Intel Optane DC Persistent Memory
성능 측정 가이드

오현주

숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

Platform specification 확인

- 기본적인 socket 수, core 수 등을 확인할 수 있는 명령

```
user@user-Super-Server:~$ lscpu
```

- Microarchitecture 확인할 수 있는 명령

```
user@user-Super-Server:~$ cat /sys/devices/cpu/caps/pmu_name
```

  - 이를 통해 캐시가 shared인지 per-core인지 확인할 수 있음

Platform specification 확인

- 메모리 정보 확인하기 위한 명령 2가지
  - 각 슬롯별 메모리 정보 확인하고 싶을 경우

```
user@user-Super-Server:~$ sudo dmidecode -t 17 | egrep 'Memory|Size' | egrep -v 'No|Device'
```

  - 현재 어떤 메모리 모드가 적용되어 있는지 확인하고 싶은 경우
    - 단, ipmctl 이 먼저 설치되어있어야 함(ipmctl 설치는 뒷부분에서 다뤄짐)

```
user@user-Super-Server:~$ sudo ipmctl show -memoryresources
```

Platform specification - 316b

# of Sockets	1
CPU Spec.	16 cores Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
L1 Cache	512KiB i-Cache & 512KiB d-Cache
L2 Cache	16MiB
L3 Cache	22MiB
Total DRAM	64GB(16GB * 4)
Total NVMM	256GB(128GB * 2)
Storage(SATA)	3.6TB

Platform specification - Intel

# of Sockets	4
CPU Spec.	Genuine Intel(R) CPU 0000%@
L1 Cache	3MB i-Cache & 3MB d-Cache(per-core)
L2 Cache	96MB(per-core)
L3 Cache	132MB(shared)
Total DRAM	768GB (32GB * 24)
Total NVMM	6144GB(256GB * 24)
	Ubuntu 20.04.1 LTS

pmem managing tool

옵테인 메모리 managing을 위해 기본적으로 필요한 tool

숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

ipmctl

- Open source utility for configuring and managing PMM
- Created and is maintained by Intel
- 이 tool을 통해 옵테인 메모리의 모드(Memory mode, App Direct mode, 혼합된 모드 등) 변경 가능
- Region의 생성과 관리 도구
- 설치 방법

```
user@user-Super-Server:~$ sudo apt install ipmctl
```

ndctl

- Non-volatile Device Control(ndctl)은 리눅스 운영체제에서 namespace를 생성하고자 할 때 추천되는 tool
- Region에서 할당 받은 영역인 namespace를 관리
- 설치 방법

```
user@user-Super-Server:~$ sudo apt install ndctl
```

Memory → App Direct 모드로의 변환 방법

Step #1.

```
user@user-Super-Server:~$ sudo ipmctl create -goal
```

```
user@user-Super-Server:~$ sudo ipmctl create –goal PersistentMemoryType=AppDirect
```

- 역할: Creates an interleaved region configured for App Direct mode
- **둘 중 하나를 선택(같은 역할을 하는 명령어들)**해서 실행 후 reboot를 하면 App Direct mode로 변환 되어 있음
  - 잘 변환되었는지 확인하는 방법: 아래의 명령을 실행시킨 후 "no result" 라고 프롬프트에 떠야 함

```
user@user-Super-Server:~$ sudo ipmctl show -goal
```

Memory → App Direct 모드로의 변환 방법

Step #2.

```
user@user-Super-Server:~$ sudo ndctl create-namespace
```

- For the capacity to be surfaced to the Linux operating system and used by applications
- 위의 명령은 DAX support 네임스페이스를 생성함
  - DAX aware filesystem may subsequently be created on this namespace to provide optimal application performance

Memory → App Direct 모드로의 변환 방법

Step #3.

```
user@user-Super-Server:~$ sudo mkdir /mnt/pmem0
user@user-Super-Server:~$ sudo mkfs.ext4 /dev/pmem0
user@user-Super-Server:~$ sudo mount –o dax /dev/pmem0 /mnt/pmem0
```

App direct → Memory 모드로의 변환 방법

```
user@user-Super-Server:~$ sudo ipmctl create -goal MemoryMode=100
```

- 위의 명령을 사용하여 모드를 바로 바꾸게 되면 다음과 같은 오류가 남

```
user@user-Super-Server:~/mlc/Linux$ sudo ipmctl create -goal MemoryMode=100
[sudo] user의 암호:
WARNING! The requested memory mode size for 2LM goal is below the recommended NM:FM limit of
1:4
The following configuration will be applied:
 SocketID | DimmID | MemorySize  | AppDirect1Size | AppDirect2Size
==========================================================================
 0x0000   | 0x0001 | 126.000 GiB | 0.000 GiB      | 0.000 GiB
 0x0000   | 0x0101 | 126.000 GiB | 0.000 GiB      | 0.000 GiB
Do you want to continue? [y/n] y
Create region configuration goal failed: Error 124 - Namespaces exist on specified DIMMs. The
y have to be removed before running this command.
WARNING: Removing Namespaces will erase existing data. Please make a backup first.
```

  - 현재 configuration이 아예 delete된 상태에서 goal 설정이 이루어져야 함
    - 기존의 namespace와 region을 지워서 초기화해야 함
      - 참고 사이트1: How can I delete a region? · Issue #107 · intel/ipmctl · GitHub
      - 참고 사이트2: https://docs.pmem.io/ndctl-user-guide/managing-namespaces

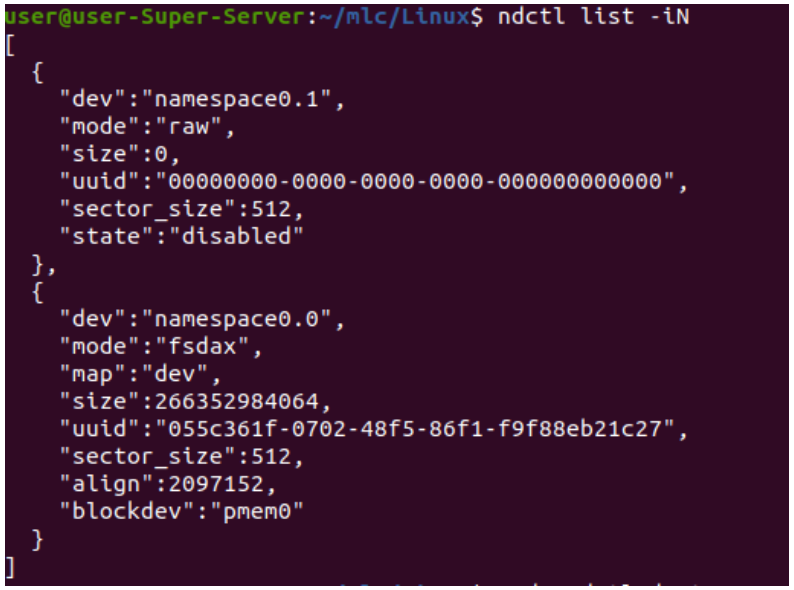
App direct → Memory 모드로의 변환 방법

Step #1. Delete current configuration

- Step #1-1. 존재하는 namespace 확인

```
user@user-Super-Server:~$ ndctl list -iN
```

  - 확인 결과



App direct → Memory 모드로의 변환 방법

Step #1. Delete current configuration

- Step #1-2. 존재하는 namespace 를 전부 disable 및 destroy하기

① Namespace를 disable 하기 전에 namespace가 mount 되어 있거나 다른 어플리케이션에 의해 사용 중인 경우 예상치 못한 결과로 이어질 수 있음

→ Always stop the application and unmount filesystems(fsdax mode) before disabling namespaces

```
user@user-Super-Server:~$ umount /dev/pmem0
```

② Disable namespace

```
user@user-Super-Server:~$ ndctl disable-namespace namespace0.0
```

App direct → Memory 모드로의 변환 방법

Step #1. Delete current configuration

- Step #1-2. 존재하는 namespace 를 전부 disable 및 destroy하기

③ Destroy namespace

```
user@user-Super-Server:~$ ndctl destroy-namespace namespace0.0
```

- 만약 disable 하지 않은 상태로 강제로 namespace를 destroy 하고 싶은 경우(f 옵션 사용)

```
user@user-Super-Server:~$ ndctl destroy-namespace -f namespace0.0
```

- 만약 존재하는 모든 namespace를 강제로 destroy 하고 싶은 경우

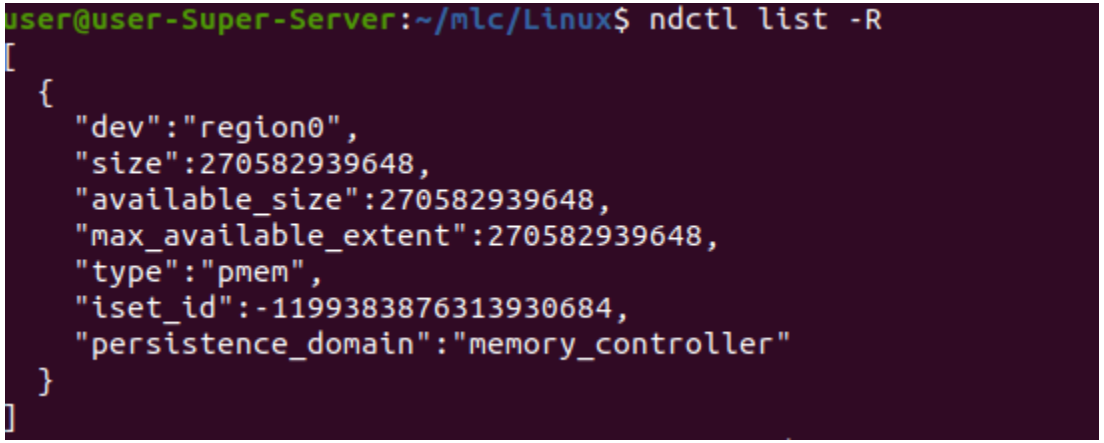
```
user@user-Super-Server:~$ ndctl destroy-namespace -f all
```

App direct → Memory 모드로의 변환 방법

Step #1. Delete current configuration

- Step #1-3. Active한 region들 disable 하기
  - ① List all active/enabled regions

```
user@user-Super-Server:~$ ndctl list -R
```



App direct → Memory 모드로의 변환 방법

Step #1. Delete current configuration

- Step #1-3. Active한 region들 disable 하기

② Region disable 하기

```
user@user-Super-Server:~$ ndctl disable-region region0
```

③ Disable된 region 확인하기(결과의 state 부분을 통해 확인 가능)

```
user@user-Super-Server:~$ ndctl list -Ri
```

App direct → Memory 모드로의 변환 방법

Step #2. 메모리 모드로 goal 설정

① 기존 goal이 있으면 삭제

```
user@user-Super-Server:~$ sudo ipmctl delete -goal
```

② goal 설정

```
user@user-Super-Server:~$ sudo ipmctl create -goal MemoryMode=100
```

③ 기존의 파일시스템 /mnt/pmem0 삭제

④ Reboot

성능 측정 도구

Intel Memory Latency Checker v3.9 사용

Intel MLC v3.9

- Default behavior of Intel MLC is to test only DRAM or Intel Optane DC persistent memory modules when configured in Memory Mode
  - To test Intel Optane DC persistent memory modules in App Direct Mode using Intel MLC, the namespaces must be created using devdax mode
- 설치 방법
  - https://software.intel.com/content/www/us/en/develop/articles/intelr-memory-latency-checker.html 홈페이지 하단에 mlc_v3.9.tgz 파일 다운로드해서 압축만 풀면 됨
  - mlc_v3.9.tgz 압축해제 후, 운영체제 환경에 따라 Linux 혹은 Windows 디렉토리 내의 mlc 실행 파일을 사용하면 됨
    - mlc 실행파일이 있는 디렉토리에서 mlc 명령을 실행시켜주면 됨(따로 설치한다는 개념이 없고, 실행파일을 사용하는 개념임)

※ **압축풀기 명령이 듣지 않는 경우,** 그냥 데스크탑에서 수동으로 해제해주거나 다른 컴퓨터에서 압축 푼 파일을 filezilla 등의 프로그램을 이용하여 전송시켜주면 됨
- 실행파일이 실행되지 않을 경우 **chmod 777** 사용

참고 사이트:
Solved: Intel's Memory Latency Checker (MLC) is showing a max read bandwidth of ~1.9TB/sec which is impossible since the theoretical max bandwidth for 12 chan... - Intel Community

Intel MLC v3.9

- 사용시 주의사항
  - 리눅스에서 사용할 경우, mlc 실행파일이 있는 디렉토리에서만 mlc 명령을 사용할 수 있음
  - prefetcher을 disable하려면 modprobe msr 명령을 실행시켜줘야함

```
user@user-Super-Server:~$ sudo modprobe msr
```

Latency 측정 명령 1

```
user@user-Super-Server:~$ sudo ./mlc --latency_matrix
```

- latency_matrix 를 이용한 latency 측정
  - Local 과 cross-socket 메모리 latency들을 matrix 형태로 출력해줌
  - By default, MLC **disables h/w prefetcher** on the thread measuring the latency and **uses sequential accesses**.
  - 같이 사용할 수 있는 옵션
    - -X: Use only 1 hyper-thread per core for bandwidth measurements(현재 시스템에서 Hyper-threading이 enable된 상태이기 때문에 준 옵션)
    - -b: 버퍼의 크기를 설정해줄 수 있는 옵션
      - 옵션 뒤에 버퍼의 크기를 같이 적어줌(ex. -b1g 라고 할 경우 버퍼의 크기를 1GB로 잡아주는 것)
    - -r: Initialize buffer(used by latency thread) with pseudo-random values so the access pattern to memory will be **random** for latency measurement

Latency 측정 명령 2

```
user@user-Super-Server:~$ sudo ./mlc --idle_latency
```

- idle_latency 를 이용한 latency 측정
  - Prints the idle memory latency of the platform.
  - By default, MLC **disables h/w prefetcher** on the thread measuring the latency and **uses sequential accesses**.
  - 같이 사용할 수 있는 옵션
    - -b: 버퍼의 크기를 설정해줄 수 있는 옵션
      - 옵션 뒤에 버퍼의 크기를 같이 적어줌(ex. -b1g 라고 할 경우 버퍼의 크기를 1GB로 잡아주는 것)
    - -J: Persistent memory의 경로를 지정해서 여기로 latency를 측정함(ex. -J/mnt/pmem0)
    - -r: Initialize buffer(used by latency thread) with pseudo-random values so the access pattern to memory will be **random** for latency measurement

Idle latency 측정

- 앞서 나온 latency 측정 명령 2가지 중 필요에 따라 사용하면 됨(각 명령마다 사용할 수 있는 옵션들이 다르므로) → 어느 명령을 사용해도 결과는 비슷하게 나옴

```
user@user-Super-Server:~/mlc/Linux$ sudo ./mlc --idle_latency -b60g
Intel(R) Memory Latency Checker - v3.9
Command line parameters: --idle_latency -b60g

Using buffer size of 61440.000MiB
Each iteration took 181.6 core clocks ( 78.9    ns)

user@user-Super-Server:~/mlc/Linux$ sudo ./mlc --loaded_latency -b60g -X
Intel(R) Memory Latency Checker - v3.9
Command line parameters: --loaded_latency -b60g -X

Using buffer size of 61440.000MiB/thread for reads and an additional 61440.000MiB/thread for writes

Measuring Loaded Latencies for the system
Using only one thread from each core if Hyper-threading is enabled
Using Read-only traffic type
^C
Exiting...


user@user-Super-Server:~/mlc/Linux$ sudo ./mlc --latency_matrix -b60g -X
Intel(R) Memory Latency Checker - v3.9
Command line parameters: --latency_matrix -b60g -X

Using buffer size of 61440.000MiB
Measuring idle latencies (in ns)...
                Numa node
Numa node            0
       0          78.9
```

Idle latency 측정

- 앞서 나온 latency 측정 명령 2가지 중 필요에 따라 사용하면 됨(각 명령마다 사용할 수 있는 옵션들이 다르므로) → 어느 명령을 사용해도 결과는 비슷하게 나옴
- 예시

```
user@user-Super-Server:~$ sudo ./mlc --idle_latency
```

```
user@user-Super-Server:~$ sudo ./mlc --idle_latency -r
```

```
user@user-Super-Server:~$ sudo ./mlc --idle_latency -J/mnt/pmem0
```

```
user@user-Super-Server:~$ sudo ./mlc --idle_latency -J/mnt/pmem0 -r
```

Idle latency 측정

- 멀티 소켓의 경우
  - 버퍼의 크기를 전체 메모리의 크기를 기준으로 결정하는 것이 아니라 소켓 1개의 메모리의 크기를 기준으로 결정해야함

예시) 전체 메모리 크기: 6144GB, 소켓 개수: 4개(인텔 스펙 참고) → 버퍼 크기를 6144g로 잡으면 **오류**

→ 버퍼 크기는 최대 **1536GB** 로 설정 가능

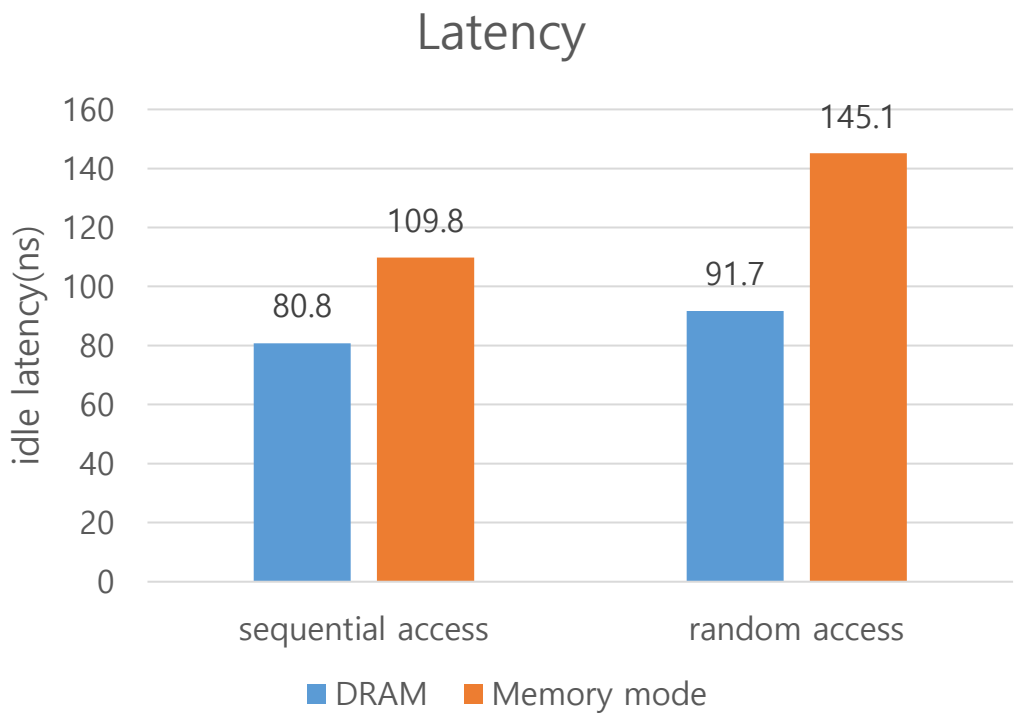
```
sdp@sdp:~/mlc_v3.9/Linux$ sudo ./mlc --latency_matrix -b2048g -X
[sudo] password for sdp:
Intel(R) Memory Latency Checker - v3.9
Command line parameters: --latency_matrix -b2048g -X

Using buffer size of -2097152.000MiB
*** Unable to modify prefetchers (try executing 'modprobe msr')
*** So, enabling random access for latency measurements
Measuring idle latencies (in ns)...
                Numa node
Numa node            0       1       2       3
       0        alloc_mem_onnode(): unable to mbind: : Invalid argument
Buffer allocation failed!
```

Idle latency 측정 결과 1

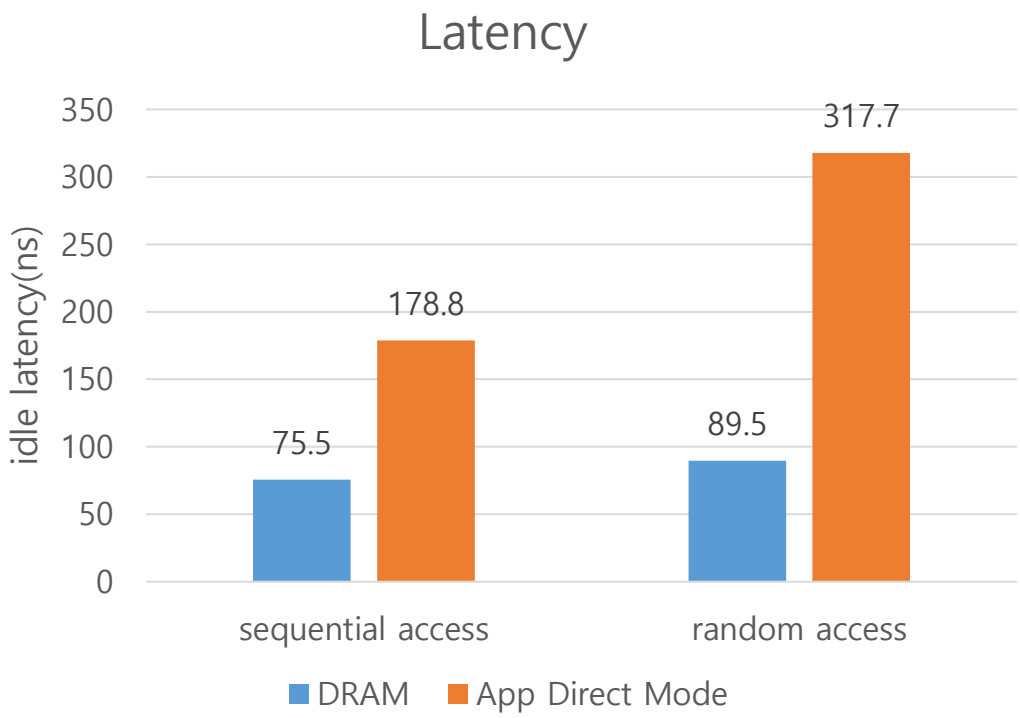
※ optane의 경우 모든 thread에서 사용하는 버퍼의 총 크기가 70~72GiB가 되도록 파라미터 조정

- 316b 서버

Memory mode

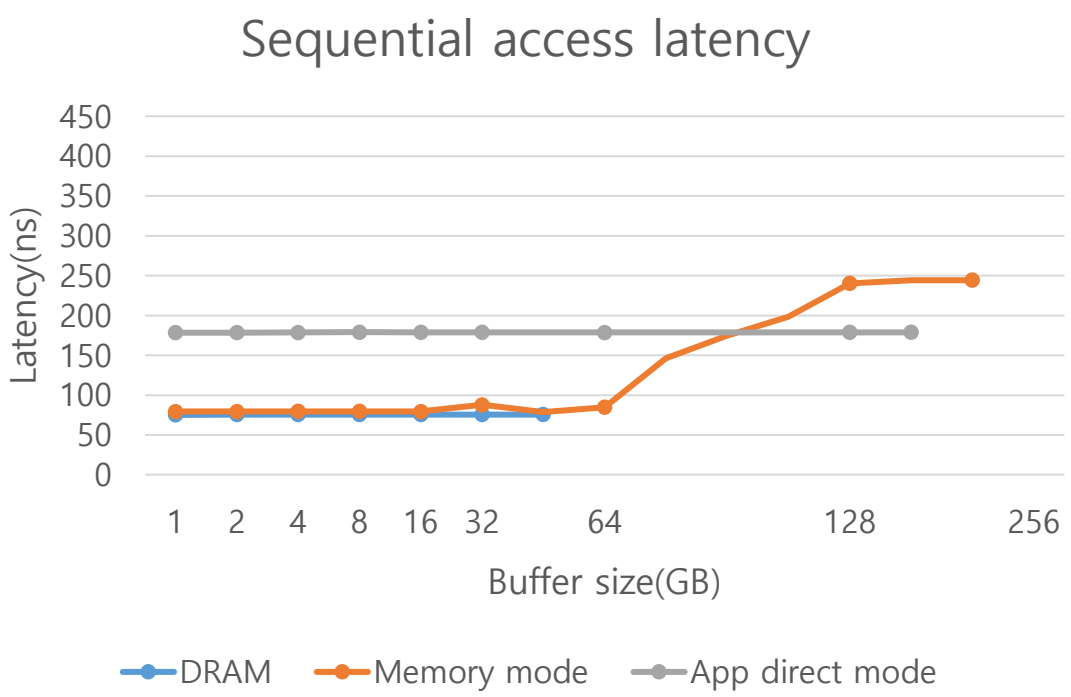


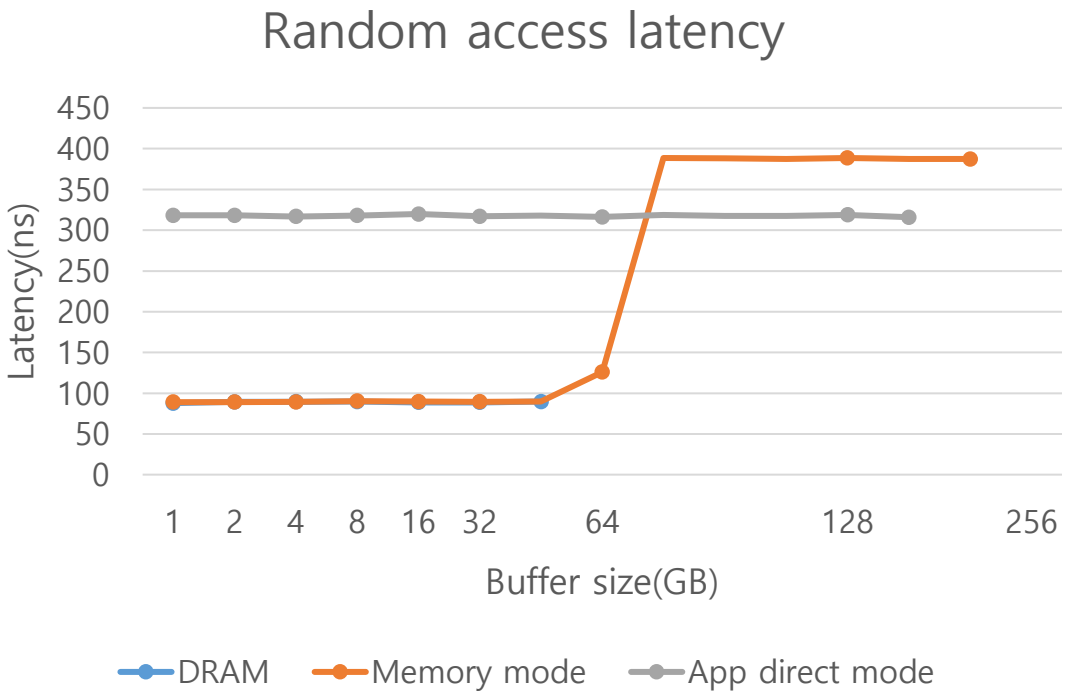
App direct mode



Idle latency 측정 결과 2

- 316b 서버





Bandwidth 측정

- Bandwidth 측정 명령

```
user@user-Super-Server:~mlc/Linux$ sudo ./mlc --loaded_latency -d0 -T -operthreadfile
```

  - -d 옵션: delay를 얼만큼 줄지에 대한 옵션(0으로 설정하면 delay가 없다는 뜻)
  - -T 옵션: 모든 thread를 bandwidth 측정에 사용하겠다는 뜻(원래는 cpu 0는 loaded_latency를 측정하는 thread)
  - -o 옵션: 뒤에 파일 이름을 붙임(현재 같은 디렉토리에 perthreadfile이라는 파일이 있음)
    - perthreadfile 내의 내용 예시(→ 뒤에 더 자세한 설명 제공)

```
0 R seq 200m dram
```

```
0 R seq 200m pmem /mnt/pmem0
```

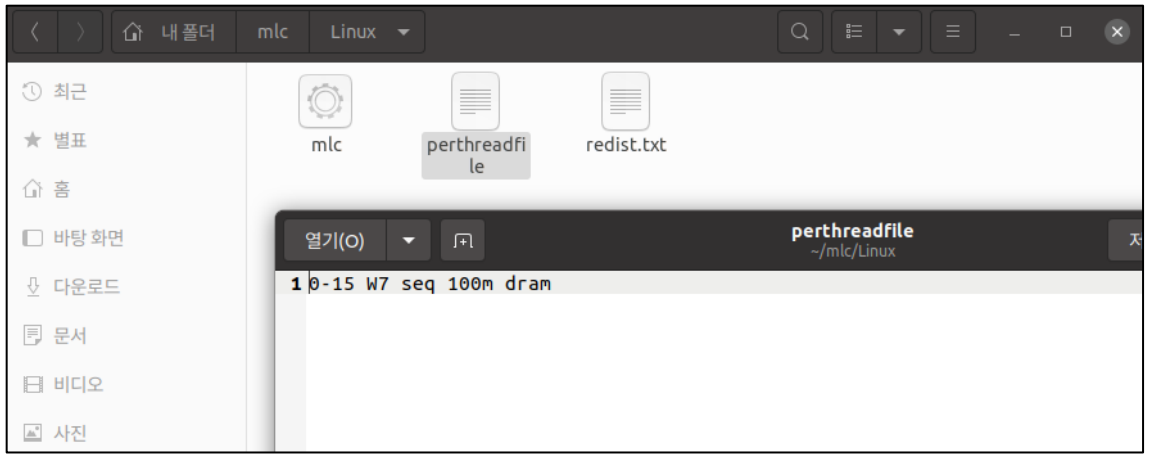
Bandwidth 측정

input file

- -o 옵션 뒤에는 파일의 이름이 위치(이 파일은 mlc 실행파일과 같은 위치에 있어야 함)
  - 예시의 경우 perthreadfile 이라는 이름이었음
  - Specify input file with options for per-thread controls during bandwidth measurements
  - loaded_latency 에서만 적용되는 옵션
  - input file 내부에 들어가는 내용은 mlc 문서 참고

<cpu-range> <traffic-type> <seq/random> <buf size in KB> <dram/pmem> <node-id/pmem-folder-path> <per-thread-delay>
The 1st field specifies the cpu-range. The remaining fields are control parameters to be applied to the cpu(s) specified by the 1st field.

input file 내에 들어가는 specification

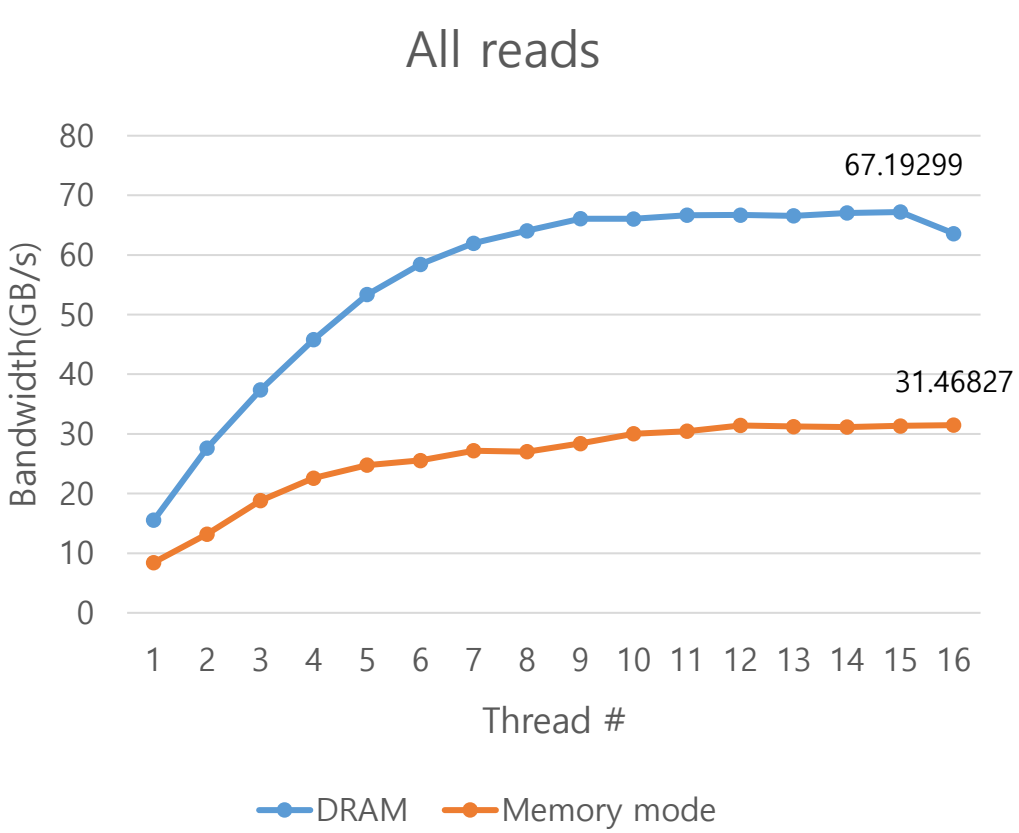


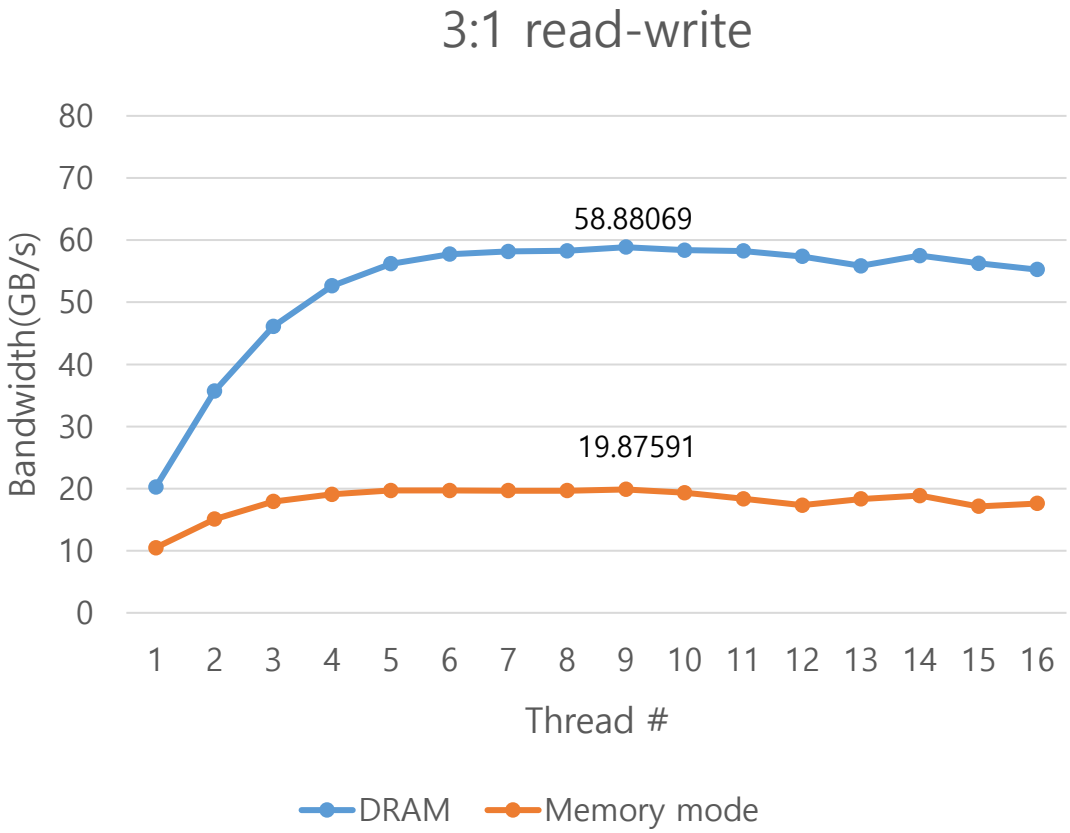
input file 위치 예시 사진

Bandwidth 측정 결과

※ optane의 경우 모든 thread에서 사용하는 버퍼의 총 크기가 70~72GiB가 되도록 파라미터 조정

- Memory mode(316b)

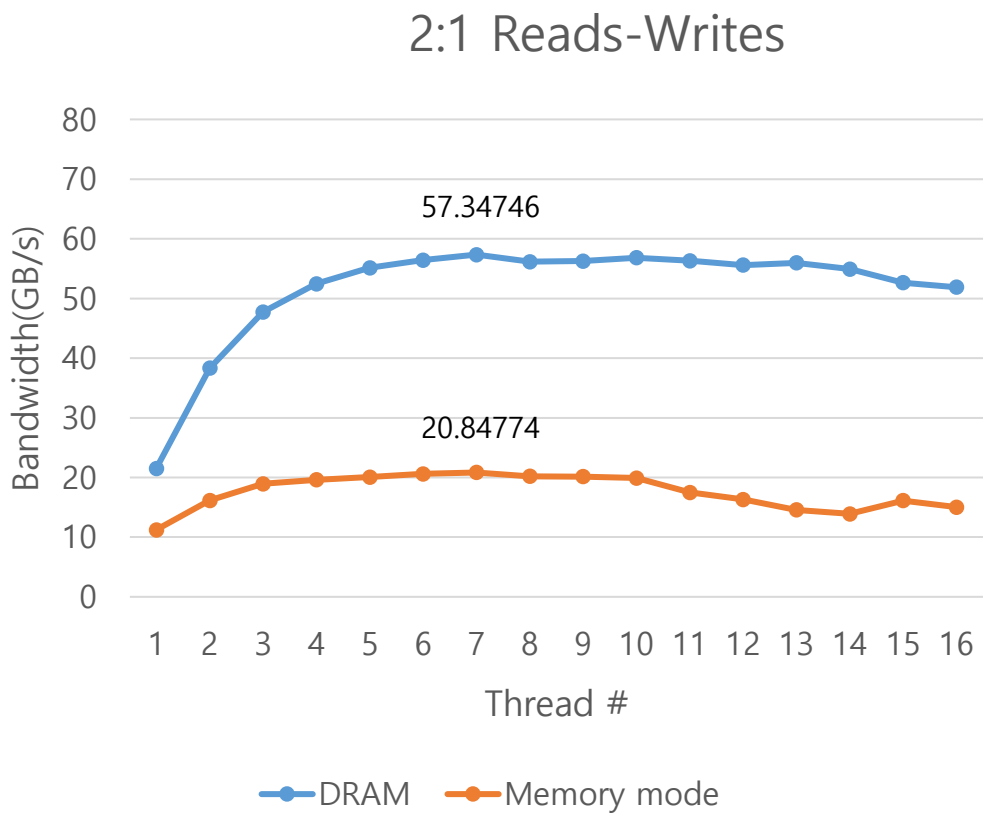


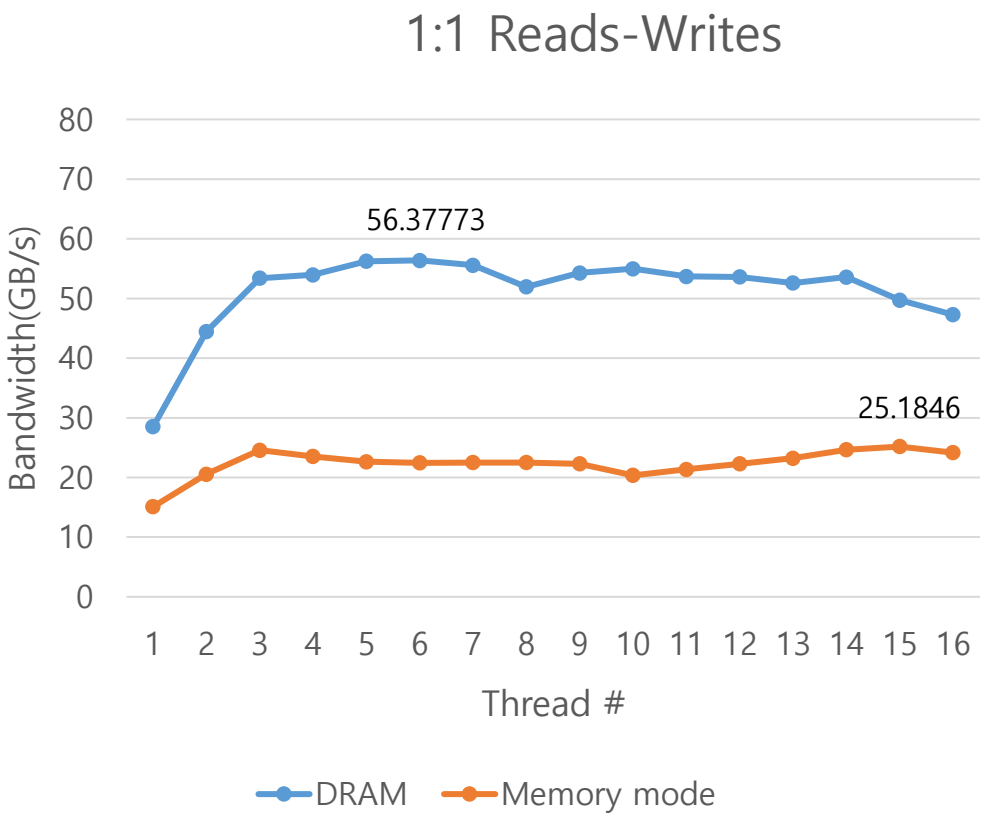


Bandwidth 측정 결과

※ optane의 경우 모든 thread에서 사용하는 버퍼의 총 크기가 70~72GiB가 되도록 파라미터 조정

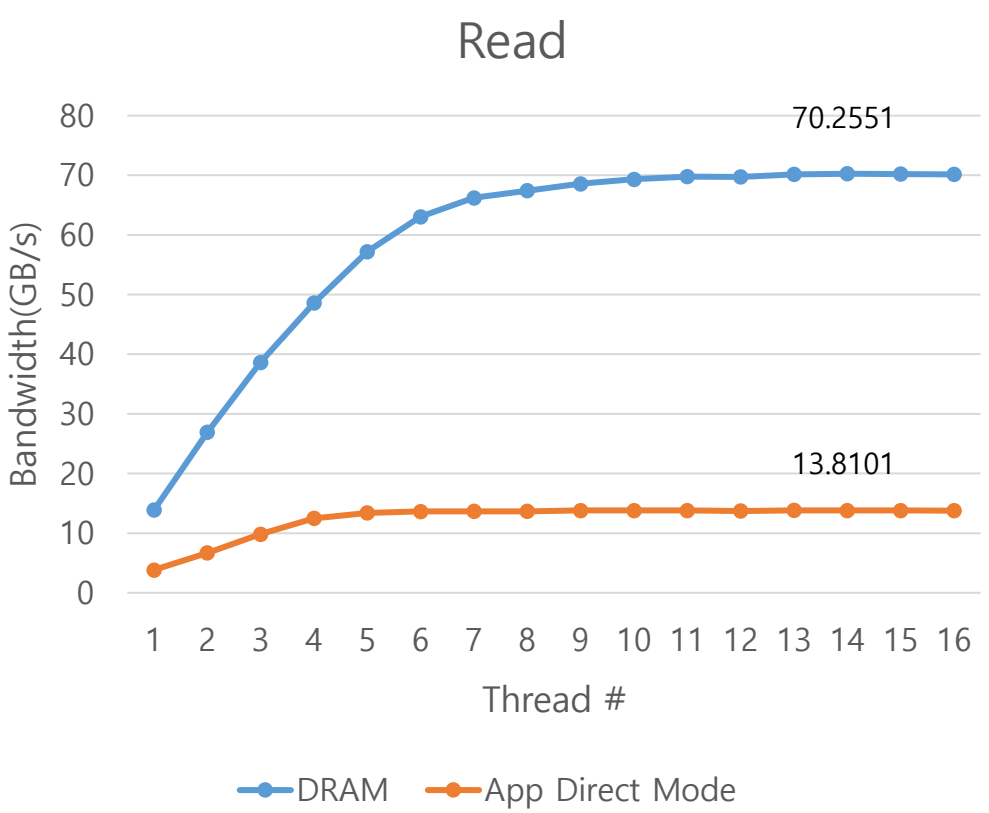
- Memory mode(316b)

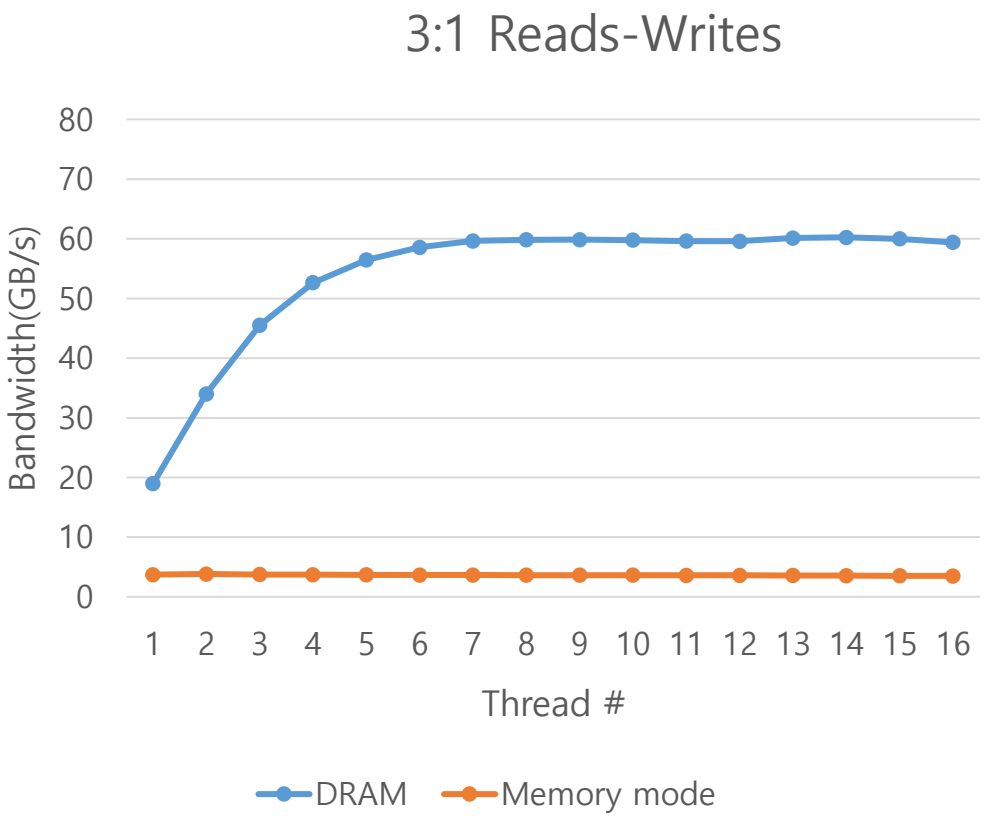




Bandwidth 측정 결과

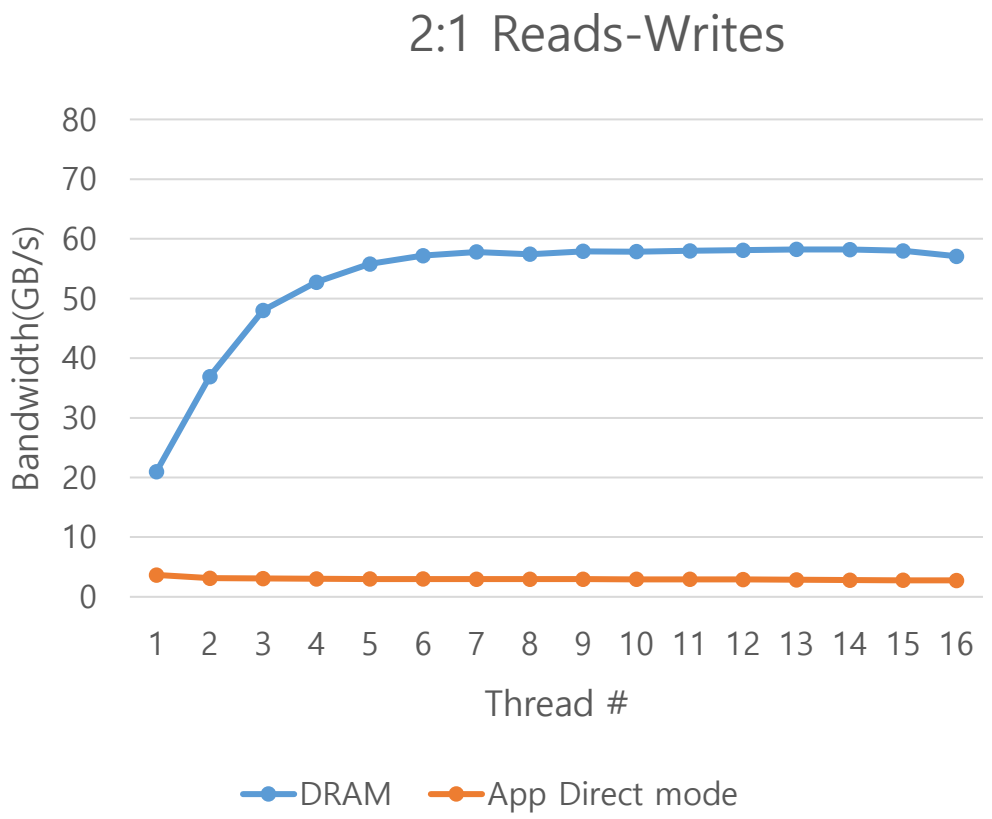
- App direct mode(316b)

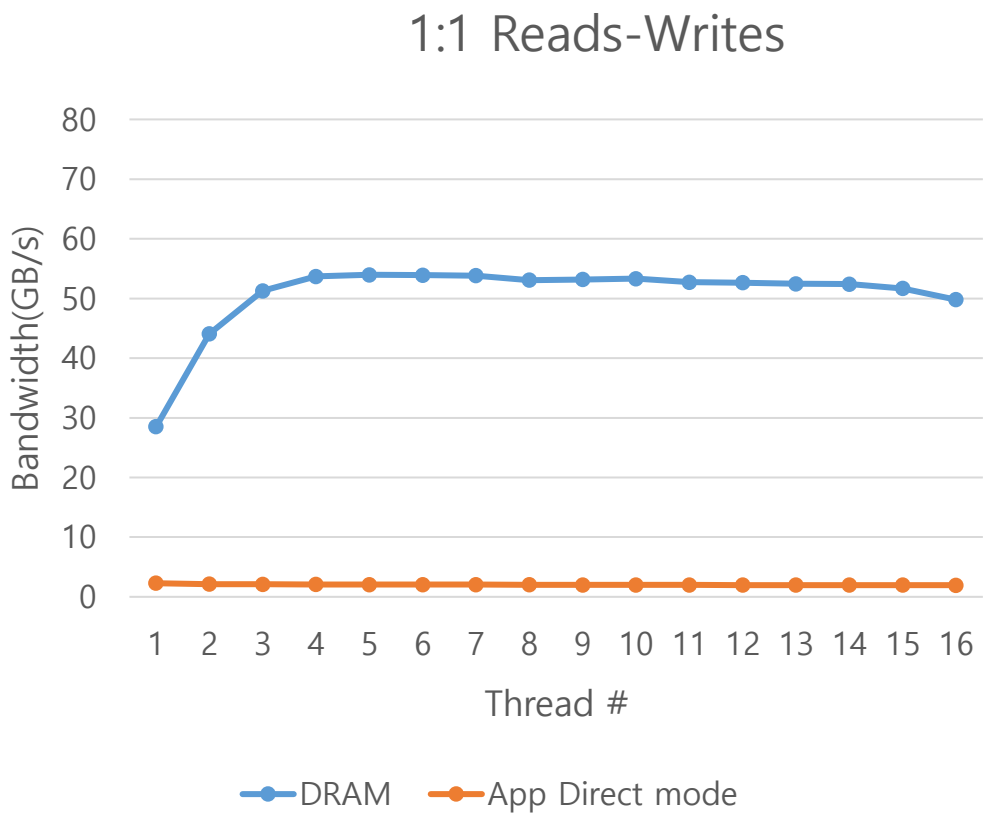




Bandwidth 측정 결과

- App direct mode(316b)



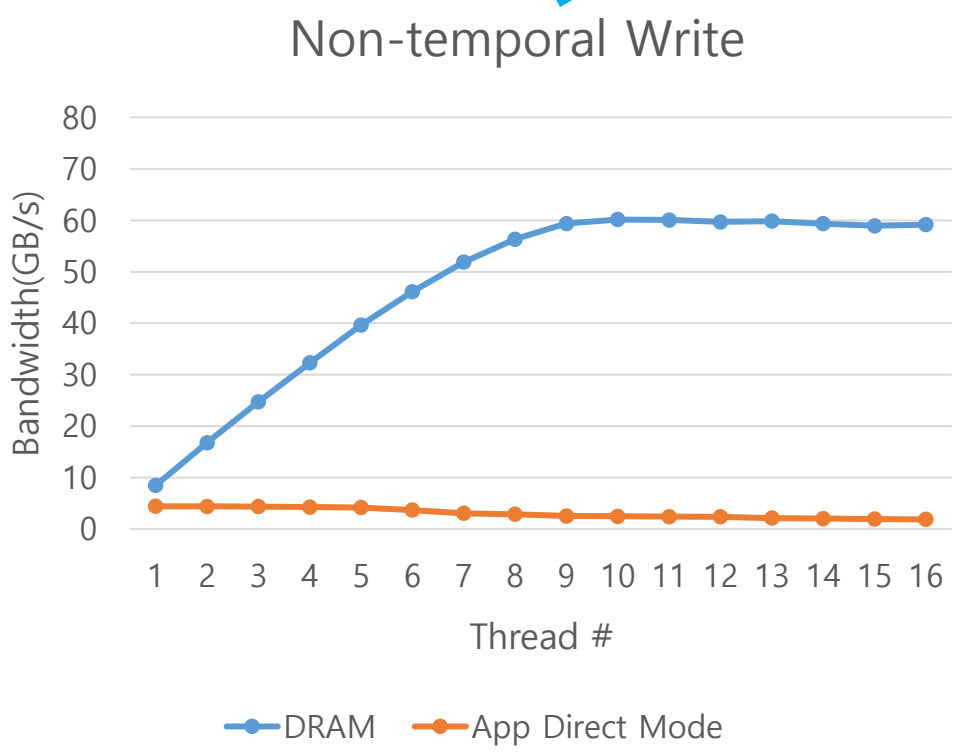


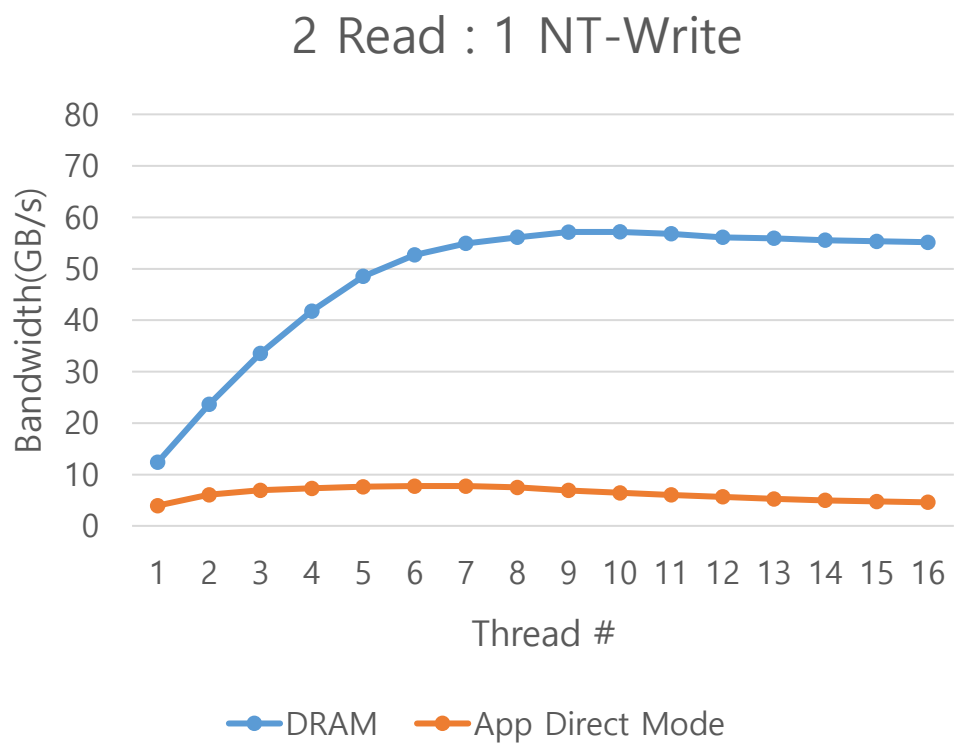
Bandwidth 측정 결과

- App direct mode(316b)

Non-temporal write

- Non-temporal: The data will not be reused soon, so there is no reason to cache it
- Non-temporal write operations do not read a cache line and then modify it; instead, the new content is directly written to memory



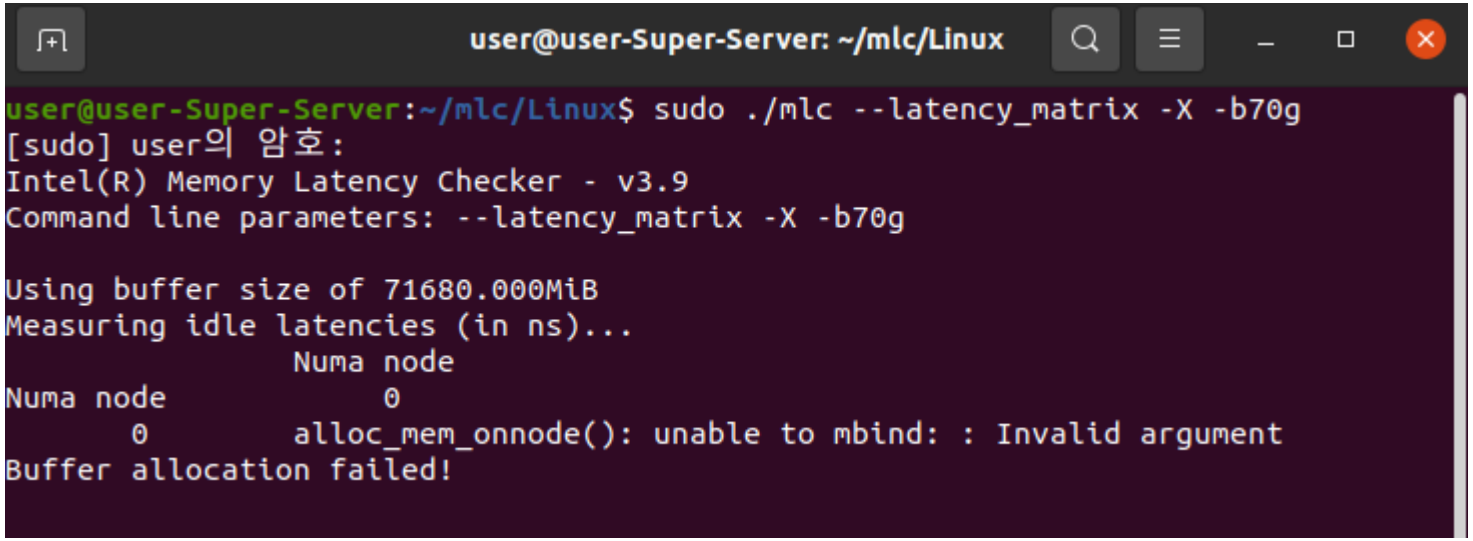


NT-Write 출처: https://lwn.net/Articles/255364/

오류 목록

MLC 사용 중 나타나는 오류 1

- alloc_mem_onnode(): unable to mbind: : Invalid argument Buffer allocation failed



- 원인: buffer 크기가 메모리의 크기를 넘어섰을 때 일어남
- 옆의 예제의 경우 옵테인 메모리가 app direct mode로 설정되어 있는 상황에서 64g 보다 더 큰 버퍼 크기를 할당하여 오류가 남
- 해결책: 옵테인 메모리가 어떤 모드인지 확인하기

※ 대부분의 오류는 버퍼 크기 설정에 대한 오류이므로 꼭 버퍼 크기가 메모리 크기에 맞는지 확인하기

MLC 사용 중 나타나는 오류 2

- 같은 버퍼 크기인데 다른 latency 값이 나옴

```
user@user-Super-Server:~/mlc/Linux$ sudo ./mlc --latency_matrix -X -b32g
Intel(R) Memory Latency Checker - v3.9
Command line parameters: --latency_matrix -X -b32g

Using buffer size of 32768.000MiB
Measuring idle latencies (in ns)...
                Numa node
Numa node            0
       0          87.8
```

```
user@user-Super-Server:~/mlc/Linux$ sudo ./mlc --latency_matrix -X -b32g
[sudo] user의 암호:
Intel(R) Memory Latency Checker - v3.9
Command line parameters: --latency_matrix -X -b32g

Using buffer size of 32768.000MiB
Measuring idle latencies (in ns)...
                Numa node
Numa node            0
       0         185.7
```

  - 해결책: 리부팅하면 다시 원래의 값으로 돌아옴(정확한 원인은 찾지 못했지만, mlc 내부의 문제라고 생각되어짐)