2022/06/21 ~ 2022/06/22

Last Level Cache(LLC) 공유로 인한 PM DAX I/O task의 noisy neighbor 판단 실험

Last Level Cache 확인

- `lscpu`를 통해 해당 시스템의 어느 cache가 last level cache인지 판단 후, `cat /sys/devices/system/cpu/cpu(core number)/cache/index(last level cache number)/shared_cpu_list`를 통해 last level cache를 공유하는 core list를 확인한다. 이때, (core number)는 보고자 하는 core의 번호이고, (last level cache)는 해당 시스템의 가장 마지막 level의 cache 번호이다.
  - lscpu 결과

```
L1d cache:                       1 MiB
L1i cache:                       1 MiB
L2 cache:                        32 MiB
L3 cache:                        44 MiB
```

  - shared_cpu_list (LLC가 L3 cache이므로 last level cache number에는 3을 넣어주어야 한다)
    `cat /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list`

```
root@pm:/sys/devices/system/cpu/cpu0/cache/index3# cat /sys/devices/system/cpu/cpu0/cache/index3/shared_cpu_list
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62
```

    현재 사용 중인 PM 서버의 경우 짝수 번째 core가 node 0의 core들이며, 이에 따라 0번의 LLC를 공유하는 core list를 살펴보면 node 0의 core들이 전부 출력되는 것을 확인할 수 있다.

Cache sensitive SPEC CPU workload

- PM으로의 DAX I/O가 LLC를 공유하는 다른 core의 computation task에 영향을 주는지와 얼마나 주는지를 판단하기 위해서는 LLC를 많이 사용하는 cache sensitive한 computation workload를 사용해야 한다.
- SPEC CPU 2017의 각 워크로드들의 cache sensitivity를 판단하기 위해 아래의 논문을 참고하였고, Figure 5를 기반으로 우선 상당 수준의 cache sensitivity가 존재하는 mcf 워크로드로 LLC noisy neighbor 관련 실험을 진행하기로 하였다.

Memory Centric Characterization and Analysis of SPEC CPU2017 Suite (ICPE'19)

- cf) 아래의 자료들 또한 cache sensitive한 워크로드 선별에 도움을 주었다.
  https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6675054/
  Cooperative virtual machine scheduling on multi-core multi-threading systems—a feasibility study

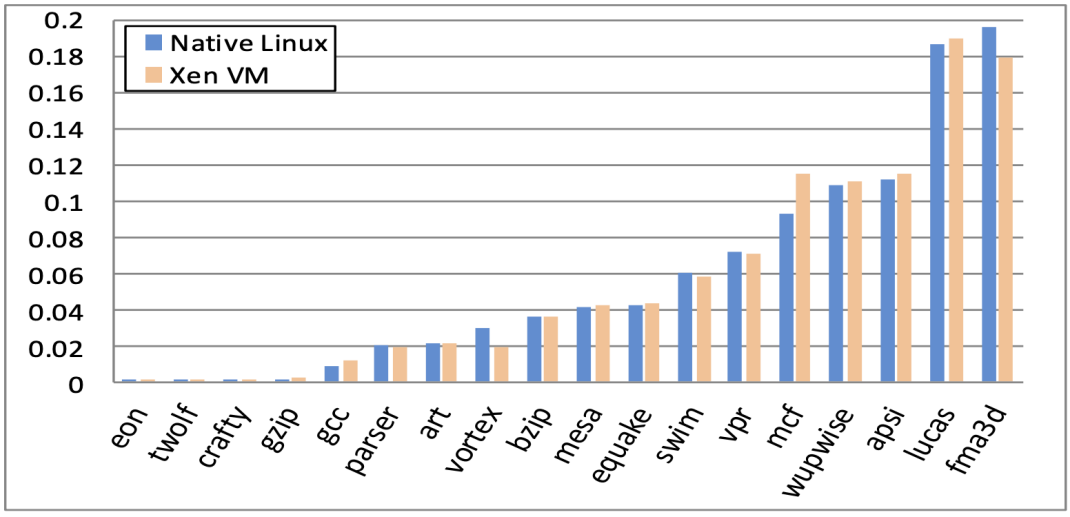

Figure 2. The LLC miss rates of SPEC CPU benchmarks when running on Native Linux vs. on Xen VM

LLC noisy neighbor 실험

▼ **실험 방법**

- mcf 워크로드는 2번 코어에서 실행을 시키되, L1 및 L2 캐시는 sibling core 간에 공유되므로 이 두 캐시로 인한 영향을 없애기 위해, 그리고 computing unit에서의 경합으로 인한 영향을 없애기 위해 sibling core인 34번 코어는 항상 idle 상태를 유지하도록 한다.
- 2번 코어를 제외한 나머지 16개의 node0 물리 코어에 PM으로의 DAX I/O를 수행하는 task들을 배치한다. 이때, 3가지 flush instruction을 사용하는 것을 고려하여 3가지 실험을 수행한다.
- PM DAX I/O task의 갯수를 변화시켜가면서 mcf 워크로드의 성능을 확인한다

▼ **실험 구성**

- mcf task 1개 (core 2)

- mcf task 1개 (core 2) + DAX I/O(clflush instruction) task 1개 (core 4)
- mcf task 1개 (core 2) + DAX I/O(clflush instruction) task 4개 (core 4, 6, 8, 10)
- mcf task 1개 (core 2) + DAX I/O(clflush instruction) task 8개 (core 4, 6, 8, 10, 12, 14, 16, 18)
- mcf task 1개 (core 2) + DAX I/O(clflushopt instruction without sfence) task 1개 (core 4)
- mcf task 1개 (core 2) + DAX I/O(clflushopt instruction without sfence) task 4개 (core 4, 6, 8, 10)
- mcf task 1개 (core 2) + DAX I/O(clflushopt instruction without sfence) task 8개 (core 4, 6, 8, 10, 12, 14, 16, 18)
- mcf task 1개 (core 2) + DAX I/O(clwb instruction without sfence) task 1개 (core 4)
- mcf task 1개 (core 2) + DAX I/O(clwb instruction without sfence) task 4개 (core 4, 6, 8, 10)
- mcf task 1개 (core 2) + DAX I/O(clwb instruction without sfence) task 8개 (core 4, 6, 8, 10, 12, 14, 16, 18)

▼ 실험 결과

- mcf task 1개 (core 2)
  - mcf test : 13.643775 s
  - mcf train : 77.325902 s
  - mcf ref : 532.873423 s
- mcf task 1개 (core 2) + DAX I/O(clflush instruction) task 1개 (core 4)
  - mcf test : 13.769359 s
  - mcf train : 77.518442 s
  - mcf ref : 532.815535 s
- mcf task 1개 (core 2) + DAX I/O(clflush instruction) task 4개 (core 4, 6, 8, 10)
  - mcf test : 13.685867 s
  - mcf train : 78.172024 s
  - mcf ref : 536.655061 s

- mcf task 1개 (core 2) + DAX I/O(clflush instruction) task 8개 (core 4, 6, 8, 10, 12, 14, 16, 18)
  - mcf test : 13.753992 s
  - mcf train : 77.719422 s
  - mcf ref : 535.810556 s
- mcf task 1개 (core 2) + DAX I/O(clflushopt instruction without sfence) task 1개 (core 4)
  - mcf test : 13.693201 s
  - mcf train : 77.765656 s
  - mcf ref : 533.783504 s
- mcf task 1개 (core 2) + DAX I/O(clflushopt instruction without sfence) task 4개 (core 4, 6, 8, 10)
  - mcf test : 13.738924 s
  - mcf train : 78.115928 s
  - mcf ref : 538.110977 s
- mcf task 1개 (core 2) + DAX I/O(clflushopt instruction without sfence) task 8개 (core 4, 6, 8, 10, 12, 14, 16, 18)
  - mcf test : 13.899757 s
  - mcf train : 80.058192 s
  - mcf ref : 564.545805 s
- mcf task 1개 (core 2) + DAX I/O(clwb instruction without sfence) task 1개 (core 4)
  - mcf test : 13.758858 s
  - mcf train : 77.595686 s
  - mcf ref : 532.827140 s
- mcf task 1개 (core 2) + DAX I/O(clwb instruction without sfence) task 4개 (core 4, 6, 8, 10)
  - mcf test : 13.709035 s
  - mcf train : 77.759863 s

    - mcf ref : 537.677619 s
- mcf task 1개 (core 2) + DAX I/O(clwb instruction without sfence) task 8개 (core 4, 6, 8, 10, 12, 14, 16, 18)
    - mcf test : 13.975927 s
    - mcf train : 80.092599 s
    - mcf ref : 563.813510 s

▼ **실험 결과 분석**

- clflush 결과 분석
    - PM DAX I/O write task 수가 증가함에 따라 mcf의 성능이 낮아지긴 하였으나, 낮아지는 폭이 8개 DAX I/O write task mcf ref 기준 0.55%로 매우 미미하였는데, 이는 clflush가 clflush 간에 ordering을 지키는 flush instruction이기 때문에 그만큼 덜 bus mastering을 수행하게 되고, 이로 인해 clflushopt 및 clwb보다 LLC를 공유하는 mcf task에 영향을 비교적 덜 미치기 때문으로 판단된다.
- clflushopt without fence 결과 분석
    - PM DAX I/O write task 수가 증가함에 따라 mcf의 성능이 낮아지는 경향성이 명확히 관측되었으며, 낮아지는 폭 또한 8개 DAX I/O write task mcf ref 기준 5.61%로 꽤 큰 성능 하락임을 확인할 수 있었다. 이는 clflushopt가 clflushopt 간에 ordering을 지키지 않아 clflush보다 더 자주 bus mastering을 수행하게 되고, 이로 인해 LLC를 공유하는 mcf task가 영향을 받아 상당한 성능 저하가 발생하는 것으로 판단된다.
- clwb without sfence 결과 분석
    - clflushopt와 완전히 동일하다.

Bus mastering이란?

- 컴퓨터 bus상에서 주변 기기를 관리하는 컨트롤러가 CPU를 거치지 않고 direct하게 bus 상의 다른 주변 기기들과 통신할 수 있도록 하는 bus 구조를 의미
    - 기존에는 data bus를 통해 data를 전송하기 위해서는 CPU가 관여함. 하지만 bus mastering 기능을 갖고 있는 컨트롤러를 사용할 경우, CPU의 개입을 최소화시켜 CPU 부하를 줄일 수 있게 됨.
    - Bus mastering 기능은 DMA(Direct Memory Access) 전송과 함께 사용됨

    - Bus master는 microprocessor 혹은 I/O 컨트롤러에 있는 프로그램으로, bus 혹은 I/O 통로 상의 통신을 제어함. (bus master가 통신을 개시 및 주관하는 master이고, I/O device들이 slave)
- 참고 자료
    - https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=nhk8318&logNo=60015996828

Direct Memory Access(DMA)란?

- CPU I/O 명령 하나에 의해 CPU를 거치지 않고 일련의 정보(block)를 memory에서 I/O device 또는 I/O device에서 memory로 전달하는 기술
    - CPU를 거치지 않고 직접 data를 전송하기 때문에 CPU 부하를 줄여서 전반적인 시스템 성능을 향상시킴
    - DMA는 data를 block 단위로 한번에 전송하며, 보통 memory의 특정 영역이 DMA를 위해 사용되도록 할당됨 (ISA bus 표준 : memory의 16MB까지 DMA를 위해 할당 가능. EISA 및 MCA 표준 : memory address의 전범위를 접근할 수 있도록 허용함)
    - PCI는 microprocessor의 I/O 제어를 PCI 컨트롤러에게 위임하는 bus mastering을 사용해서 DMA를 수행함
    - cf) DMA와 대비되는 방식은 PIO(Programmed Input/Output) interface로, 이 방식에서는 모든 data가 CPU를 통해서 device들에 전달됨
- 참고 자료
    - https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=nhk8318&logNo=60015996828