchmarks such as *bzip2*, *gcc*, *hmmer*, and *h264*, the non-inclusive cache has more than a 10% performance degradation compared to the exclusive cache in the 1MB cache size. In fact, *hmmer* shows a 20.1% performance degradation in the non-inclusive cache, but we can see that the FLEXclusive cache reduces the degradation to 1% compared to the exclusive cache, which leads to a 5.9% performance improvement over the non-inclusive cache on average. This performance loss in the non-inclusive cache vanishes as the cache size increases. However, the FLEXclusive cache always shows performance similar to that of the exclusive cache for all the different cache sizes. On average, the non-inclusive cache loses 6.3%, 3.0%, and 2.6% over the exclusive cache for the 1MB, 2MB, and 4MB sizes, while the FLEXclusive cache only loses 1.0%, 1.2%, and 1.6%, respectively.

Figure 10 shows the number of L3 insertions normalized to the non-inclusive L3 cache. The result shows that the exclusive L3 insertion is significantly higher, up to 34 times, than that in the non-inclusive L3 cache for most of the benchmarks. Interestingly, *hmmer* shows the L3 insertion traffic that is closest to that in the non-inclusive cache in all cases. This is because most of the L2 victim lines in the benchmark are dirty, so the victims are written back



(a) Group A+B in 1-Core (b) 2-MIX-S in 2-Cores (c) 4-MIX-S in 4-Cores

Figure 11. Percentage of the non-inclusive and exclusive mode used in FLEXclusion.

to the L3 cache in both non-inclusive and exclusive caches. In contrast, in *bwaves*, most of the L2 victims are clean as described in the previous section, which leads to 95.6%, 80.2%, and 96.3% reductions in the insertion traffic compared to the exclusive cache for the 1MB, 2MB, and 4MB cache sizes, respectively.

Figure 10 also shows that the FLEXclusive cache efficiently reduces the L3 insertion traffic. For example, in *bzip2*, in the 1MB cache, the performance difference is high, so the FLEXclusive cache pursues the performance by configuring the LLC as the exclusive mode at the cost of L3 insertion traffic. On the other hand, when the cache size is 2MB or 4MB, the performance of non-inclusion is comparable to that of exclusion. Thus, FLEXclusion configures the LLC as the non-inclusive mode to reduce the traffic. On average, FLEXclusion leads to 57.8% (1MB), 50.6% (2MB) and 72.6% (4MB) reductions in L3 insertion traffic.

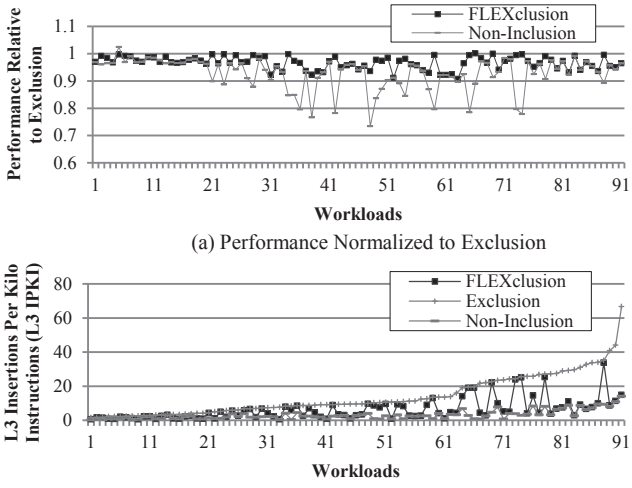
5.3. Adaptiveness on Effective Cache Size

FLEXclusion adaptively configures the L3 inclusion property, depending not only on the running workload, but also on the effective cache size that the individual workload experiences. For this purpose, we vary the number of threads from one to four on quad-core CMPs with the 512KB L2 and 4MB shared L3 caches. To observe the effective cache sizes only, we intentionally allocate the same benchmarks (2-MIX-S, 4-MIX-S).

As shown in Figure 11(a), when only one thread is enjoying the entire L3 cache, the non-inclusive mode is preferred for the L3 inclusion on most workloads. Only for a few benchmarks, such as *calculix* and *sphinx3*, does FLEXclusion set the L3 cache to the exclusive mode for some periods. However, as the number of competing threads increases, the L3 cache is configured as the exclusive mode for more workloads in order to increase the effective cache size for each individual thread, at the cost of on-chip traffic (Figure 11(b)). In fact, as shown in Figure 11(c), when all four cores compete for the shared L3 cache, seven out of 13 benchmarks need the L3 cache to behave as an exclusive cache for most of the running period.

5.4. FLEXclusion on Multi-programmed Workloads

In this section we evaluate 91 2-MIX-A multi-programmed workloads. Figure 12 shows the performance of non-inclusion and FLEXclusion relative to exclusion and the L3 IPKI values when 2-CPU mixed workloads are executed on the 4MB L3 cache. In each figure, the x-axis represents the mixed workloads (2-MIX-A) sorted by the value in exclusion L3 IPKI ascending order.



(a) Performance Normalized to Exclusion

(b) L3 Insertions Per Kilo Instructions (IPKI)

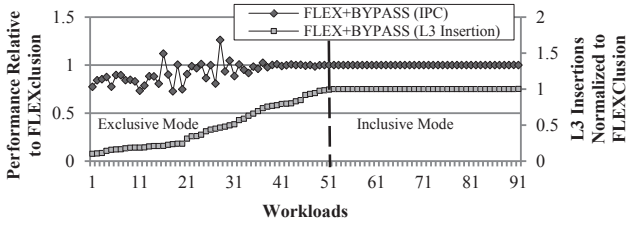
Figure 12. Normalized performance and L3 insertions per kilo instructions (L3 IPKI) on 2-CPU workloads (2-MIX-A).

The result shows that FLEXclusion recovers huge performance drops occurring in non-inclusion. Many workloads show more than 10% performance degradations in non-inclusion, but almost all FLEXclusion shows within a 5% range (Figure 12(a)). Furthermore, it reduces bandwidth consumption over exclusion when the performance difference is insignificant, thereby overcoming the disadvantages of static non-inclusive/exclusive LLCs. Exclusion shows high L3 IPKI in many workloads, but for the majority of the workloads FLEXclusion shows significantly low L3 IPKI and often similar to that in non-inclusion (Figure 12(b)). On average, FLEXclusion reduces L3 insertion traffic by 55.1% compared to the exclusive cache (with the cost of only 3.38% performance degradation over the exclusive cache) while achieving 4.78% more performance over the non-inclusive cache.

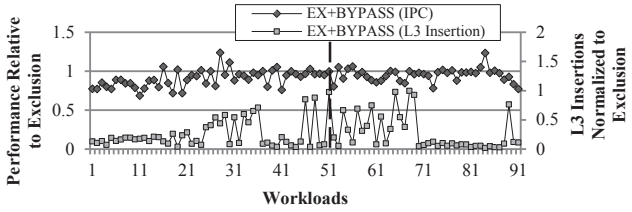
5.5. Extensions on FLEXclusion

All experiments in this section use the same experimental configuration (4MB L3 cache, 91 2-MIX-A) as in Section 5.4, but the workloads are sorted based on the normalized L3 insertions of each extension (BYPASS and AGG).

5.5.1. Bypass Algorithms with FLEXclusion (BYPASS) To show that other cache optimization techniques can be safely applied to the FLEXclusive cache, we implemented the *static version* of the bypass algorithm in [5] on top of the FLEXclusive cache. The basic idea of the bypass algorithm is to use the cache block's usage information (from L2) and trip counts (between L2 and L3) to find dead and live blocks for the bypass decision, which is realized with a small number of observer sets. In the FLEXclusive cache, the existing dedicated exclusive sets act for the observer sets. Only in the exclusive mode, the bypass decision is made on the cache blocks that access the FLEXclusive cache's following sets, just like the original bypass mechanism is applied to an exclusive cache.



(a) Bypass Algorithm on FLEXclusion



(b) Bypass Algorithm on Exclusion

Figure 13. Applying a bypass algorithm to FLEXclusion.

Figure 13 shows the performance and L3 insertions relative to FLEXclusion and exclusion when a bypass algorithm is applied on each. As Figure 13(a) shows, almost 50 workloads enter the exclusive mode in the FLEXclusive cache, so the bypass algorithm is applied. The results show that the bypass algorithm reduces bandwidth significantly, but it also sacrifices performance in the FLEXclusive cache. This is the same for applying the bypass algorithm to the baseline exclusive cache (Figure 13(b)). As expected, when the bypass improves performance in the baseline, it also improves performance in the FLEXclusive cache. For example, for workload 16 and 28, employing bypass increases performance by 12.0% and 26.2% in the FLEXclusive cache and by 5.7% and 23.5% in the exclusive cache, respectively. Hence, we can conclude that the negative or the benefit of bypass algorithms can be applied to the

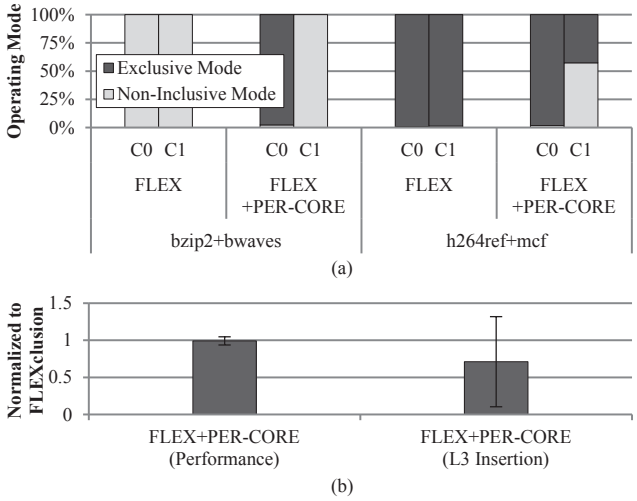


Figure 14. (a) Operating mode distribution for bzip2+bwaves and h264ref+mcf. C0 and C1 denote core-0 and core-1. (b) Average performance and L3 insertions with +/- standard deviation of FLEX+PER-CORE (normalized to FLEXclusion).

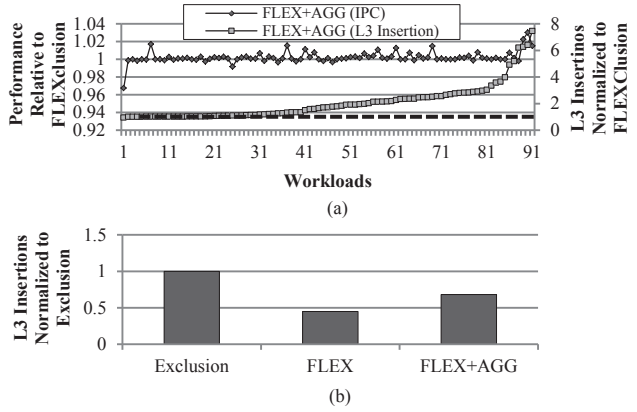


Figure 15. FLEXclusion with the aggressive mode.

FLEXclusive cache. Unfortunately, unlike FLEXclusion, a simple bypass algorithm saves bandwidth consumption at the cost of non-negligible performance loss. More complex bypass algorithms can overcome these issues, but again, the complex bypass algorithms can be also applied on top of the FLEXclusive cache. On average, FLEXclusion with the bypass algorithm reduces the L3 insertion traffic by 48.9% over FLEXclusion.

5.5.2. Per-Core Inclusion Policy (PER-CORE) To provide insight into the per-core policy, we show the operating mode distribution for two workload mixtures in Figure 14(a). As shown in Figure 11, bzip2 and h264ref favor the exclusive mode when multiple workloads are competing for the shared L3 cache, while bwaves and mcf mostly favor the non-inclusive mode. In the leftmost two bars (FLEX in bzip2+bwaves), we can see that L3 is set to the non-inclusive mode when bzip2 and bwaves are running on core-0 and core-1, respectively. However, the per-core policy (FLEX+PER-CORE) allows each core to have a different

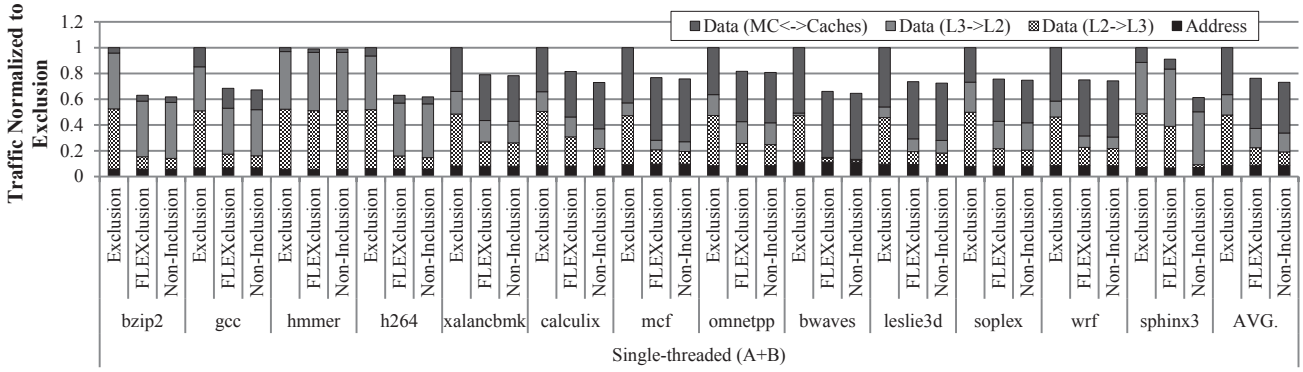


Figure 16. Traffic breakdown for single-threaded workloads (A+B).

operating mode that is the best for each workload, thereby leading to a 8.8% performance improvement over FLEXclusion. Similarly, for h264ref+mcf, FLEX+PER-CORE allows core-1 to operate in the non-inclusive mode, which reduces 28% of L3 insertions over FLEXclusion.

Without the per-core policy, the L3 cache is configured as the exclusive mode for many workload mixtures due to the decrease in effective cache space, which penalizes one of the workloads that can favor the non-inclusive mode. As shown in Figure 14(b), the per-core policy addresses this problem and reduces L3 insertion traffic over FLEXclusion with almost no performance degradation.

5.5.3. Aggressive Non-Inclusive Mode (AGG) Figure 15 shows the result when FLEXclusion tries to achieve performance more aggressively at the cost of more bandwidth consumption in the non-inclusive mode, especially when the bandwidth consumption is very low. For the evaluation, we use $Insertion_{th} = 20$ (IKPI) as a threshold. As a result, 56 out of 91 workloads improve performance as much as 3.02% in workload 90 (*gcc+bwaves*), as shown in Figure 15(a). Due to the aggressiveness, FLEX+AGG increases the number of L3 insertions as expected, but FLEX+AGG still reduces L3 insertion traffic by 31.9% over exclusion (Figure 15(b)).

5.6. Discussion

5.6.1. Considering Other Traffic We have so far considered the number of L3 insertions as a metric because (1) L3 insertions make a difference in traffic between non-inclusion and exclusion (Eq. (5)) and (2) the insertion traffic can be eliminated, unlike others, and thus is a target of FLEXclusion. To discuss the overall impact on traffic reduction in an on-chip network, we consider all traffic generated by cache and memory. Figure 16 shows the breakdown of traffic for single-threaded workloads (A+B).

As shown in the figure, L3 insertions take up a significant portion of all traffic in an exclusive cache, and FLEXclusion attempts to alleviate the insertion traffic wherever possible. Note that data messages between a memory controller (MC) and caches cannot be reduced since they are either memory requests or write-backs to memory. Data messages from L3 to L2 also cannot be eliminated because they are L3 hits, which must be provided into upper-level caches. Figure 16 also shows that the contribution of address messages to the total amount of traffic is less significant compared to data messages. This is because, although the number of address messages is large and comparable to data messages, these are short (1 flit) compared to data messages (9 flits).

5.6.2. Power Savings of FLEXclusion The traffic reduction provided by FLEXclusion is translated into the power savings of an on-chip network.[6] In on-chip interconnects, links and routers are the main contributors to energy consumption, and router energy is mostly consumed by FIFO buffers, crossbars, and arbiters [4, 15]. Most power consumption is due to the dynamic power dissipation of these components, which is directly related to the amount of traffic inside on-chip interconnects.

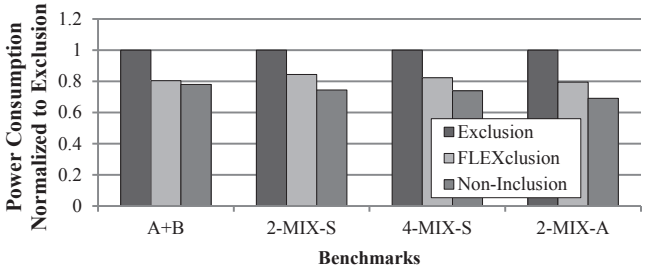


Figure 17. On-chip network power consumption.

Figure 17 shows the average power consumption of the evaluated on-chip network normalized to exclusion for different workload mixtures. In A+B, 2-MIX-S and 4-MIX-S, FLEXclusion reduces power consumption over exclusion by 19.6%, 15.6%, and 17.6%, respectively. Also, in 2-MIX-A, FLEXclusion reduces on-chip network power by 20.5%. We expect that FLEXclusion would play a more important role in the future as the number of cores increases and on-chip interconnects have more routers. For instance,

[6]The increase in power consumption to support FLEXclusion is very small. For set dueling, only 3% of cache sets are sampled, and a storage overhead is a few bytes. Compared to the dynamic power reduction in LLCs offered by FLEXclusion, register accesses due to set dueling are negligible. Hence, we can safely discuss power numbers only regarding on-chip networks.

the Raw multiprocessor [18] has a 4×4 mesh on-chip network, which results in consuming 7.1W (about 36% of the total chip power) [21]. However, the evaluation of FLEXclusion with various on-chip networks, such as with varying the number of nodes and different topologies, is beyond the scope of this paper.

5.6.3. Impact of FLEXclusion on Off-chip Traffic FLEXclusion increases off-chip traffic negligibly compared to exclusion since it reverts from non-inclusion to exclusion for the workloads where the difference in off-chip accesses between the two would increase significantly. Also, as shown in Figure 16 (Data (MC<->Caches) can represent the amount of off-chip traffic), the difference in off-chip traffic between FLEXclusion and exclusion is much smaller than the difference in on-chip traffic. For the reason, the slight increase in power expended due to extra off-chip accesses is very small compared to the on-chip power savings.

5.6.4. FLEXclusion on Multi-threaded Workloads With multi-threaded applications, FLEXclusion is expected to perform well. When workloads do not have much shared data, we expect that FLEXclusion has an impact on performance and traffic reduction similar to multi-programmed workloads since they resemble each other to some extent. For workloads where data is frequently shared among threads, the benefit of power reduction by FLEXclusion can be a bit decreased due to cache coherence messages. However, the energy consumption of coherence messages is reported to be around 18% of the total on-chip network [14], so FLEXclusion still can reduce the rest of the on-chip network power significantly. In addition, there are techniques to implement power-efficient cache coherence protocols for on-chip networks such as [4], and FLEXclusion is orthogonal to these techniques. Hence, FLEXclusion does not lose its importance with multi-threaded workloads.

5.6.5. Sensitivity of FLEXclusion Figure 18 shows the relationship of performance loss and L3 insertion reduction among different performance thresholds. The results are the average of 91 workloads. As the results show, we can flexibly balance performance and bandwidth requirements by controlling the threshold. In this paper, we used T=5 for all experiments. However, when on-chip bandwidth requirements are less strict, we can use smaller threshold values to alleviate the performance loss.

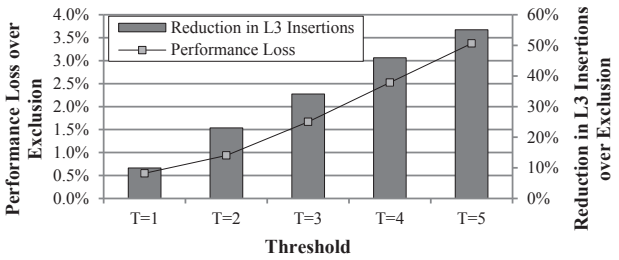


Figure 18. Performance loss and L3 insertion reduction of different thresholds (T=1 is a 1% difference in cache miss ratios between exclusive and non-inclusive sets).

6. Related Work

6.1. Cache Inclusion Property

Baer and Wang studied the concept of multi-level inclusive cache hierarchies to simplify cache coherence protocols [3]. Zahran et al. discussed several non-inclusion cache design strategies along with the benefits of non-inclusive caches [24]. The performance benefit of exclusive caches over inclusive caches has been studied by several researchers, including Zheng et al. [12, 25]. All these works provide the benefits and disadvantages of exclusive, inclusive, and non-inclusive caches, but they assume that the cache inclusion policy is statically determined at design time. However, in our FLEXclusion, we break this assumption and dynamically configure caches between exclusion and non-inclusion at run-time.

The most relevant work to FLEXclusion is the recently proposed temporal locality-aware (TLA) cache management policy by Jaleel et al. [8]. The goal of the TLA mechanism begins from the opposite direction of FLEXclusion. FLEXclusion aims to achieve the best of both exclusive cache and non-inclusive cache, while TLA targets the best of both non-inclusive and inclusive caches. TLA tries to achieve performance that is similar to non-inclusive cache without breaking the inclusion property. However, TLA still cannot achieve performance similar to exclusive caches. On the contrary, the FLEXclusive cache achieves almost the same performance as that of an exclusive cache while saving on-chip bandwidth consumption significantly by operating as non-inclusive mode when it is more efficient.

6.2. Bypassing LLC Insertions

Including the recent work of Gaur et al. [5], several bypass algorithms with dead block predictions have been proposed [11,13,19]. Although bypass mechanisms and FLEXclusion share similar actions (i.e., both schemes do not insert some blocks into the LLC), the cause and performance impact are actually very different. Bypass algorithms do not insert a block when the block is predicted to be dead. On the contrary, FLEXclusion does not insert a block to the LLC because the block is already in the LLC (non-inclusive mode).[7] Hence, in bypass mechanisms, bypassed blocks could result in a performance loss if the prediction is wrong (i.e., the block is live). However, bypassed blocks in FLEXclusion do not cause a performance degradation because the blocks are still in the LLC. Furthermore, FLEXclusion can always employ bypass mechanisms on the exclusive mode, as shown in Section 5.5.1.

[7] Since the non-inclusive cache does not strictly enforce inclusion, the block might not exist in the LLC, but this happens less frequently.

7. Conclusions

While exclusive last-level caches (LLCs) are effective at reducing off-chip memory accesses by fully utilizing on-chip cache capacity, exclusive caches require higher on-chip bandwidth compared to inclusive and non-inclusive caches due to a higher rate of LLC insertion traffic. A non-inclusive cache, on the other hand, does not require clean victims from the upper level caches to be inserted in the LLC, thus reducing the demand on an on-chip network. Unfortunately, the data replication in non-inclusive caches causes them to have lower effective cache capacity, thereby reducing performance when the workload needs more cache capacity.

This paper investigated a dynamic mechanism called FLEXclusion that can change the cache organization between exclusion and non-inclusion depending on the workload requirement. Our evaluation shows that the FLEXclusive cache reduces the LLC insertion traffic compared to the static exclusive LLC design, thereby saving power. The FLEXclusive cache also improves performance compared to the static non-inclusive LLC cache. We also show that the FLEXclusive cache can employ other cache optimization techniques such as bypassing mechanisms to achieve further benefits.

In this paper, we restricted FLEXclusion to choose between non-inclusion and exclusion, as these two designs have similar coherence framework. An alternative FLEXclusion design can also select between inclusion and other modes; however, it would need to ensure that the inclusion requirements are met before the cache mode is set to inclusion (for example by flushing the upper level caches). Thus, a generalized form of FLEXclusion can be designed to adapt between inclusion, non-inclusion, and exclusion depending on the workload requirements. Exploring such a generalized design is part of our future work.

Acknowledgments

Thanks to Viji Srinivasan for discussions during the early phase of this work. Many thanks to Si Li, other CASL members, Nagesh B. Lakshminarayana, HPArch members and the anonymous reviewers for their suggestions and feedback. We gratefully acknowledge the support of the NSF CAREER award 1139083; Sandia National Laboratories; Intel Corporation; Advanced Micro Devices. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, SNL, Intel, or AMD.

References

[1] AMD Phenom$^{(TM)}$ II processor model. http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages.phenon-ii-key-architectural-features.aspx.

[2] Macsim simulator. http://code.google.com/p/macsim/.

[3] J.-L. Baer and W.-H. Wang. On the inclusion properties for multi-level cache hierarchies. In *ISCA-10*, pages 73–80, 1988.

[4] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *ISCA-28*, 2006.

[5] J. Gaur, M. Chaudhuri, and S. Subramoney. Bypass and insertion algorithms for exclusive last-level caches. In *ISCA-33*, 2011.

[6] INTEL. Intel Core i7 Processors. http://www.intel.com/products/processor/corei7/specifications.htm.

[7] Intel. Intel®Nehalem Microarchitecture. http://www.intel.com/technology/architecture-silicon/next-gen/index.htm?iid=tech_micro+nehalem.

[8] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr., and J. Emer. Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies. In *MICRO-43*, MICRO '43, 2010.

[9] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, Jr., and J. Emer. Adaptive insertion policies for managing shared caches. In *PACT-17*, 2008.

[10] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer. High performance cache replacement using re-reference interval prediction (RRIP). In *ISCA-32*, 2010.

[11] T. L. Johnson and W. mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *ISCA-19*, pages 315–326, 1997.

[12] N. P. Jouppi and S. J. E. Wilton. Tradeoffs in two-level on-chip caching. In *ISCA-16*, 1994.

[13] M. Kharbutli and Y. Solihin. Counter-based cache replacement and bypassing algorithms. *IEEE Trans. Computers*, 57(4):433–447, 2008.

[14] H. Kim, P. Ghoshal, B. Grot, P. V. Gratz, and D. A. Jiménez. Reducing network-on-chip energy consumption through spatial locality speculation. In *NOCS-5*, pages 233–240, 2011.

[15] A. K. Mishra, N. Vijaykrishnan, and C. R. Das. A case for heterogeneous on-chip interconnects for cmps. In *ISCA-33*, pages 389–400, 2011.

[16] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *ISCA-29*, 2007.

[17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS-X*, 2002.

[18] M. B. Taylor et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22:25–35, March 2002.

[19] G. Tyson, M. Farrens, J. Matthews, and A. Pleszkun. A new approach to cache management. In *MICRO-28*, 1995.

[20] VIA. VIA C7 Processors. http://www.via.com.tw/en/products/processors/c7/.

[21] H. Wang, L.-S. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *MICRO-36*, 2003.

[22] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *MICRO-35*, 2002.

[23] Y. Xie and G. H. Loh. PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches. In *ISCA-31*, 2009.

[24] M. M. Zahran, K. Albayraktaroglu, and M. Franklin. Non-inclusion property in multi-level caches revisited. *International Journal of Computers and Their Applications*, 14:99–108, 2007.

[25] Y. Zheng, B. T. Davis, and M. Jordan. Performance evaluation of exclusive cache hierarchies. In *ISPASS'04*, 2004.