creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

캐시 분할 기술을 이용한 공유 라스트 레벨 캐시의 분할 정책에 따른 성능 민감도 연구

2018 년 2 월

서울대학교 융합과학기술대학원

융합과학부 지능형융합시스템전공

서 슬 기

초 록

멀티 코어 시스템에서 여러 어플리케이션들을 동시에 실행할 때, 시스템 공유 자원(공유 캐시, 메인 메모리 등)에서 발생하는 경합/간섭은 일부 또는 모든 어플리케이션의 성능 저하를 유발할 수 있으며, 이를 양적으로 예측하는 것은 매우 어렵다. 특히, 여러 어플리케이션들이 한정된 공유 캐시 용량을 경합을 통해 나누어 사용하다 보면, 실시간 어플리케이션 또는 실행 시간이 중요한 어플리케이션이 다른 어플리케이션에 의해 캐시 점유율을 과도하게 빼앗겨 심각한 성능 저하를 겪을 수 있다. 이러한 공유 캐시에서 발생할 수 있는 부정적인 상황을 방지하기 위해서, 우선순위가 높은 어플리케이션에 적정 수준의 캐시 용량을 독립적으로 할당하는 방법을 사용할 수 있다. 이러한 방법은 실제 제품에 적용되기 전부터 광범위하게 연구되고 실험되어왔다.

인텔은 제온 프로세서 v3 제품 군부터 어플리케이션마다 공유 라스트 레벨 캐시를 분할/할당할 수 있는 Cache Allocation Technology(CAT) 기술을 적용하였다. 공유 캐시의 분할은 동일한 우선순위를 갖는 어플리케이션들의 집합 단위로 이루어진다. 캐시 분할 방법에서는 독립 분할과 중첩 분할 방식을 제공한다. 중첩 분할을 사용하면 우선순위가 높은 어플리케이션에 모든 캐시 영역을 할당할 수 있기 때문에 독립 분할을 사용할 때 보다 우선순위가 높은 어플리케이션의 성능을 최대화하는 데 유리할 수 있다. 그러나 이러한 직관적인 예상과는 반대 경향을 보일 가능성 또한 존재한다.

본 연구는 특정 어플리케이션의 성능을 최대화하기 위해 CAT를 사용할 때, 독립 분할과 중첩 분할의 성능을 하드웨어 실험을 통하여

비교하고 분석 프로그램과 시뮬레이션으로 그 원인을 파악하였다. 20 개 조합(어플리케이션 쌍) 중 14개 조합에서 독립 분할이 성능 우위(~12%)를 보였으며, 나머지 조합에서는 중첩 분할이 성능 우위(~16%)를 보였다. 독립 분할이 성능 우위를 보이는 경우는 중첩 분할 시 공유 영역에서 어플리케이션 간의 경쟁으로 인한 캐시 미스가 과도하게 발생하여 성능이 저하되는 것을 확인하였다. 시뮬레이션을 통해 이를 재현하였으며 캐시 미스가 증가한 것은 중첩 분할 시 캐시 교체 정책(예를 들어, least recently used 정책)을 제대로 적용할 수 없기 때문인 것을 확인하였다.

주요어: 공유 라스트 레벨 캐시, 독립 분할, 중첩 분할, 캐시 교체정책
학　번: 2016-26036

목 차

제 1 장 서 론 1
제 1 절 연구동기 1
제 2 절 관련연구 및 배경 3
제 3 절 연구내용 7

제 2 장 독립 분할과 중첩 분할 성능 비교 9
제 1 절 독립 분할과 중첩 분할 설정 9
제 2 절 하드웨어 실험 환경 및 설정 10
제 3 절 어플레케이션 단독 수행 결과 12
제 4 절 독립 분할과 중철 분할 성능 결과 비교 분석 13

제 3 장 중첩 분할 성능 열화 원인 분석 20
제 1 절 캐시 교환 정책 및 중첩 분할 성능 연관성 20
제 2 절 시뮬레이션 실험 환경 및 설정 21
제 3 절 검증 실험 결과 22

제 4 장 결론 27
참고 문헌 28
Abstract 28

표 목 차

표 1. 실험에서 사용한 하드웨어 시스템 구성 10
표 2. 독립 분할과 중첩 분할 시 RELATIVE IPC 14
표 3. 실험에서 사용한 시뮬레이션 시스템 구성 22

그 림 목 차

그림 1. 공유 캐시 분할/할당 정책 미적용(좌),적용(우) 1
그림 2. CAT의 독립 분할(좌)과 중첩 분할(우) 2
그림 3. HIGH PRIORITY APPLICATION의 성능 영향성[3] 4
그림 4. APPLICATION, TRHEAD 혹은 VM 별로 CLOS할당..... 5
그림 5. CLOS BITMASK 설정: ISOATED와 OVERLAPPED 분할..... 6
그림 6. 독립 분할(좌)와 중첩 분할(우) 설정에 따른 캐시 점유율..... 9
그림 7. 단일 어플리케이션의 캐시 민감도..... 12
그림 8. 독립 분할과 중첩 분할 시 어플리케이션 A의 IPC 차이 (RELATIVE IPC는 중첩 분할 시 IPC를 기준(1)으로 함) 13
그림 9. 각 어플리게이션 조합의 IPC, MPKI 변화..... 15
그림 10. 독립 분할이 우위인 조합(a), 중첩 분할이 우위인 조합(b)의 어플리케이션 A의 IPC 및 LLC MPKI..... 18
그림 11. 중첩 분할에서 캐시 교체 예시..... 20
그림 12. 단일 어플리케이션의 캐시 민감도 비교..... 23
그림 13. 시뮬레이션 재현 및 하드웨어 실험과 비교..... 24
그림 14. 시뮬레이션에서 독립 분할이 우위인 조합(a), 중첩 분할이 우위인 조합(b)의 어플리케이션 A의 IPC 및 LLC MPKI 25

제 1 장 서 론

제 1 절 연구동기

여러 어플리케이션들이 멀티 코어 시스템에서 한정된 공유 자원(공유 캐시, 메인 메모리 등)을 사용하는 경우, 해당 공유 자원에서 발생하는 어플리케이션간 경합/간섭으로 인해 일부 또는 모든 어플리케이션의 성능 저하를 유발할 수 있다. 캐시 히트율이 높은 실시간(real-time) 또는 QoS(Qaulity of Service)가 보장되어야 하는 우선순위가 높은 어플리케이션에 경우, 공유 캐시 내에서 발생하는 경합은 다른 어플리케이션에 의해 캐시 점유율(occupancy)을 과도하게 빼앗겨 심각한 성능 저하를 겪을 수 있다.

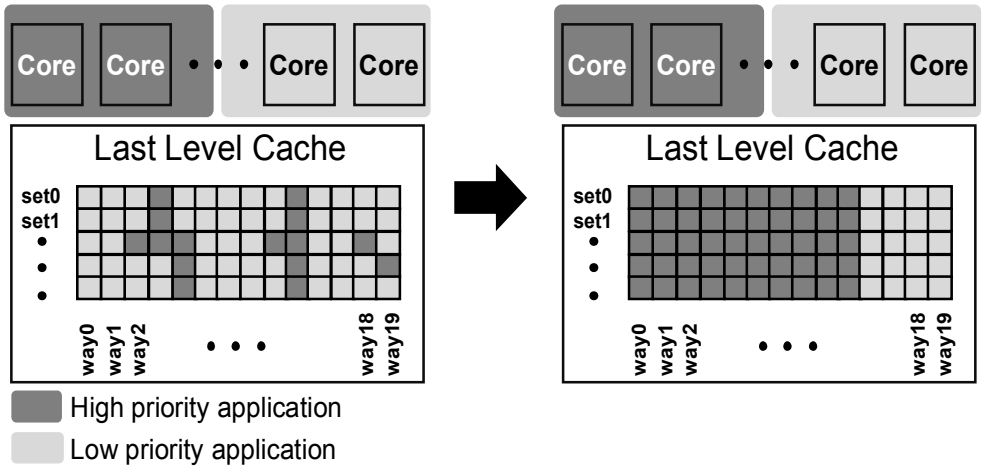


그림 1. 공유 캐시 분할/할당 정책 미적용(좌),적용(우)

공유 캐시에서 발생할 수 있는 이러한 부정적인 상황을 방지하기 위해서, 우선순위가 높은 어플리케이션에 적정 수준의 캐시 용량을 독립적으로 할당하는 방법을 사용할 수 있다[4, 5]. 그림 1은

우선순위가 높은 어플리케이션(High priority application)이 공유 라스트 레벨 캐시(LLC: Last Level Cache)에서 경합으로 인해 적은 용량을 사용하는 상황(좌)과 공유 캐시 분할/할당 정책을 적용하여 LLC에서 많은 용량을 사용하는 상황(우)을 보여준다. 관련 연구 및 방법은 실제 제품에 적용되기 전부터 광범위하게 연구되고 실험 되어왔다[1, 2, 3].

인텔은 제온 프로세서 v3 제품군(Haswell-EP)부터 어플리케이션마다 공유 LLC를 분할/할당할 수 있는 기술인 CAT (Cache Allocation Technology)를 적용하여 공유 LLC에서 발생하는 문제를 해결하고자 하였다.[4]. 공유 LLC의 분할은 동일한 우선순위를 갖는 어플리케이션들의 집합인 CLOS (Classes of Service) 단위로 이루어진다. 즉, 여러 개의 어플리케이션은 사용자가 지정한 우선순위에 따라 복수의 CLOS로 분류되고 각 CLOS에 사용자가 지정한 LLC의 용량이 할당된다. 용량을 할당하는 방법은 CLOS가 사용할 수 있는 way의 수를 지정하는 것이다.

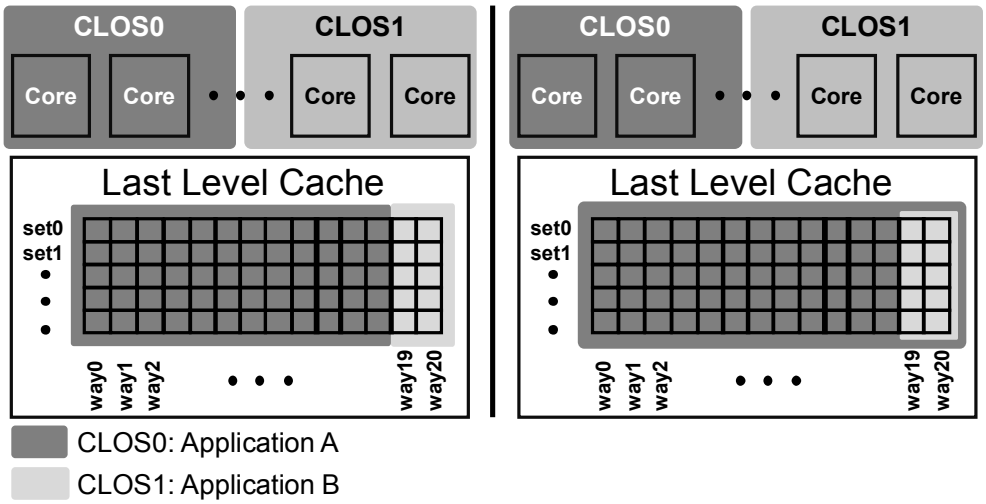


그림 2. CAT의 독립 분할(좌)과 중첩 분할(우)

중요한 점은 인텔 CAT은 각 CLOS에 way를 할당하는 방법에 있어서 두 가지 옵션을 제공한다. 첫째는 '독립(isolated) 분할'로서 그림 2의 좌측 그림과 같이 각 CLOS에 독점적으로 사용할 수 있는 way들 할당하는 방법이다. 둘째는 '중첩(overlapped) 분할'로서 그림 2의 우측 그림과 같이 특정 CLOS만 사용할 수 있는 way와 여러 CLOS가 공유할 수 있는 way를 같이 설정하는 방법이다[4].

중첩 분할 방법을 사용하면 우선순위가 높은 CLOS에 모든 캐시 영역을 할당할 수 있기 때문에(일부는 독점적으로 사용, 일부는 다른 CLOS와 공유해서 사용) 독립 분할 방법을 사용할 때 보다 우선순위가 높은 어플리케이션의 성능을 최대화하는 데 유리할 수 있다. 그러나 이러한 직관적인 예상과는 반대 경향을 보일 가능성 또한 존재한다. 예를 들어, 다른 CLOS와 공유하고 있는 캐시 영역에서 발생하는 contention miss[6]로 인해 독립 분할 방법 사용보다 오히려 어플리케이션 성능이 나빠질 수 있다. 본 연구는 특정 어플리케이션의 성능을 최대화하기 위해 CAT을 사용할 때, 독립 분할과 중첩 분할의 성능을 하드웨어 실험을 통하여 비교하고 시뮬레이션으로 그 원인을 분석 및 개선 사항에 대해서 연구를 진행하였다.

제 2 절 관련연구 및 배경

캐시 분할방법을 이용하여 어플리케이션의 성능을 보장하기 위한 연구는 오랜 시간 동안 진행되어 왔으며 시뮬레이션 등을 통해서 각 연구에서 제안하는 방법들이 우위를 보였다. 이는 우선순위가 높은 어플리케이션에 전체 공유 캐시의 일부를 세트(set), 웨이(way) 혹은 캐시 라인(cache line)단위로 할당함으로써 해결할 수 있다. 이와

관련한 내용은 캐시 분할 기법에 대한 논문들을 정리한 논문을 통해서 캐시 분할 기법의 종류, 목적과 실험 방법 등을 확인할 수 있다[11]. 캐시 분할 기법을 적용하는 것 외에 데이터 센터와 같은 여러 서버가 존재하는 환경에서는 동일 서버에서 우선순위가 높은 어플리케이션이 동작 시 다른 어플리케이션을 배치하지 않는 방식을 적용할 수도 있다[5].

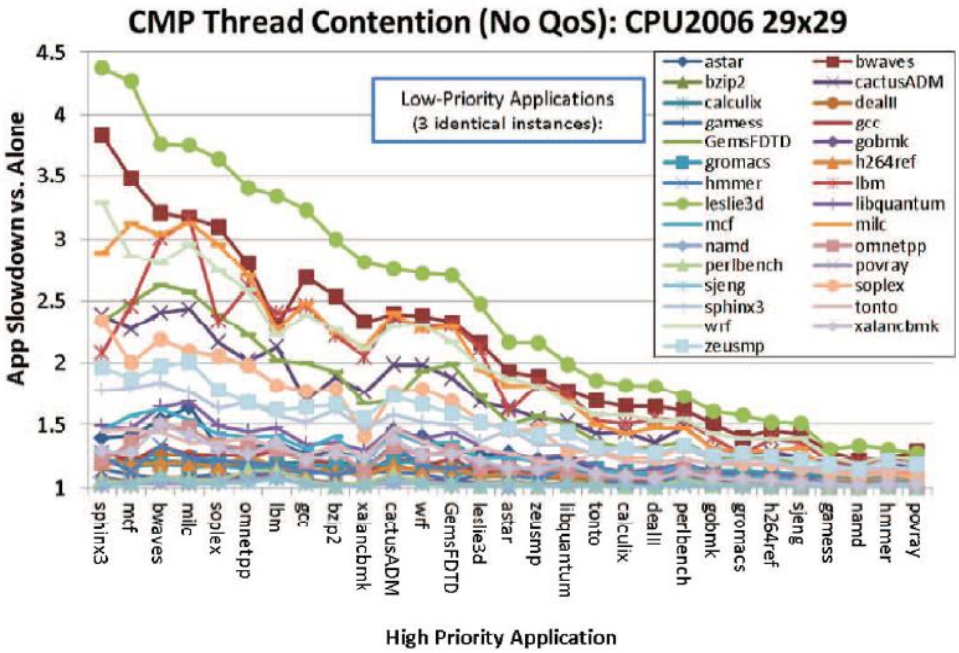


그림 3. High priority application의 성능 영향성[4]

멀티 코어 플랫폼에서 여러 어플리케이션이 동시에 실행될 때 공유 캐시 내에서 경합의 영향은 각 어플리케이션의 공유 자원 소비에 따라 달라진다. 그림 3은 인텔 제온 프로세서에서 SPEC CPU2006[12] 중 하나를 우선순위가 높은 어플리케이션으로 정했을 때, 다른 하나의 우선순위가 낮은 어플리케이션에 대한 성능 영향을 보여주고 있다. 단독으로 동작했을 때 대비 수행 시간(execution time)을 비교한 것이며 공유 캐시 내에서의 경합으로 인해 최대 4.4배 수행시간이 늘어나게 된다[4]. 해당 공유 캐시의 경합에 대한 문제를 해결하기 위해

인텔에서는 제온 프로세서 v3부터 CAT(Cache Allocation Technology)와 CMT(Cache Morniotoring Technology)를 적용하여 해당 문제에 대한 해결책을 해결하고자 하였다[4, 8]. CAT와 CMT는 각 어플리케이션(혹은 스레드, 가상 머신)별로 공유 라스트 레벨 캐시 내에서 캐시 할당 제어 및 모니터링을 할 수 있는 기술이다.

CAT를 통해 공유 라스트 레벨 캐시 내에 캐시 분할/할당은 CLOS(class of service)별로 가능하며 각 CLOS는 그림 4와 같이 단일 어플리케이션(혹은 스레드, 가상 머신) 혹은 여러 개로 구성할 수 있다. 어플리케이션의 독점적 사용과 우선 순위 조절이 가능하며, 이를 통해서 LLC 접근이 많고 LLC 재 사용율 (LL3 hit)이 적은 특정 어플리케이션으로 인한 다른 어플리케이션들의 성능 저하를 막을 수 있다.

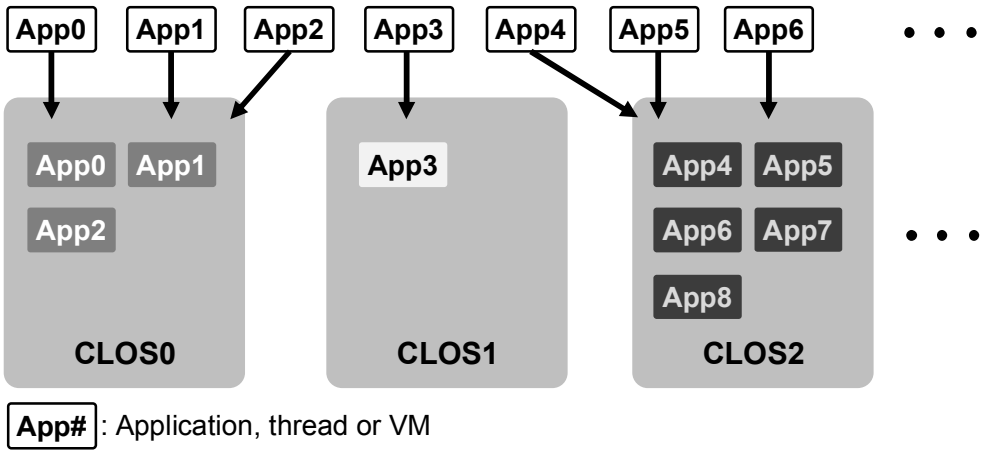


그림 4. Application, trhead 혹은 VM 별로 CLOS할당

CAT는 각 CLOS에 LLC의 way를 할당하는 방식이며 그림 5와 같이 LLC의 way들은 bitmask와 연결되고, 이 bitmask를 통해 CLOS의 LLC의 way 사용 할당 비율을 지정한다(bitmask의 각 bit는 LLC의 1 way에 해당). 각 way는 LLC의 독립적인 일부 용량을 나타낸다. 예로

총 20 way를 가진 50MB LLC에서는 각 1bit는 2560kB/1way를 나타낸다. 그리고 해당 캐시 할당 설정은 run-time에 동적으로 변경할 수 있으며, CMT의 관찰 정보를 활용하여 캐시 분할 및 할당을 변경할 수 있다.

CLOS는 독립, 중첩으로 masks를 구성할 수 있다. 그림 2와 5에서

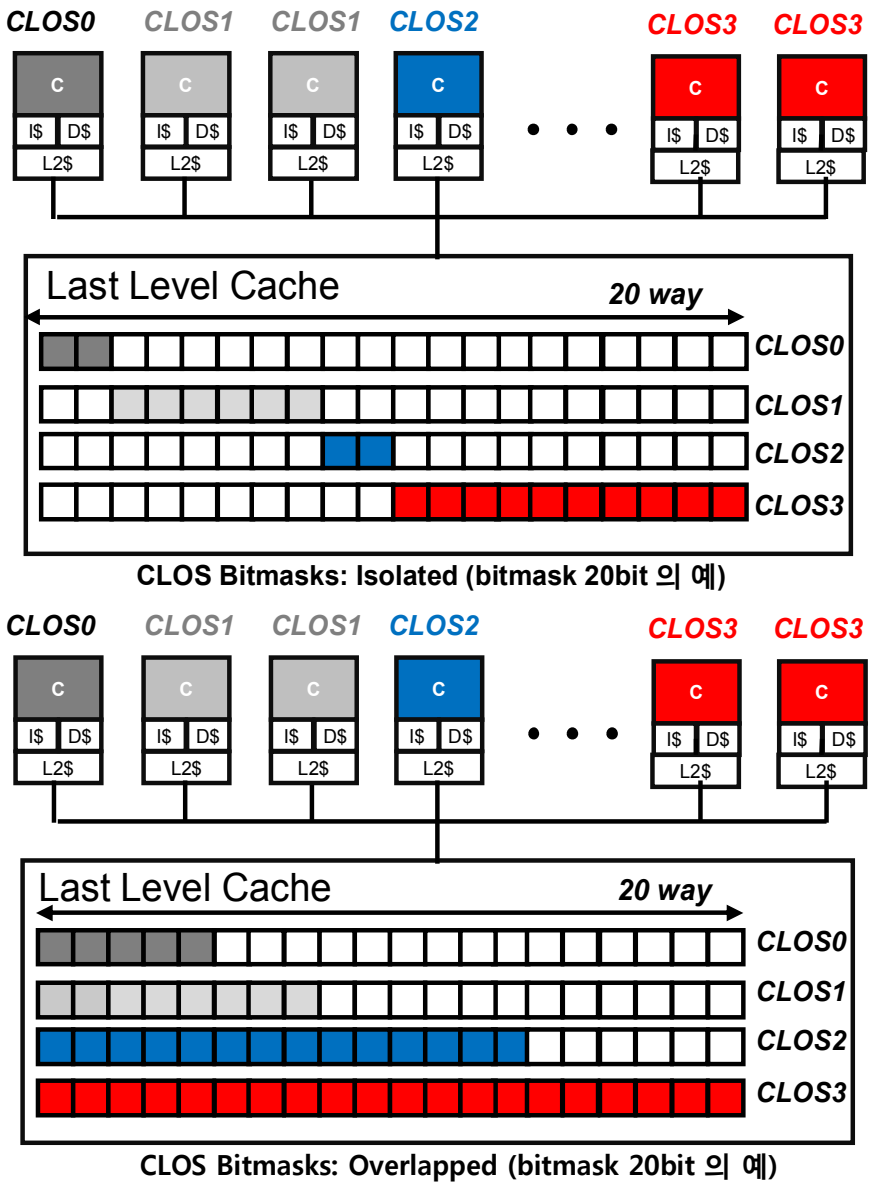


그림 5. CLOS bitmask 설정: Isolated와 Overlapped 분할

보이는 설정처럼 특정 어플리케이션에 더 많은 캐시를 다른 어플리케이션과 독립적으로 할당하거나 여러 어플리케이션들과 중첩되게 설정할 수 있다. 즉, 특정 어플리케이션이 전체 캐시를 사용하되 다른 어플리케이션과 일부를 공유하는 하는 방법과 각 어플리케이션이 캐시 일부만을 독립 사용하는 방법이 가능하다.

앞선 연구들에서는 캐시의 독립적으로 사용하는 것에 집중하였으며 중첩하여 사용하는 경우에 대한 장점에 대해서는 크게 논의되지 않았다[4].

CMT를 통해서 단일 혹은 복수 개의 어플리케이션(혹은 스레드, 가상 머신)에 대한 LLC 점유율(LLC cache occupancy) 관찰이 가능하며, 제온 프로세서 v3 (Haswell-EP) 다음 v4 (Broadwell-EP) 마이크로아키텍처부터는 Memory Bandwidth Monitoring (MBM)이 추가되어 LLC에서 외부(동일 소켓과 다른 소켓 모두)로 전달되는 메모리 대역폭(LLC total external bandwidth)과 LLC에서 내부 소켓으로 전달되는 메모리 대역폭(LLC local external bandwidth)의 측정이 가능하다. 이를 통해 외부로 전달되는 메모리 대역폭과 동일 소켓으로 전달되는 메모리 대역폭의 차이를 외부 소켓으로 전달되는 메모리 대역폭으로 정의할 수 있다(LLC total external bandwidth – LLC local external bandwidth).

제 3 절 연구내용

본 논문은 캐시의 독립, 중첩 분할에 따른 특정 어플리케션의 성능에 대해서 비교 및 분석을 하드웨어 실험을 통하여 진행하였으며, 문제 상황에 대해서는 시뮬레이션 실험을 통해 재현 및 성능 개선을

확인하였다. SPEC CPU2006에서 캐시 용량에 따라 성능의 변화를 보이는 어플리케이션들을 관련 논문[7] 및 실험을 통해 선별하였으며 해당 어플리케이션의 조합에 대해서 하드웨어와 시뮬레이션 실험을 통해 비교 및 분석하였다.

본 논문의 2장에서는 최신 인텔 제온 프로세서 v4(Broadwell-EP)에서 독립과 분할 할당에 따른 성능 비교를 분석하였고, 3장에서는 분석된 결과를 바탕으로 시뮬레이션을 통해 그 원인에 대한 근거를 제시하였으며, 이것에 대한 분석 및 개선 사항에 대해서 McSimA+[9]를 이용하여 시뮬레이션을 진행하였다. 하드웨어 및 시뮬레이션 실험 환경 및 방법에는 각 2, 3장에서 정리하였고, 4장에서는 본 논문의 결론을 정리하였다.

제 2 장 독립 분할과 중첩 분할 성능 비교

제 1 절 독립 분할과 중첩 분할 설정

하드웨어 실험 목적은 CAT를 사용하여 특정 어플리케이션의 성능을 최대화 시킬 때, 독립 분할과 중첩 분할 중 효과적인 방법을 확인하는 것이다. 독립 분할 설정 시 각 어플리케이션들의 사용 캐시 공간, 즉 LLC의 way를 독점적으로 할당 받게 하였다. 중첩 분할 설정 시에는 특정 어플리케이션은 전체 LLC way를 사용하게 하고 다른 어플리케이션은 전체 way 중 할당 받은 일부 LLC way를 공유하도록 하였다. 해당 중첩 분할 방식은 일부 영역만을 공유하여 사용하는 방식보다 특정 어플리케이션이 전체 LLC를 사용하여 성능을 극대화할 수 있는 방법이다.

중첩 분할 방법을 적용할 경우, 그림 6에서 보이는 것 같이 일부 공유 영역에서는 contention miss가 발생할 수 있다. Contention miss는

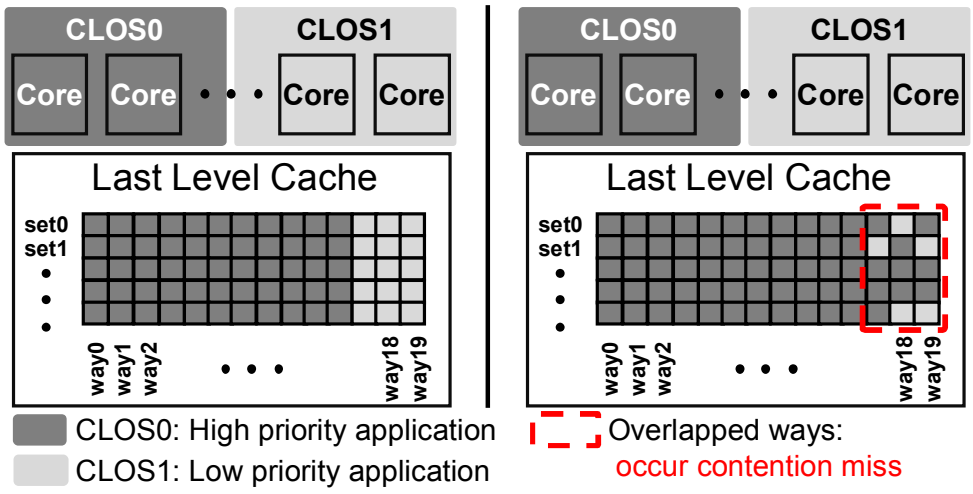


그림 6. 독립 분할(좌)와 중첩 분할(우) 설정에 따른 캐시 점유율

캐싱 되어 있던 데이터가 타 어플리케이션에 의해 제거(eviction)되어 발생하는 캐시 미스이며, 과도한 contention miss로 인해 더 많은 캐시를 사용하지만 독립 분할 방식을 사용하여 적은 way를 독립적으로 사용하는 경우보다 성능이 나빠질 수 있다.

제 2 절 하드웨어 실험 환경 및 설정

processor mode	E5-2698 v4 (Broadwell)
# of physical cores	20 core
Clock Speed	2.2GHz (turbo boost off)
logical cores per physical core	2 (hyper-threading on)
LLC size/way	50MB/20-way
Memory type	DDR4 2400 2 rank/channel
Memory size	64GB
Memory channel	4
OS	Ubuntu 16.04
Kernel version	4.4.0
gcc version	5.4.0

표 1. 실험에서 사용한 하드웨어 시스템 구성

하드웨어 실험에 사용한 시스템은 표 1과 같다. 프로세서(Intel E5-2698 v4)는 20개의 물리적 코어(physical core)와 50MB 공유 LLC (20 way)를 가지고 있다. 하이퍼스레딩(Hyper-theading) 기능을 설정하여 논리적 코어(logical core) 총 40개를 사용하였다. 그림 2와 같이, 어플리케이션 A의 20 copy를 20개의 논리적 코어로 이루어진 CLOS0에서 실행하였고, 어플리케이션 B의 20 copy를 CLOS1 (20개의 나머지 논리적 코어)에서 실행하였다.

실험은 CLOS0에 속한 어플리케이션 A의 성능 변화를 확인하는 데 초점을 맞추었다. 독립 분할 실험에서는 어플리케이션 A (CLOS0)에 할당한 way의 수를 2 way에서 1 way 씩 18 way까지 늘려가는 동시에, 어플리케이션 B (CLOS1)에 할당한 way의 수는 줄여가며 18 way에서 1 way 씩 2 way까지 줄여가며 성능을 측정하였다. 중첩 분할 실험에서는 어플리케이션 A (CLOS0)에 고정적으로 20 way 전체를 할당하고, 어플리케이션 B (CLOS1)에 할당한 way의 수는 18 way에서 1 way 씩 2 way까지 줄여가면 성능을 측정하였다.

해당 실험의 CAT 설정과 CMT 모니터링을 위해서 Intel에서 제공하는 오픈 소스 프로그램 사용과 메뉴얼을 참고하였다[10, 14]. 해당 프로그램을 통해서 어플리케이션이 동작하는 논리적 코어별 CLOS 할당이 가능하며, IPC(Instruction Per Cycle), LLC miss, LLC occupancy, LLC external과 internal bandwidth 측정이 가능하다. 각 실험의 결과는 어플리케이션 별 IPC (논리적 코어당 평균 IPC)를 측정하여 성능을 확인하였고, 추가적인 분석을 위해 오픈 소스의 코드를 수정하여 어플리케이션 별 LLC MPKI (Miss Per Kilo Instruction, 논리적 코어당 평균 MPKI)를 측정하였다.

실험에서는 캐시 사이즈에 성능이 민감한 어플리케이션들이 사용되었으며 스트림(stream) 벤치마크와 같이 캐시 사이즈 변화에 따라 성능의 차이가 없는 어플리케이션들은 실험에서 제외하였다. 이유는 캐시 사이즈에 따라 성능 변화가 없는 어플리케이션인 경우에는 사용 캐시 사이즈를 최소화하는 것으로 문제를 해결할 수 있기 때문이다. 어플리케이션의 단독 수행 결과는 결과는 제2장 2절 어플리케이션 단독 수행 결과에서 추가해 두었다.

제 3 절 어플레케이션 단독 수행 결과

SPEC CPU2006 벤치마크 중 캐시 용량에 민감한 어플리케이션 5개인 bzip2, omntepp, soplex, leslie3d, sphinx3를 기존 연구[7] 및 실험을 통해 선정하였다.

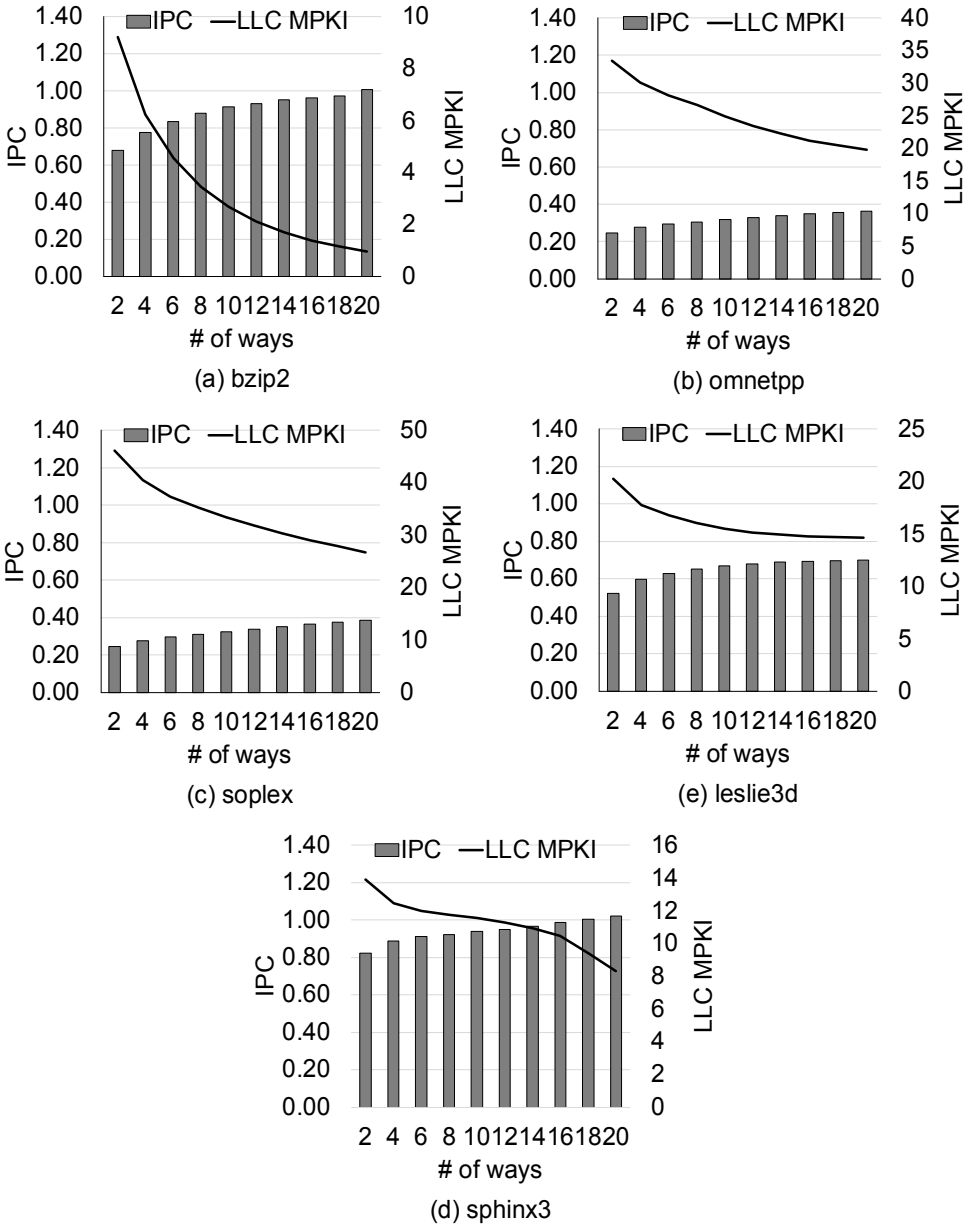


그림 7. 단일 어플리케이션의 캐시 민감도

그림 7은 단일 어플리케이션의 캐시 민감도를 보여주고 있다. 20 코어를 사용하여 평균 코어별 IPC와 MPKI를 측정하였다. 그 결과는 기존 연구과 유사한 캐시 용량에 따른 성능 변화를 보였으며 이 5개의 어플리케이션으로 20개의 어플리케이션 조합(A-B 쌍)을 만들어 실험에 사용하였다.

제 4 절 독립 분할과 중철 분할 성능 결과 비교 분석

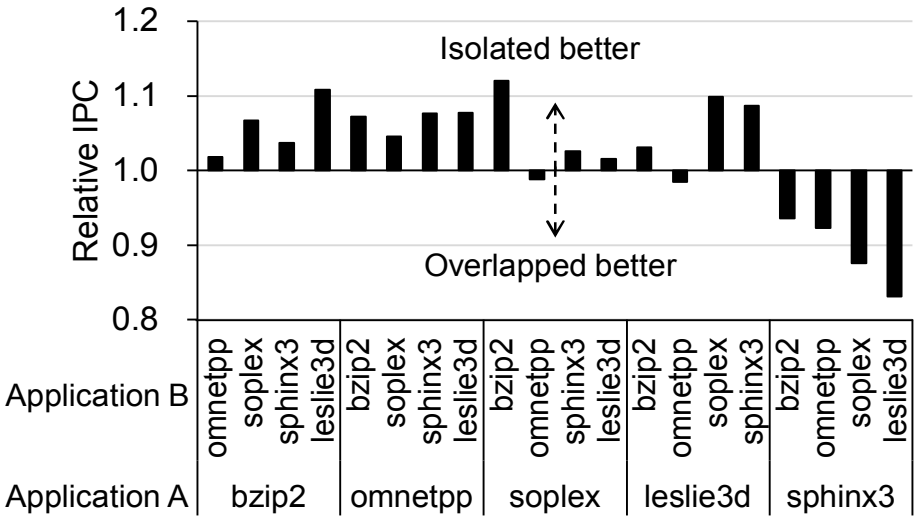


그림 8. 독립 분할과 중첩 분할 시 어플리케이션 A의 IPC 차이
(Relative IPC는 중첩 분할 시 IPC를 기준(1)으로 함)

그림 8은 CAT를 두 가지 분할 방법(독립 분할과 중첩 분할)으로 적용했을 때 어플리케이션 A의 IPC를 비교하였다. 어플리케이션 A에 가장 많은 way를 사용하게 하는 경우, 독립 분할은 18개 way 할당과 중첩 분할은 공유 way를 2개로 최소화, 총 20개 조합 중 14개 조합에서 독립 분할이 더 우수한 성능(~12%)을 보였고 나머지는 중첩 분할이 더 좋은 성능(~16%)을 보였다. Relative IPC는 중첩 분할 시 IPC를 '1'로 하였을 때 독립 분할 시 IPC를 나타낸다.

Test case	Benchmark A	Benchmark B	Relative IPC
Case1-1	bzip2	omnetpp	1.02
Case1-2		soplex	1.07
Case1-3		sphinx3	1.04
Case1-4		leslie3d	1.11
Case2-1	omnetpp	bzip2	1.07
Case2-2		soplex	1.05
Case2-3		sphinx3	1.08
Case2-4		leslie3d	1.08
Case3-1	soplex	bzip2	1.12
Case3-2		omnetpp	0.99
Case3-3		sphinx3	1.03
Case3-4		leslie3d	1.02
Case4-1	leslie3d	bzip2	1.03
Case4-2		omnetpp	0.98
Case4-3		soplex	1.10
Case4-4		sphinx3	1.09
Case5-1	sphinx3	bzip2	0.94
Case5-2		omnetpp	0.92
Case5-3		soplex	0.88
Case5-4		leslie3d	0.83

표 2. 독립 분할과 중첩 분할 시 Relative IPC

그림 9-1~5는 표 2의 각 'Test case'의 Case1~5의 way 변화에 따른 IPC와 LLC MPKI 경향을 보여주고 있다. X축 '# of ways'는 어플리케이션 A가 독점적으로 사용하는 way의 수를 나타내며, 독립 분할의 경우에는 어플리케이션 A가 단독으로 사용하는 way의 수를 나타내며 어플리케이션 B는 어플리케이션 A way의 나머지를 사용한다. 예로 '# of way'가 18인 경우 어플리케이션 A는 18개의 way를 사용하는 것이고, 어플리케이션 B는 2개 way를 사용하는 것이다. 중첩 분할일 경우 어플리케이션 A는 전체 20개 way를 사용하고 어플리케이션 B는 '# of way'의 나머지를 사용한다. '# of way'가

18인 경우 어플리케이션 A는 20개의 way를 사용하는 것이고, 어플리케이션 B는 독립 분할 시와 같이 2개 way를 사용한다. 이 수가 18이면 '그림 8'의 조건이다.

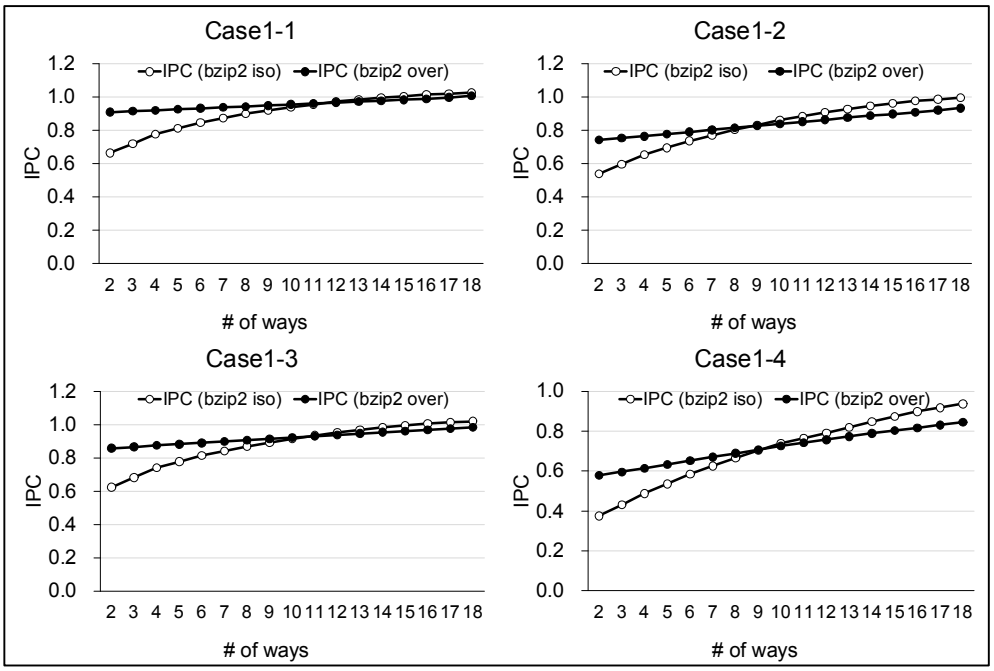


그림 9-1. Case1 (Application A: bzip2)

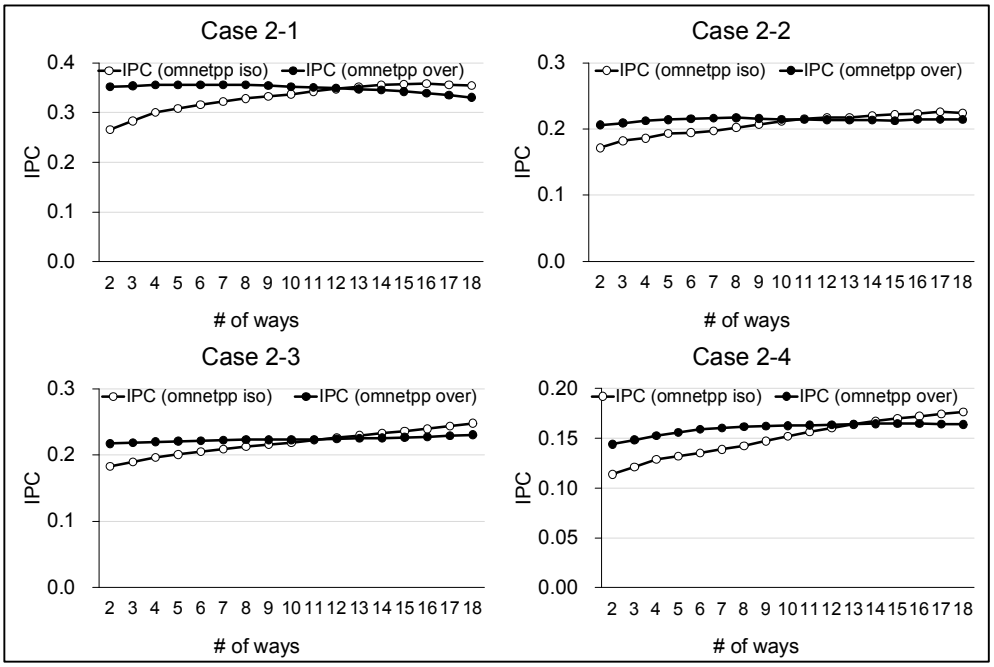


그림 9-2. Case2 (Application A: omnetpp)

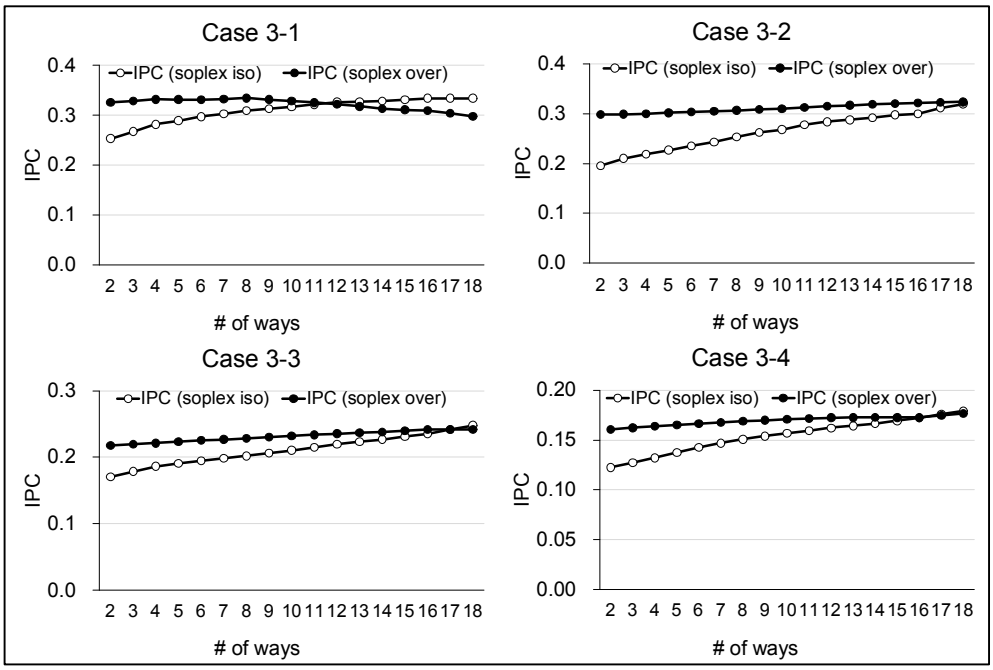


그림 9-3. Case3 (Application A: soplex)

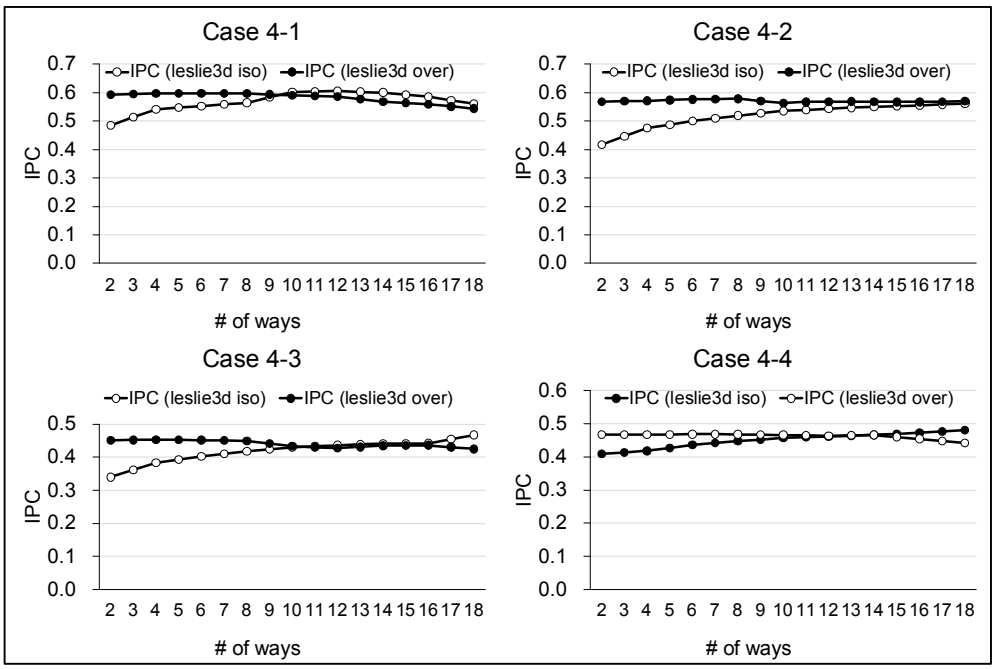


그림 9-4. Case4 (Application A: leslie3d)

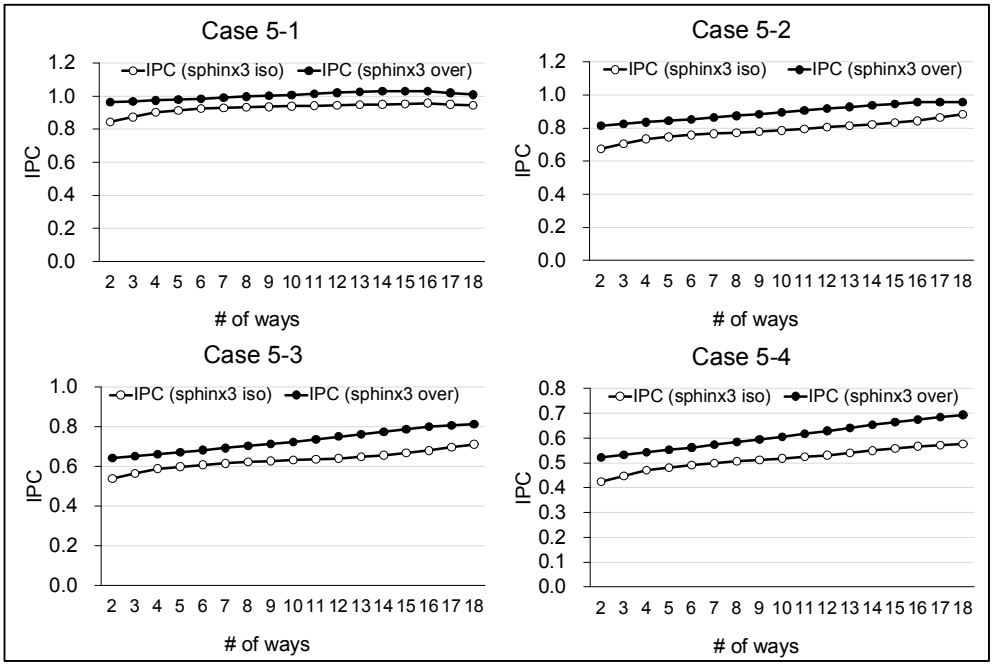


그림 9-5. Case5 (Application A: sphinx3)

각 어플리케이션의 조합에서 way 수 증가에 따른 IPC 경향을 확인하면 예상대로 어플리케이션 A에 많은 way를 할당할수록(또는 B와 공유하는 way의 수가 줄어들수록) IPC가 증가하는 것을 확인할 수 있다. 독립 분할 시(iso) 어플리케이션 A가 사용하는 캐시 용량이 2에서 18 way로 증가함에 따라 IPC가 증가하는 것을 볼 수 있다. 중첩 분할 시(over)에는 어플리케이션 A가 사용하는 캐시 용량은 20 way로 일정하나, 어플리케이션 B와 공유하지 않고 독점적으로 사용하는 캐시의 용량이 2에서 18 way로 증가함에 따라 IPC가 증가하는 것을 관찰할 수 있다.

어플리케이션 B에 18 way를 할당하였을 때는 중첩 분할 시 어플리케이션 A가 사용할 수 있는 캐시 용량(20 way)이 독립 분할 시 A가 사용할 수 있는 캐시 용량(2 way)보다 훨씬 크기 때문에 중첩 분할이 우위를 보인다. 그러나, 어플리케이션 B에 way를 줄어감에 따라,

즉 어플리케이션 A에 독립적 할당 way 수가 증가하면서 성능 역전이 발생하는 경우들을 많은 조합에서 확인할 수 있다. Case1-2(bzip2-soplex)에서 보면 B에 10 way 이하를 할당되었을 때 성능 역전이 발생한다.

상기 현상의 정확한 원인 분석을 위해 모든 조합의 LLC MPKI 경향을 추가적으로 확인하였고, 그림 10은 각 분할 방법이 우위를 보이는 대표적인 두 가지 조합에 대한 IPC와 LLC MPKI 경향을 보여준다. 그림 10 (a)는 독립 분할이 우위를 보이는 조합인 bzip2 (어플리케이션 A)와 soplex (어플리케이션 B)를 동시에 실행시킨 결과이다. 그림 10 (a)의 LLC MPKI 그래프에서 'Alone'이라고 명시된 선은

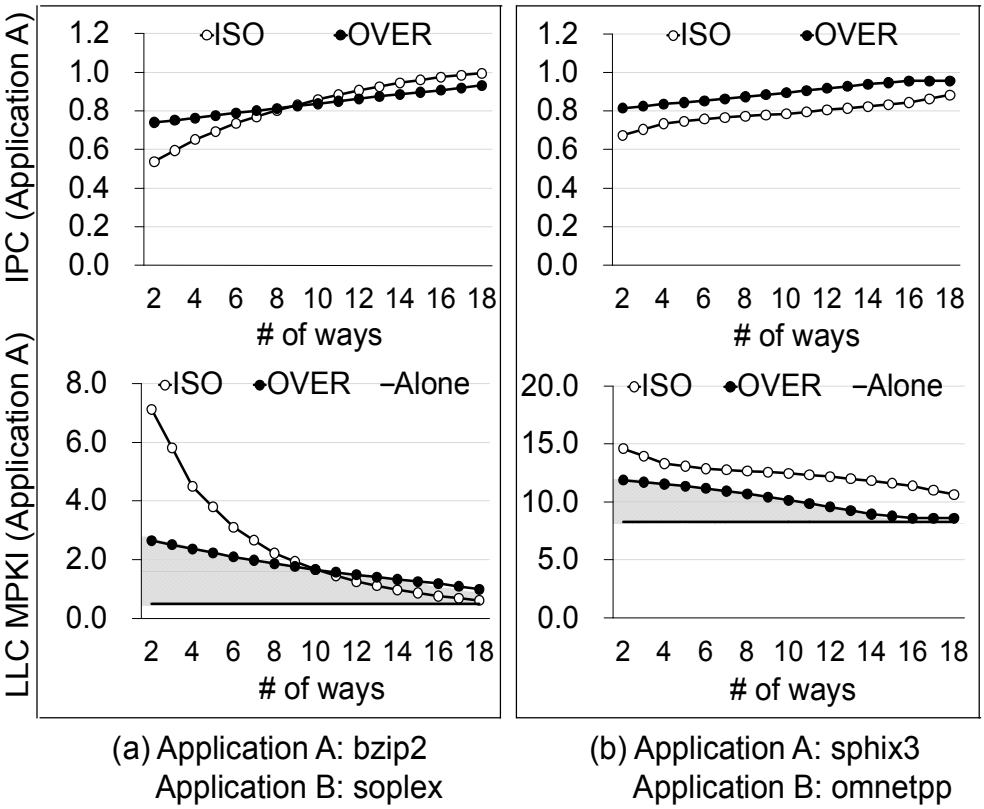


그림 10. 독립 분할이 우위인 조합(a), 중첩 분할이 우위인 조합(b)의 어플리케이션 A의 IPC 및 LLC MPKI

어플리케이션 A가 20 way를 모두 할당받았을 때의 LLC miss (MPKI)를 나타낸다. 이 선과 중첩 분할 시(OVER) LLC miss의 차이를 나타내는 회색 영역이 바로 contention miss에 의해서 발생한다. 즉, 중첩 분할의 경우도 20 way를 사용하지만 어플리케이션 B와 캐시를 공유하면서 발생하는 contention miss만큼 LLC miss가 늘어난 것이다. 반면에 독립 분할시(ISO) LLC miss와 'Alone' 선의 차이는 사용하는 캐시 용량의 차이에 의해 발생한다. 결국, 앞서 말한 성능 역전은 독립 분할 시(ISO) LLC의 일부를 다른 어플리케이션에 할당하면서 발생한 miss보다 중첩 분할 시(OVER) 발생하는 contention miss가 많을 때 발생한다. 그림 10 (b)는 중첩 분할 시 우위를 보이는 조합인 sphinx3 (어플리케이션 A)와 omnetpp (어플리케이션 B)를 동시에 수행시킨 결과이다. 중첩 분할(OVER)이 독립 분할(ISO)보다 모든 케이스에서 좋은 성능을 보이는 이유는, LLC MPKI 그래프에서 보여주는 바와 같이, 독립 분할 시 LLC의 일부를 다른 어플리케이션에 할당하면서 발생한 miss보다 중첩 분할 시 발생하는 contention miss (회색 영역)가 적기 때문이다. 본 연구는 그림 10의 독립 분할이 우세한 조합들에서 contention miss가 과도히 발생한 원인에 대해서 시뮬레이션을 통한 규명을 시도하였다.

제 3 장 중첩 분할 성능 열화 원인 분석

제 1 절 캐시 교환 정책 및 중첩 분할 성능 연관성

중첩 분할 시 발생하는 contention miss의 수는 캐시 교체 정책(cache replacement policy)과 연관이 있다. 인텔 CAT는 각 어플리케이션에 '물리적으로 위치가 고정된' 특정 way들을 할당하고, 할당한 way 내에서만 캐시 교체가 일어난다[8, 13]. 해당 방식은 중첩 분할 시 Least Recently Used (LRU)가 아닌 캐시 라인을 자주 evict시킨다. 그림 11 (좌)는 이러한 상황을 묘사하고 있다. 그림은 특정 set의 way들이 LRU (위)부터 MRU (아래)까지 정렬된 상태를 보여준다. way#의 '#'은 해당 way의 실제 물리적 저장 위치를 나타낸다.

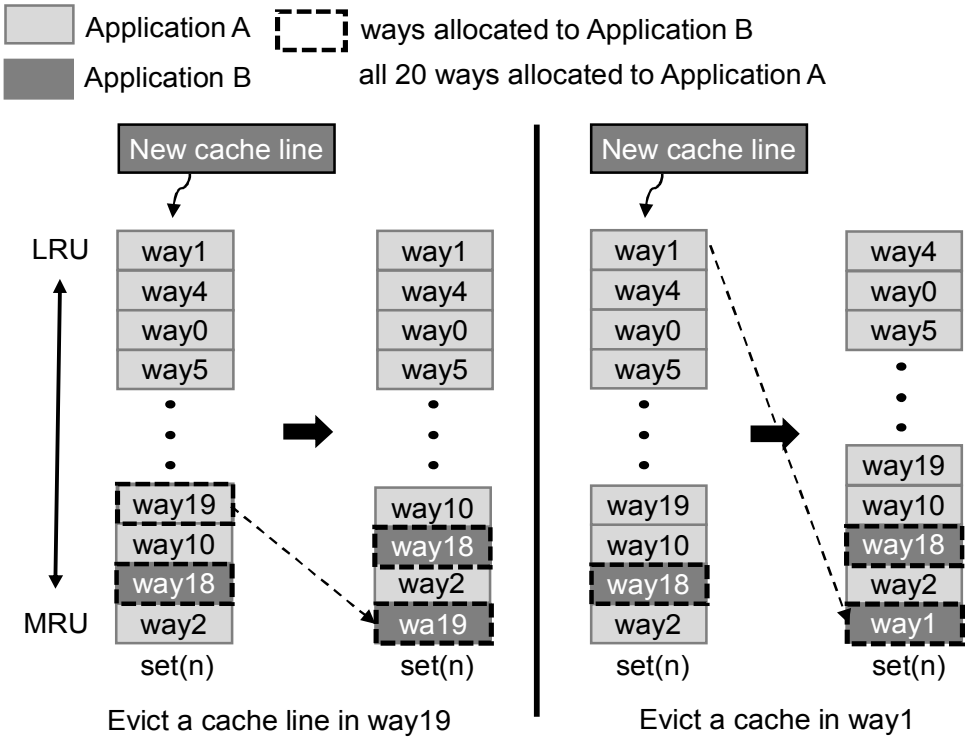


그림 11. 중첩 분할에서 캐시 교체 예시

어플리케이션 A에 way0-19이 할당되고 B에 way18-19이 할당되었다고 가정하면, 어플리케이션 A는 새로운 캐시 라인을 캐싱하기 위해 LRU인 way1을 evict시킬 수 있다. 그러나 어플리케이션 B는 새로운 캐시 라인을 캐싱하기 위해 공유하고 있는 way 중에서 가장 오래전에 사용한 way19를 evict시킨다. 만약 way19가 어플리케이션 A의 데이터를 갖고 있었다면 이 데이터는 어플리케이션 A 입장에서는 꽤나 최근에 사용된 데이터일 것이다. 이때 LRU인 way1을 evict시키는 것보다 미래에 캐시 미스가 발생할 확률이 매우 높다. 결과적으로 중복 분할 시 contention miss가 늘어나는 결과를 초래한다. 만약 CAT가 고정된 way를 할당하는 것이 아니라 어플리케이션에 할당된 way의 총 개수만을 유지/관리한다면, 그림 11 (우)처럼 어플리케이션 B에 의해 어플리케이션 A의 데이터가 evict되어야 할 때 항상 어플리케이션 A의 데이터 중 LRU인 way (way1)를 evict시킬 수 있다. 해당 상황에 대한 검증 실험을 McSimA+ 시뮬레이터를 이용하였다. 단일 프로그램에서의 way 할당에 따른 IPC 및 MPKI 변화를 확인하였고, 어플리케이션의 쌍으로 재현 및 개선에 대한 검증 실험을 진행하였다.

제 2 절 시뮬레이션 실험 환경 및 설정

시스템 시뮬레이션 실험은 다음과 같은 순서로 검증을 진행하였다. 첫째 단일 어플리케이션을 수행시켜 하드웨어 실험의 특성과 유사한 특성을 보이는지 확인하였다. 둘째, 그림 9의 결과를 시뮬레이션으로 재현하였다. 셋째, 중첩 분할의 경우 그림 10 (좌) 상황에서 평균적으로 위(LRU)에서 몇 번째 way가 evict되는 지 확인하였다. 넷째, 독립

분할이 우위를 보였던 조합이 CAT를 수정하여 그림 10 (우)대로 동작하는 상황에서 중첩 분할이 더 좋은 성능을 보이는지 확인하였다. 시뮬레이션 시스템 구성은 표 3과 같으며, 시스템 모델링과 어플리케이션 성능 측정을 위해 McSimA+를 사용하였다.

Component	Parameter
Processor Model	Out of Order
Processor Frequency	3.6 GHz
# of cores	16
Cache Line Size	64Byte
L1 I/D-Cache Size/Associativity	32 KB/4-way
L2 Cache Size/Associativity	256 KB/8-way
LLC(L3) Cache Size/Associativity	40 MB/20-way
L1,L2,L3 Replacement Policy	LRU
Network Configuration	Ring
Memory	DDR4 2400 MHz (2 rank/channel) x 4 channel
max_total_instrs	100000000

표 3. 실험에서 사용한 시뮬레이션 시스템 구성

제 3 절 검증 실험 결과

1. 단일 어플리케이션 실험 결과

하드웨어 실험에서 진행하였던 어플리케이션들을 시뮬레이터에서 동일하게 진행하였다. 코어별 L1, 2, LLC의 사이즈 및 시스템 구성을 동일하게 하여(표 3) 어플리케이션 별 IPC(논리적 코어당 평균)와 LLC MPKI(논리적 코어 당 평균)를 측정하여 비교하였다. 실험 결과는 그림 12과 같이 머신 실험과 유사한 결과를 얻을 수 있었다.

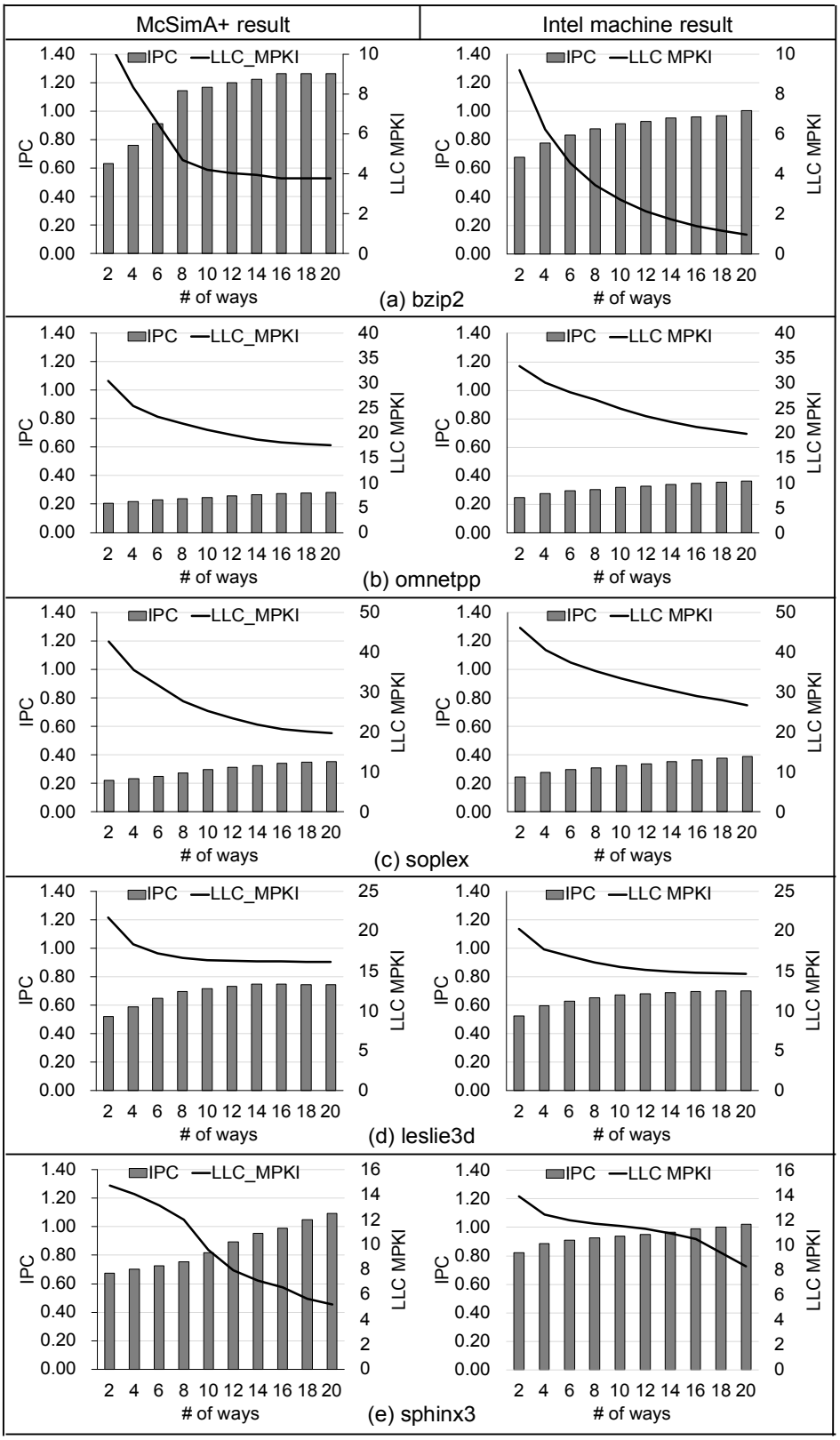


그림 12. 단일 어플리케이션의 캐시 민감도 비교

2. 시뮬레이션 재현 및 검증 실험 결과

하드웨어 실험에서 진행하였던 동일한 어플리케이션의 조합으로 진행한 결과는 way 변화에 따른 IPC 및 LLC MPKI 경향은 비슷하였다. 어플리케이션 A에 가장 많은 way를 할당하는 경우(App B에 2 way를 할당), 그림 13과 같이 11개 어플리케이션 조합(빨간 박스)이 하드웨어 실험과 유사한 결과를 보였다. 모든 조합에서 하드웨어와 동일 현상이 보이지 않았던 이유는 하드웨어와의 세부 구현 차이로 기인한 것으로 판단하였다.

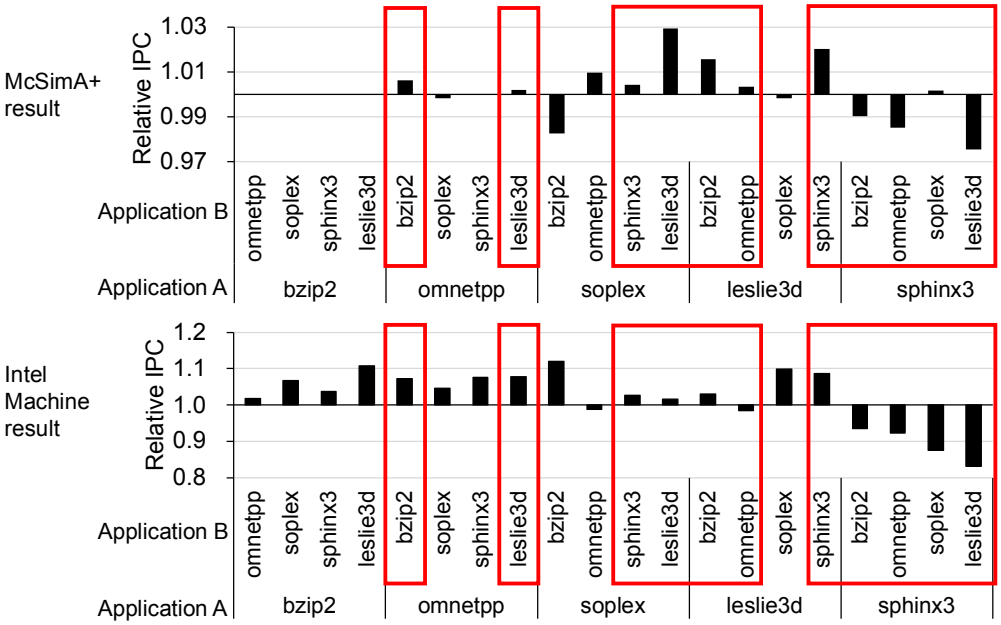
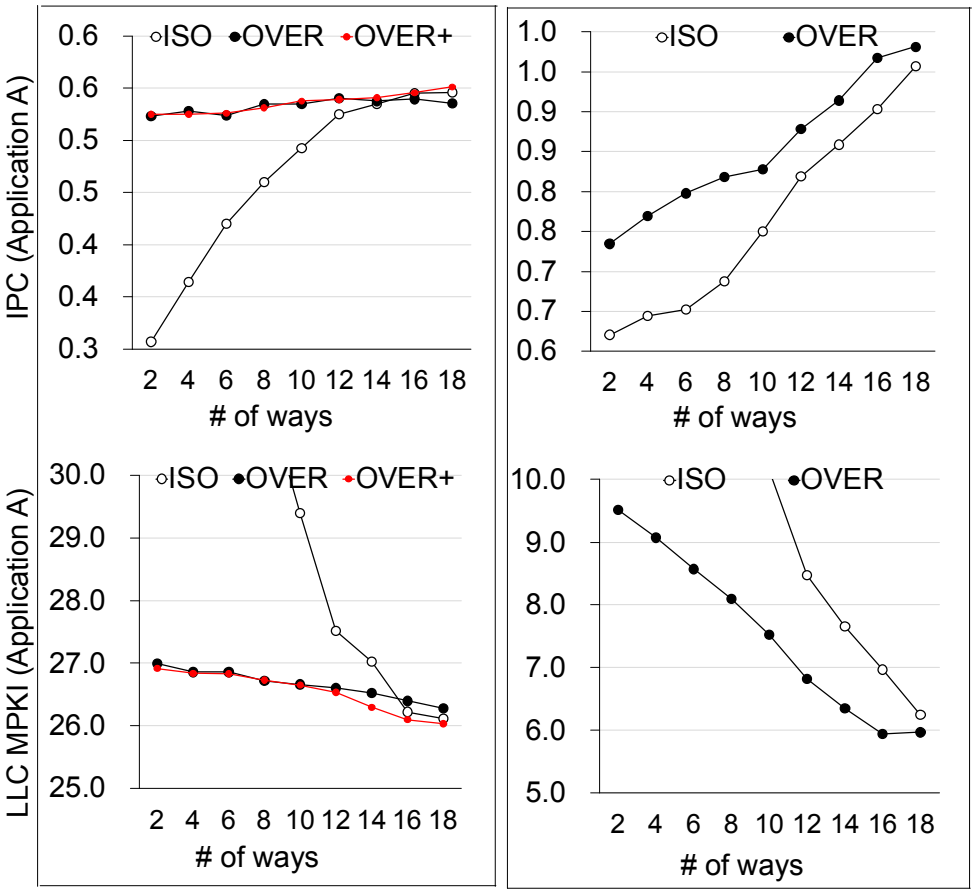


그림 13. 시뮬레이션 재현 및 하드웨어 실험과 비교

재현된 조합에서 독립 분할이 우위를 보이는 경우와 중첩 분할이 우위를 보이는 경우에 IPC와 MPKI의 경향을 보면 그림 14과 같다.

그림 14 (a)는 독립 분할이 우위를 보이는 조합의 IPC와 LLC MPKI 경향을 시뮬레이션을 통해 확인한 것이다. 어플리케이션 A가 독점적으로 사용할 수 있는 way 수가 18일 때, 중첩 분할(OVER)보다 독립 분할(ISO) 성능이 열세했으나 LRU 정책이 잘 반영되도록 수정한 중첩 분할(OVER+)의 경우는 독립 분할보다 우세한 성능을 보였고 MPKI도 더욱 적었다. 그리고 CAT 수정 전 중첩 분할(OVER)에서 평균적으로 LRU에서 평균 13번째 way가 evict되었으며 수정한 중첩 분할(OVER+)의 경우에는 LRU의 way가 evict된 것을 확인할 수 있다.



(a) Application A: leslie3d
Application B: sphinx3

(b) Application A: sphix3
Application B: soplex

그림 14. 시뮬레이션에서 독립 분할이 우위인 조합(a), 중첩 분할이 우위인 조합(b)의 어플리케이션 A의 IPC 및 LLC MPKI

그림 14 (b)는 중첩 분할이 우위를 보이는 조합의 IPC와 LLC MPKI 경향을 시뮬레이션을 통해 확인한 것이다. 하드웨어 실험 결과와 유사하게 공유 영역에서 contention miss가 적게 발생하여 해당 결과를 가져온 것으로 확인할 수 있다. 독립 분할이 우위를 보이는 경우와 다르게 중첩 분할에서 evict되는 way가 평균적으로 LRU에서 7번째 way로써 LRU에 가까운 캐시 라인이 제거되는 것을 확인하였다.

시뮬레이션 결과를 통해, 독립 분할이 우세했던 경우도 캐시 교체 방식의 비효율성만 개선한다면 중복 분할 시 contention miss가 크게 줄어서 독립 분할보다 좋은 성능을 가질 수 있다는 것을 보여주었다.

제 4 장 결 론

본 논문에서는 공유 라스트 레벨 캐시의 분할/할당 기능을 제공하는 인텔 CAT를 대상으로 하여, 두 가지 분할 방법(독립 분할과 중첩 분할)의 성능을 비교하기 위해 인텔 머신과 SPEC CPU2006 벤치마크로 실험을 진행하였다. 하드웨어 실험을 통해 20개 조합(어플리케이션 쌍) 중 14개 조합에서 독립 분할이 성능 우위를 보였으며, 나머지 조합에서는 중첩 분할이 성능 우위를 보였다. 독립 분할이 성능 우위를 보이는 경우는 중첩 분할 시 공유 영역에서 contention miss (캐싱 되었던 데이터가 타 어플리케이션에 의해 evict되어 발생하는 미스)가 과도하게 발생하여 성능이 저하되는 것을 확인하였다. 하드웨어 실험에서 확인한 중첩 분할 시 발생하는 과도한 contention miss에 대한 원인을 분석하고자 McSimA+ 시뮬레이터를 이용하였다. Contention miss가 증가한 것은 CAT가 특정 way를 할당하기 때문에 중첩 분할 시 캐시 교체 정책(예를 들어, LRU 정책)을 제대로 적용할 수 없기 때문인 것을 관련 문서를 통해 확인하였다. 이를 시뮬레이션 실험을 통해 재현하였고, LRU 캐시 교체 정책을 적용 시 LRU가 아닌 MRU에 가까운 캐시 라인이 evict 되는 것을 확인하였다. 해당 문제의 개선 방법으로 할당받은 way 총 수만을 관리하는 방법 적용하였다. 개선안 적용 시 과도한 contention miss가 줄어 독립 분할 방식보다 더 나은 성능을 보였다. 본 연구를 통해 공유 캐시 분할 방법에 대한 평가와 성능 개선 사항을 파악할 수 있었다.

참고 문헌

[1] Qureshi, Moinuddin K., and Yale N. Patt. "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches." Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on. IEEE, 2006.

[2] R. Iyer, “CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms”, Int’l Conference on Supercomputing (ICS), 2004.

[3] L. Zhao, R. Iyer, R. Illikkal, J. Moses, S. Makineni, D. Newell, "CacheScouts: Fine-Grain Monitoring of Shared Caches in CMP Platforms", PACT, 2007

[4] Herdrich, Andrew, et al. "Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family." High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. IEEE, 2016.

[5] Papadakis, Ioannis, et al. "Improving QoS and Utilisation in modern multi-core servers with Dynamic Cache Partitioning." Co-Scheduling of HPC applications (COSH 2017) and Virtualization Solutions for High-Performance Computing: 21.

[6] Subramanian, Lavanya, et al. "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory." Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on. IEEE, 2015.

[7] Jaleel, Aamer. "Memory characterization of workloads using instrumentation-driven simulation." Web Copy: http://www.jaleels.

org/ajaleel/workload/SPECanalysis.pdf

[8] Intel, C. A. T. "Improving Real-Time Performance by Utilizing Cache Allocation Technology." Intel Corporation, April (2015).

[9] Jung Ho Ahn et al., “McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling,” ISPASS, 2013.

[10] Intel CAT, CMT utility: https://github.com/intel/intel-cmt-cat

[11] Mittal, Sparsh. "A Survey of Techniques for Cache Partitioning in Multicore Processors." ACM Computing Surveys (CSUR) 50.2 (2017): 27.

[12] SPEC CPU 2006, https://www.spec.org/cpu2006

[13] Achieving QoS in Server Virtualization Intel Platform Shared Resource Monitoring/Control in Xen, https://events.static.linuxfound.org/sites/events/files/slides/Achieving%20QoS%20in%20Server%20Virtualization.pdf

[14] Intel, ”Intel® 64 and IA-32 Architectures Software Developer’s Manual”, 2017.

Abstract

A Study on Performance Based on Shared LLC Partitioning Method Using Cache Partitioning Technology

Seulgi Seo
Intelligent System
Department of Transdisciplinary Studies
The Graduate School
Seoul National University

When multiple applications are running on a multi-core system, contention and interference in a shared cache can hurt the performance of some or all of applications. Especially, when several applications use limited shared cache capacity, a real-time application or an application in which run-time is important may suffer performance degradation due to excessive cache occupancy by other applications. Some prior works proposed allocating independent cache resource to high-priority applications (e.g., cache partitioning).

Recently, Intel applied Cache Allocation Technology (CAT) which enables cache partitioning on its Xeon processor v3 family. The allocated cache spaces can be overlapped or isolated by

configuration. Overlapped partitioning can be advantageous to maximize the performance of higher priority applications than using isolated partitioning. Because all cache spaces can be allocated to high priority applications. However, there is the possibility that this prediction can be the opposite.

In this paper, we evaluate and compare the isolated and overlapped configuration of CAT with SPEC CPU2006 benchmark. In some application combinations, the prioritized application performs better on the overlapped cache partitioning configuration owing to its more cache capacity allocated to it. Whereas, in the other combinations, the prioritized application performs worse on the overlapped configuration due to excessive contention misses. Moreover, by system simulations, we show that these contention misses are increased by the limitation of applying cache replacement policy (e.g., LRU) under the overlapped configuration.

Keywords: Shared Last Level Cache, Isolated partitioning, Overlapped partitioning, Cache replacement policy

Student Number: 2016-26036