RPMA - Performance Report

Release Cascade Lake + Mellanox + CentOS 8.4 kernel 4.18 + FSDAX

Testing Date: February 2022

Performed by:

- Łukasz Dorau (lukasz.dorau@intel.com)
- Tomasz Gromadzki (tomasz.gromadzki@intel.com)
- Oksana Sałyk (oksana.salyk@intel.com)
- Patryk Zbierski(patryk.zbierski@intel.com)
- Kacper Stefański (kacper.stefanski@intel.com

Audience and Purpose

This report is intended for people who are interested in evaluating RPMA performance.

The purpose of the report is not to imply a single "correct" approach, but rather to provide a baseline of well-tested configurations and procedures that produce repeatable results. This report can also be viewed as information regarding best-known methods/practices when testing the RPMA performance.

This report focuses on the persistent memory accessed by (exposed via) File System DAX.

Disclaimer

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel technologies may require enabled hardware, software or service activation.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Test Setup

Common Configuration (both initiator and target)

Item	Description
Server Platform	`S2600WFT`

Item	Description
CPU	Intel® Xeon® Gold 6252 CPU@2.10GHz
Memory	12x 32GB Micron 36ASF4G72PZ-2G9E2, DDR4, 2666MT/s Total of 384GB DRAM
Persistent Memory	12x 256GB Intel® Optane™ Persistent Memory 100 Series, 2666MT/s Total of 3072GB PMem 12GiB File System DAX on 6x Interleaved region
RDMA-capable NIC	100Gbps Mellanox Technologies ConnectX-5 MTU 4200 RoCEv2
Storage	Intel® SSDSCKKB48
Operating System	CentOS Linux 8.4 (Core)
BIOS	SE5C620.86B.02.01.0013.121520200651
Linux kernel version	4.18.0-305.3.1.el8.x86_64
librpma	0.12.0
libibverbs	1.14.35.0
libpmem	1.6.1
FIO version	https://github.com/pmem/fio/tree/rpma fio-3.29
rdma-core	Version 54mlnx1 Release 1.54103
Testing Date	February 2022

Target Configuration

Intel® Data Direct I/O Technology (DDIO) turned off on a per PCIe Root Port basis for the Mellanox NIC PCIe Root Port. This is required to allow direct writing to remote PMem via RDMA.

Item	Description
Repository	https://github.com/pmem/pmem.github.io
Version	7068cc62c108a31e573a7df7ba50df2e34d4fb9b
Applied procedure	Details
Testing Date	February 2022

BIOS Settings

Item	Description
Repository	https://github.com/pmem/pmem.github.io
Version	7068cc62c108a31e573a7df7ba50df2e34d4fb9b
Applied procedure	Details
Testing Date	February 2022

Excerpt:

Item	Description
Package C-State	C0/C1 state
C1E	Disabled
CPU Power and Performance Policy	Performance

Item	Description
Fan Profile	Performance
Memory Configuration - Average Power Budget	18mW
Memory Configuration - NMV Performance Setting	Latency Optimized
Intel® Turbo Boost Technology	Enabled
Energy Efficient Turbo	Disabled
Intel® Hyper-Threading Technology	Disabled
Intel® Virtualization Technology	Disabled
Thermal Monitor	Disabled
Testing Date	February 2022

Kernel & BIOS spectre-meltdown information

Both server systems use CentOS 4.18.0-305.3.1.el8.x86_64 kernel version available from repository with default patches for spectre-meltdown issue enabled.

BIOS on all systems was updated to post spectre-meltdown versions as well.

Benchmarking process

To learn how the collection, processing, and presentation of the data was conducted please see https://github.com/pmem/rpma/blob/master/tools/perf/BENCHMARKING.md.

Introduction to RPMA

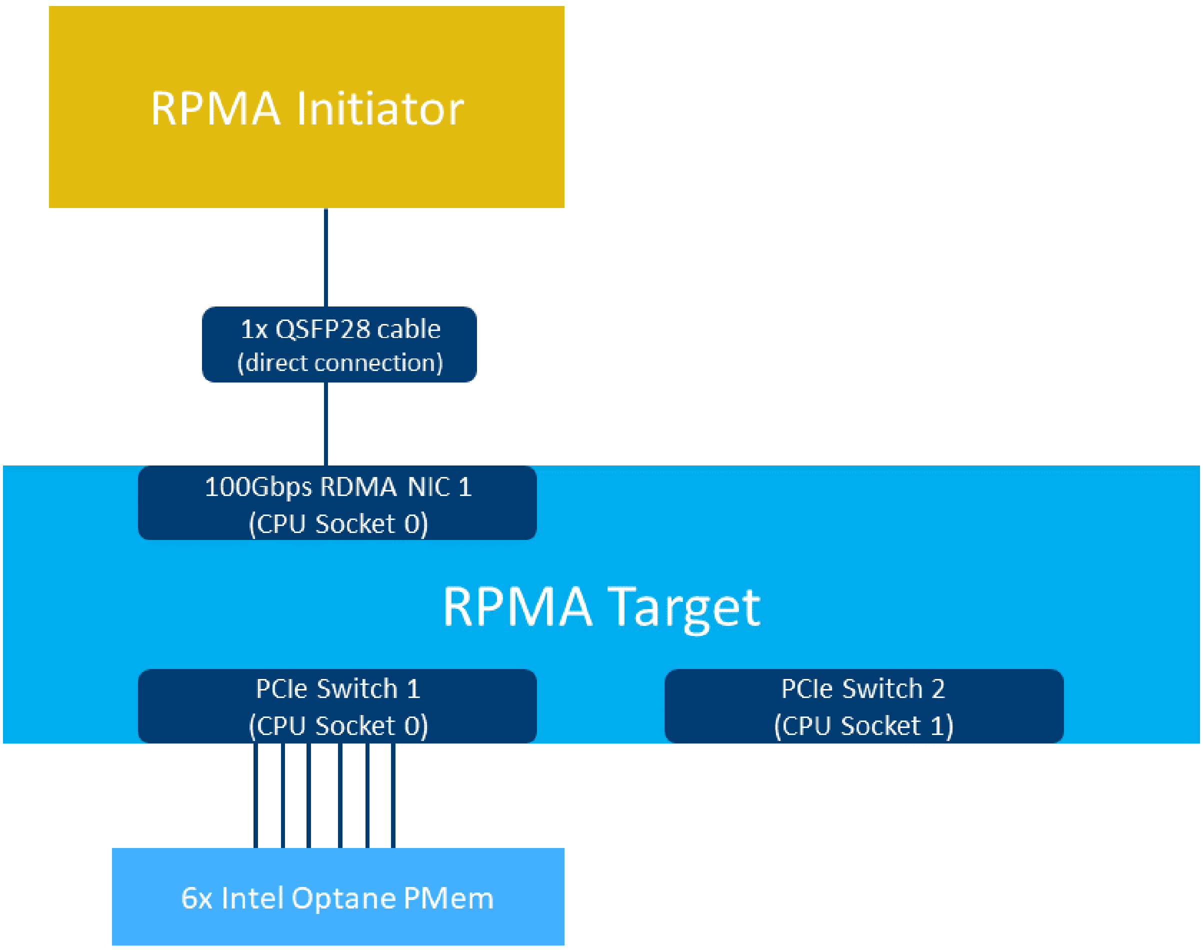
The Remote Persistent Memory Access (RPMA) library (librpma) uses Remote Direct Memory Access (RDMA) to provide easy to use interface for accessing Persistent Memory (PMem) on the remote system. It is a user-space library which may be used with all available RDMA implementations (InfiniBand™, iWARP, RoCE, RoCEv2) from various providers as long as they support the standard RDMA user-space Linux interface namely the libiverbs library.

The RPMA-dedicated FIO engines are created as complementary pairs namely librpma_apm_client along with librpma_apm_server and librpma_gpspm_client along with librpma_gpspm_server. For the simplicity sake, both parts are implemented independently without any out-of-band communication or daemons allowing to prepare the target memory for I/O requests. The server engine prepares memory according to provided job file (either DRAM or PMem), registers it in the local RDMA-capable network interface (RNIC) and waits for the preconfigured (via the job file) number of incoming client's connections. The client engine establishes the preconfigured number of connections with the server. After these setup steps the client engine starts executing I/O requests against the memory exposed on the server side.

The RPMA library and any application using it (including FIO with dedicated engines) should work on all flavours of RDMA transport but it is currently tested against RoCEv2.

The FIO should be configured in a way that guarantees running all its threads and allocating all its buffers from a single NUMA node, the same the used RDMA interface is attached to, to avoid costly cross-NUMA synchronizations (e.g. using Ultra Path Interconnect).

Please see a high-level schematic of the systems used for testing in the rest of this report. The setup consists of two individual systems (one used as the initiator and one used as the target). The target system has six Intel® Optane™ Persistent Memory 300 Series devices connected to the respective NUMA node. Both the initiator and the target are equipped with 100Gbps Mellanox ConnectX-5 NICs. The initiator is connected to the target system directly using QSFP28 cables without any switches. From two ports available on NICs only one is in use during the measurements.



Read from PMem

Benchmarking the `rpma_read()` operation against various data sources (PMem, DRAM) and comparing the obtained results to standard RDMA benchmarking tools: `ib_read_lat` and `ib_read_bw`.

Read from PMem: Latency

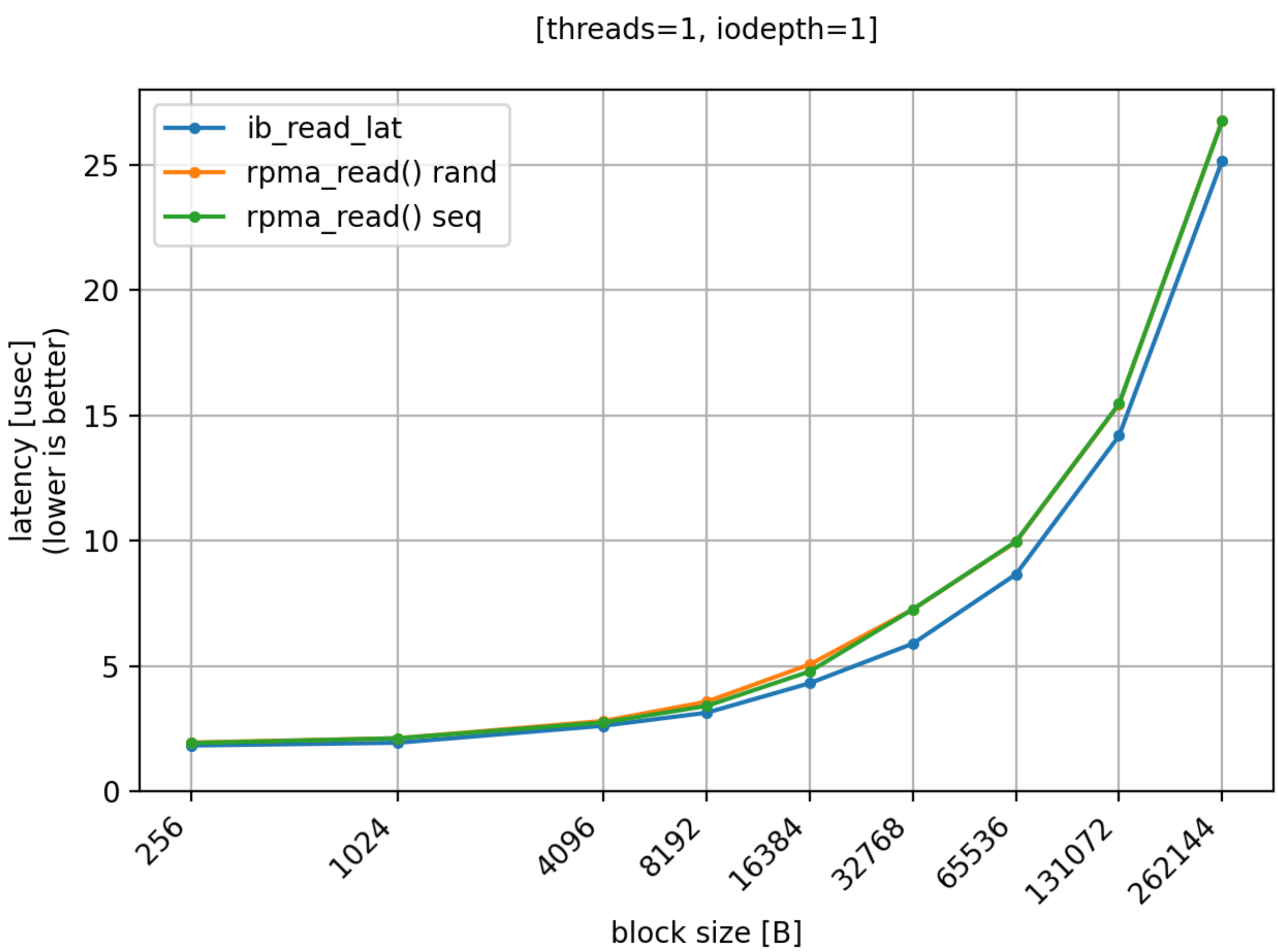
Comparing the latency of `rpma_read()` from PMem on **the RPMA Target** to the latency of `rpma_read()` from DRAM on **the RPMA Target** using the `ib_read_lat` as the baseline.

Item	Description
Server - ib_read_lat configuration	`--size $blocksize`
Client - ib_read_lat configuration	`--iters $iters --size $blocksize --perform_warm_up $serverip`

Server - FIO engine configuration	`[global]`
	`ioengine=librpma_apm_server`
	`create_serialize=0`
	`kb_base=1000`
	`serverip=$serverip`
	`port=7204`
	`thread`
	
	`[server]`
	`direct_write_to_pmem={0, 1} # 1 for Device DAX`
	`numjobs=1`
	`size=100MiB`
	`filename={malloc, /dev/daxX.Y} # for DRAM or Device DAX`
	`filename_format=/mnt/pmem/file # for File System DAX`

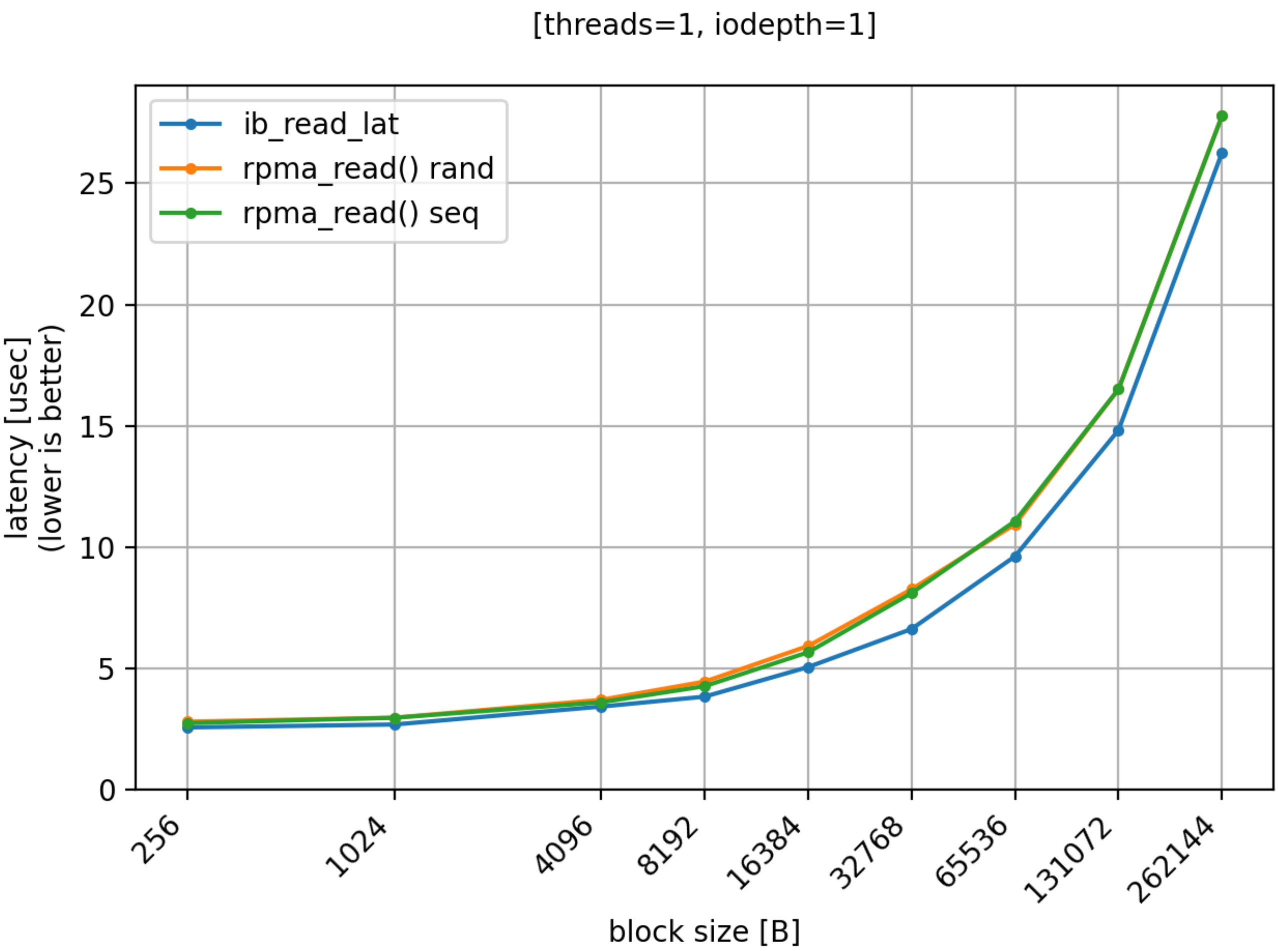
Item	Description
Client - FIO engine configuration	[global] ioengine=librpma_apm_client create_serialize=0 serverip=$serverip port=7204 thread disable_clat=1 lat_percentiles=1 percentile_list=99.0:99.9:99.99:99.999 [client] sync=1 readwrite={read, randread} blocksize=$blocksize ramp_time=15 time_based runtime=60

Figure 1. Latency (lat_avg): ib_read_lat vs rpma_read() from DRAM (both non-DW2PMem)



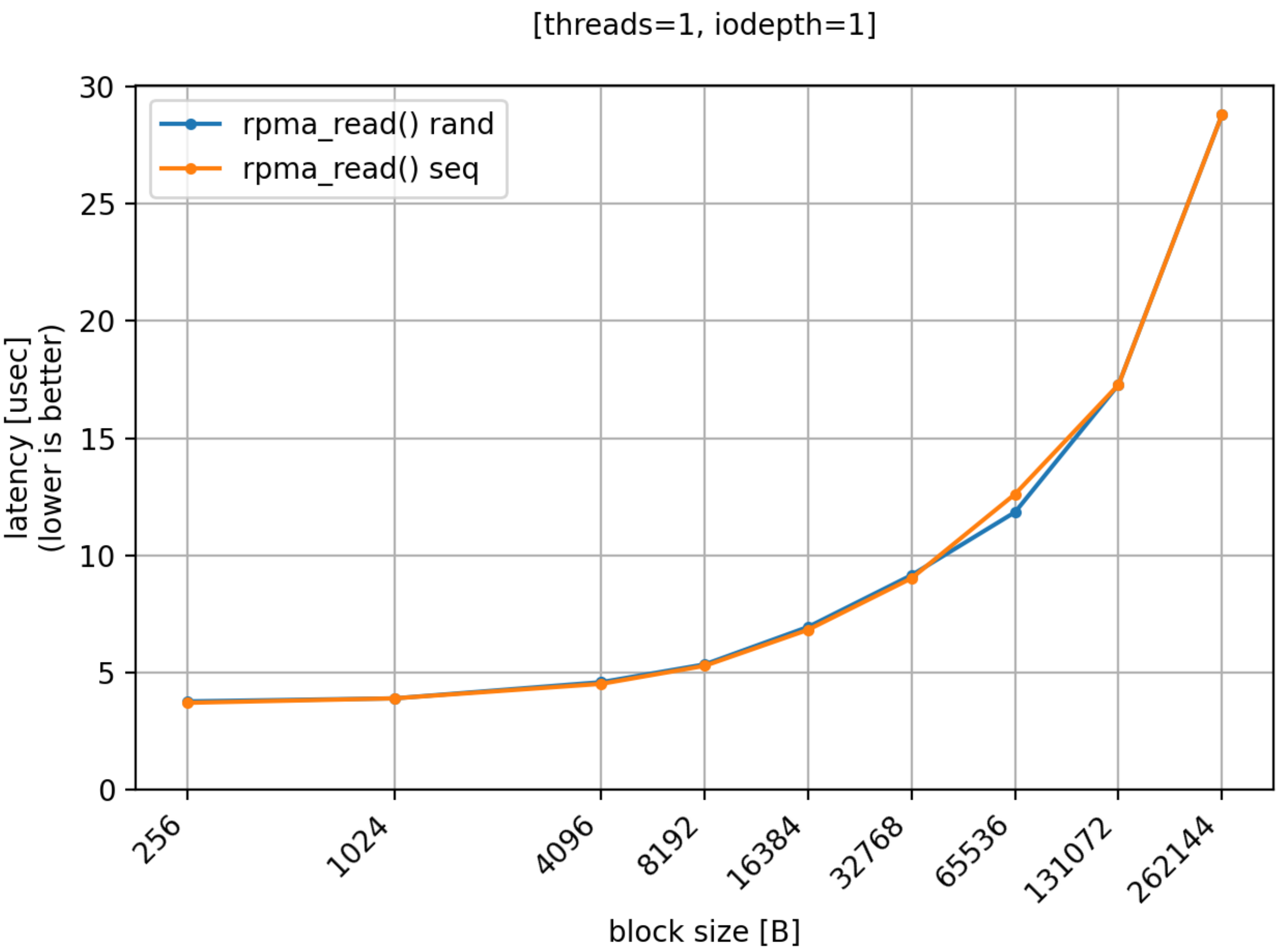
	256	1024	4096	8192	16384	32768	65536	131072	262144
ib_read_lat	1.82	1.93	2.61	3.13	4.31	5.89	8.66	14.19	25.18
rpma_read() rand	1.94	2.11	2.80	3.57	5.06	7.26	9.95	15.45	26.77
rpma_read() seq	1.92	2.11	2.74	3.40	4.78	7.26	9.97	15.47	26.76

Figure 2. Latency (lat_pctl_99.9): ib_read_lat vs rpma_read() from DRAM (both non-DW2PMem)



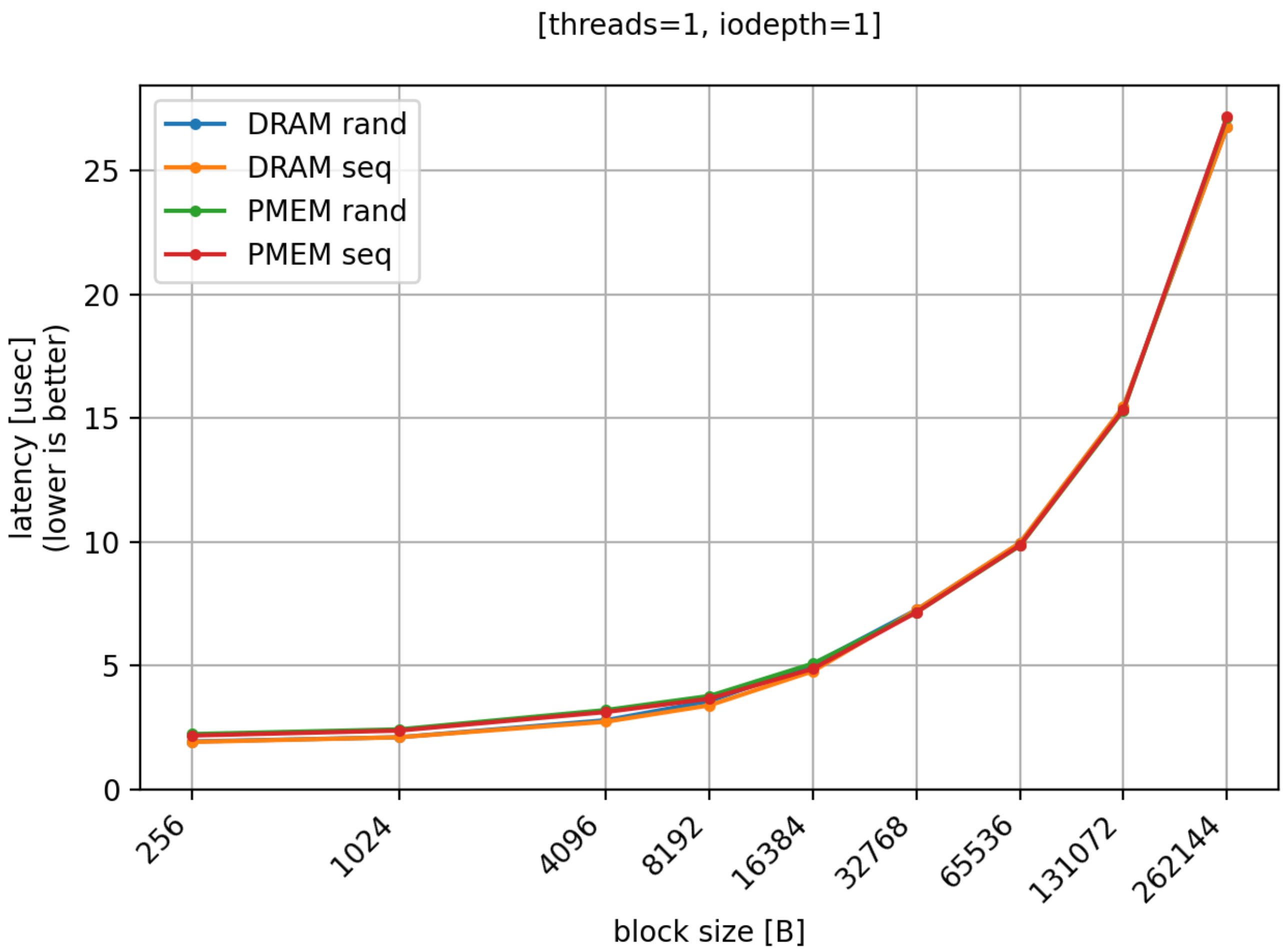
	256	1024	4096	8192	16384	32768	65536	131072	262144
ib_read_lat	2.56	2.68	3.42	3.83	5.05	6.62	9.62	14.82	26.26
rpma_read() rand	2.80	2.96	3.70	4.45	5.92	8.26	10.94	16.51	27.78
rpma_read() seq	2.74	2.96	3.60	4.26	5.66	8.10	11.07	16.51	27.78

Figure 3. Latency (lat_pctl_99.99): ib_read_lat vs rpma_read() from DRAM (both non-DW2PMem)



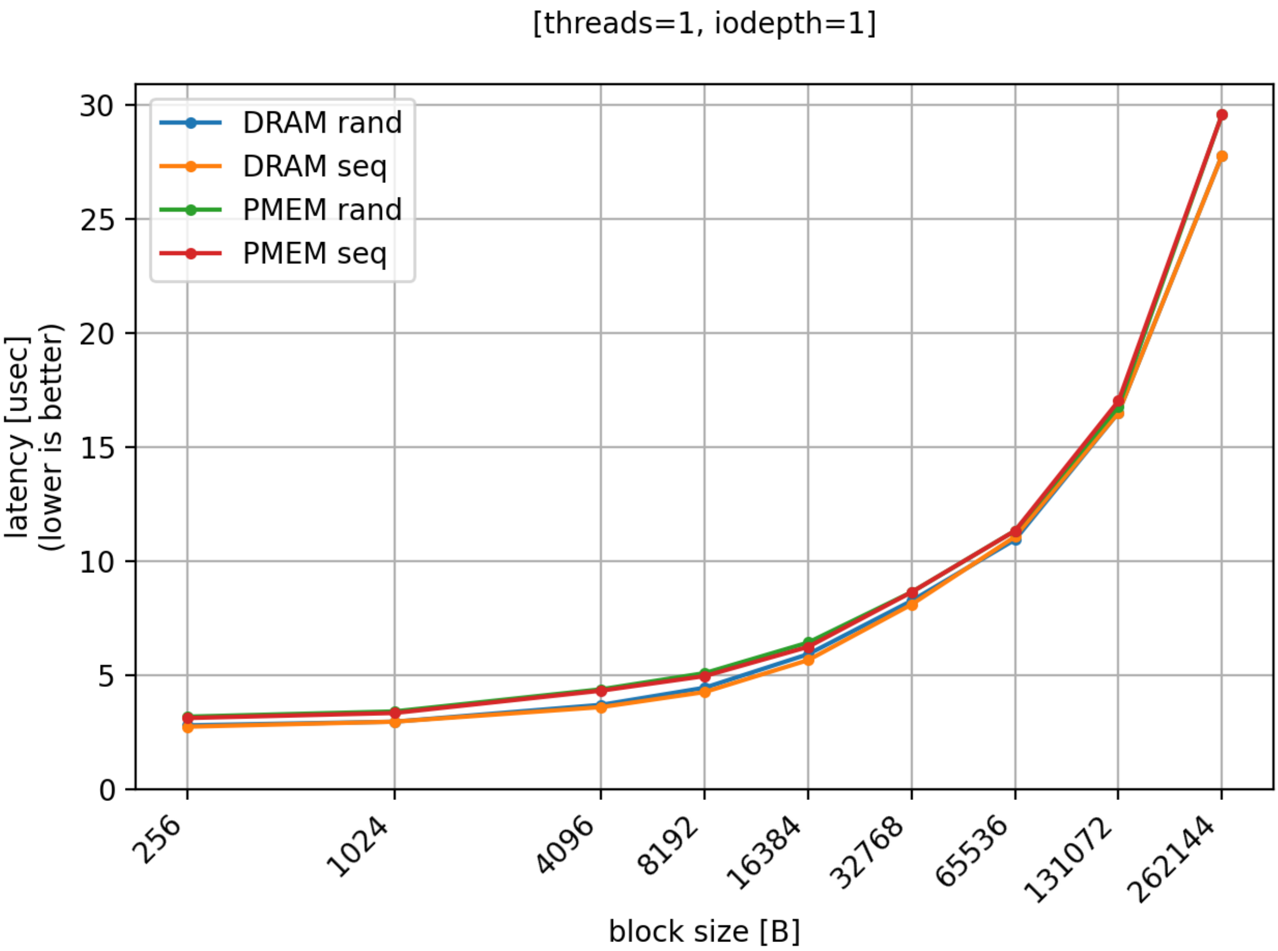
	256	1024	4096	8192	16384	32768	65536	131072	262144
rpma_read() rand	3.76	3.89	4.58	5.34	6.94	9.15	11.84	17.28	28.80
rpma_read() seq	3.70	3.89	4.51	5.28	6.82	9.02	12.61	17.28	28.80

Figure 4. Latency (lat_avg): rpma_read() from DRAM (non-DW2PMem) vs from PMEM (DW2PMem)



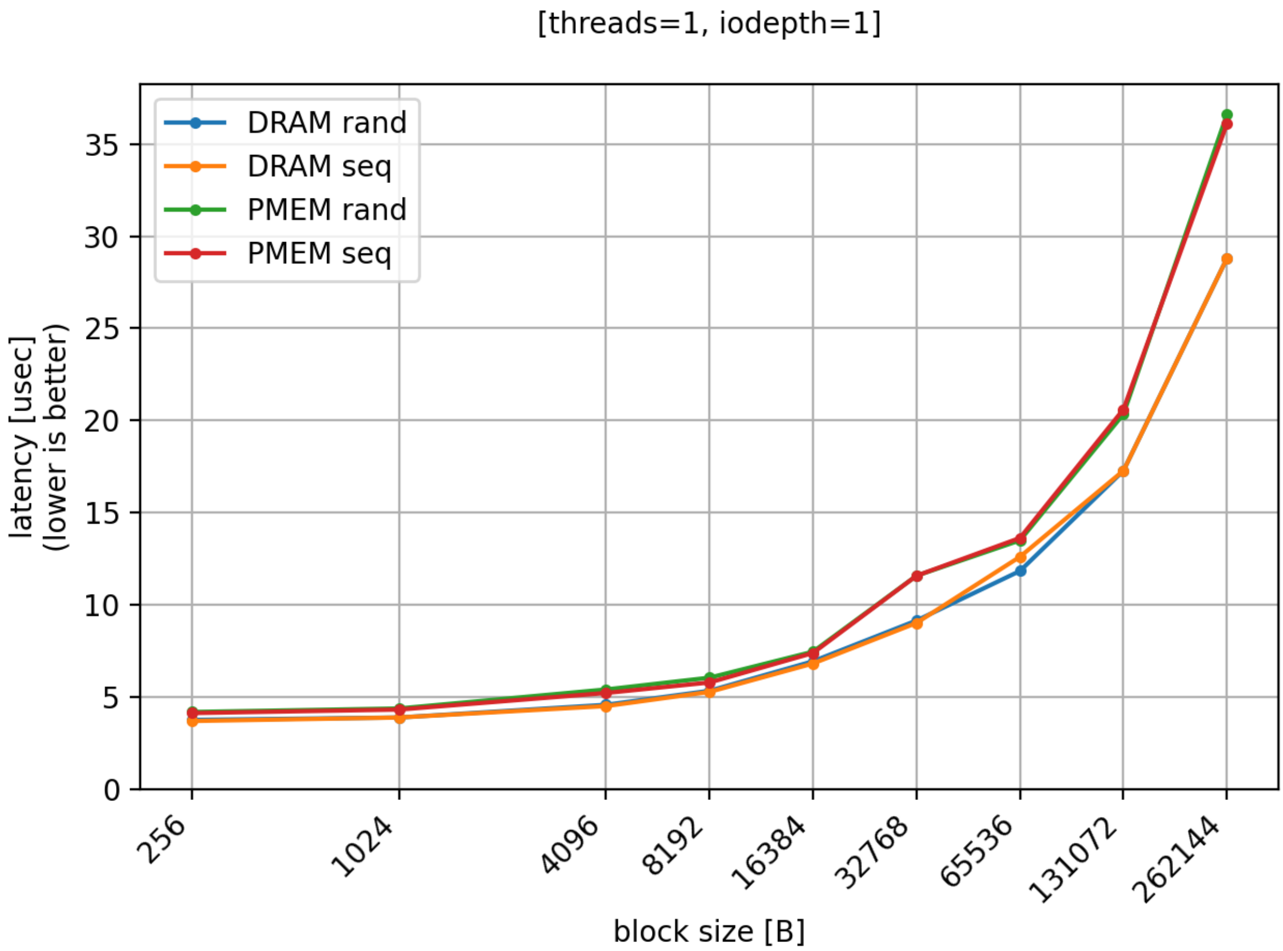
	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	1.94	2.11	2.80	3.57	5.06	7.26	9.95	15.45	26.77
DRAM seq	1.92	2.11	2.74	3.40	4.78	7.26	9.97	15.47	26.76
PMEM rand	2.24	2.43	3.21	3.78	5.09	7.16	9.85	15.32	27.15
PMEM seq	2.18	2.38	3.13	3.67	4.88	7.16	9.87	15.36	27.19

Figure 5. Latency (lat_pctl_99.9): rpma_read() from DRAM (non-DW2PMem) vs from PMEM (DW2PMem)



	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	2.80	2.96	3.70	4.45	5.92	8.26	10.94	16.51	27.78
DRAM seq	2.74	2.96	3.60	4.26	5.66	8.10	11.07	16.51	27.78
PMEM rand	3.18	3.41	4.38	5.09	6.43	8.64	11.33	16.77	29.57
PMEM seq	3.12	3.34	4.32	4.96	6.24	8.64	11.33	17.02	29.57

Figure 6. Latency (lat_pctl_99.99): rpma_read() from DRAM (non-DW2PMem) vs from PMEM (DW2PMem)



	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	3.76	3.89	4.58	5.34	6.94	9.15	11.84	17.28	28.80
DRAM seq	3.70	3.89	4.51	5.28	6.82	9.02	12.61	17.28	28.80
PMEM rand	4.19	4.38	5.41	6.05	7.46	11.58	13.50	20.35	36.61
PMEM seq	4.13	4.32	5.22	5.79	7.39	11.58	13.63	20.61	36.10

Read from PMem: Bandwidth

Comparing the bandwidth of `rpma_read()` from PMem on **the RPMA Target** to the bandwidth of `rpma_read()` from DRAM on **the RPMA Target** using the `ib_read_bw` as the baseline.

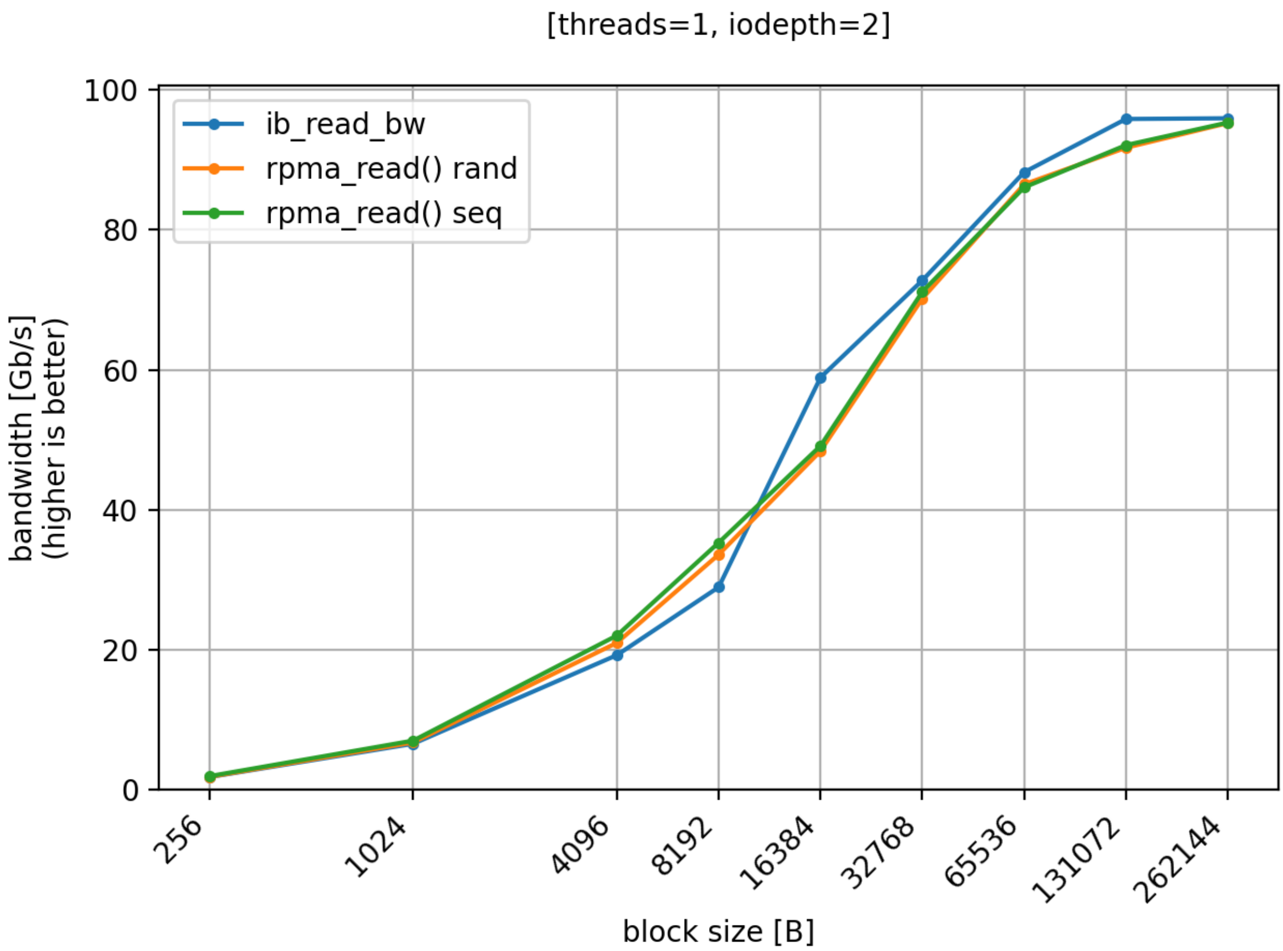
Item	Description
Server - ib_read_bw configuration	`--size $blocksize --qp 1 --tx-depth=2` `--size 4096 --qp $threads --tx-depth=2`

Client - ib_read_bw configuration	`--iters $iters --size $blocksize --qp 1 --tx-depth=2 \`
	`    --report_gbits $serverip`
	`--iters $iters --size 4096 --qp $threads --tx-depth=2 \`
	`    --report_gbits $serverip`

Server - FIO engine configuration	`[global]`
	`ioengine=librpma_apm_server`
	`create_serialize=0`
	`kb_base=1000`
	`serverip=$serverip`
	`port=7204`
	`thread`
	
	`[server]`
	`direct_write_to_pmem={0, 1} # 1 for Device DAX`
	`numjobs=1`
	`size=100MiB`
	`filename={malloc, /dev/daxX.Y} # for DRAM or Device DAX`
	`filename_format=/mnt/pmem/file # for File System DAX`

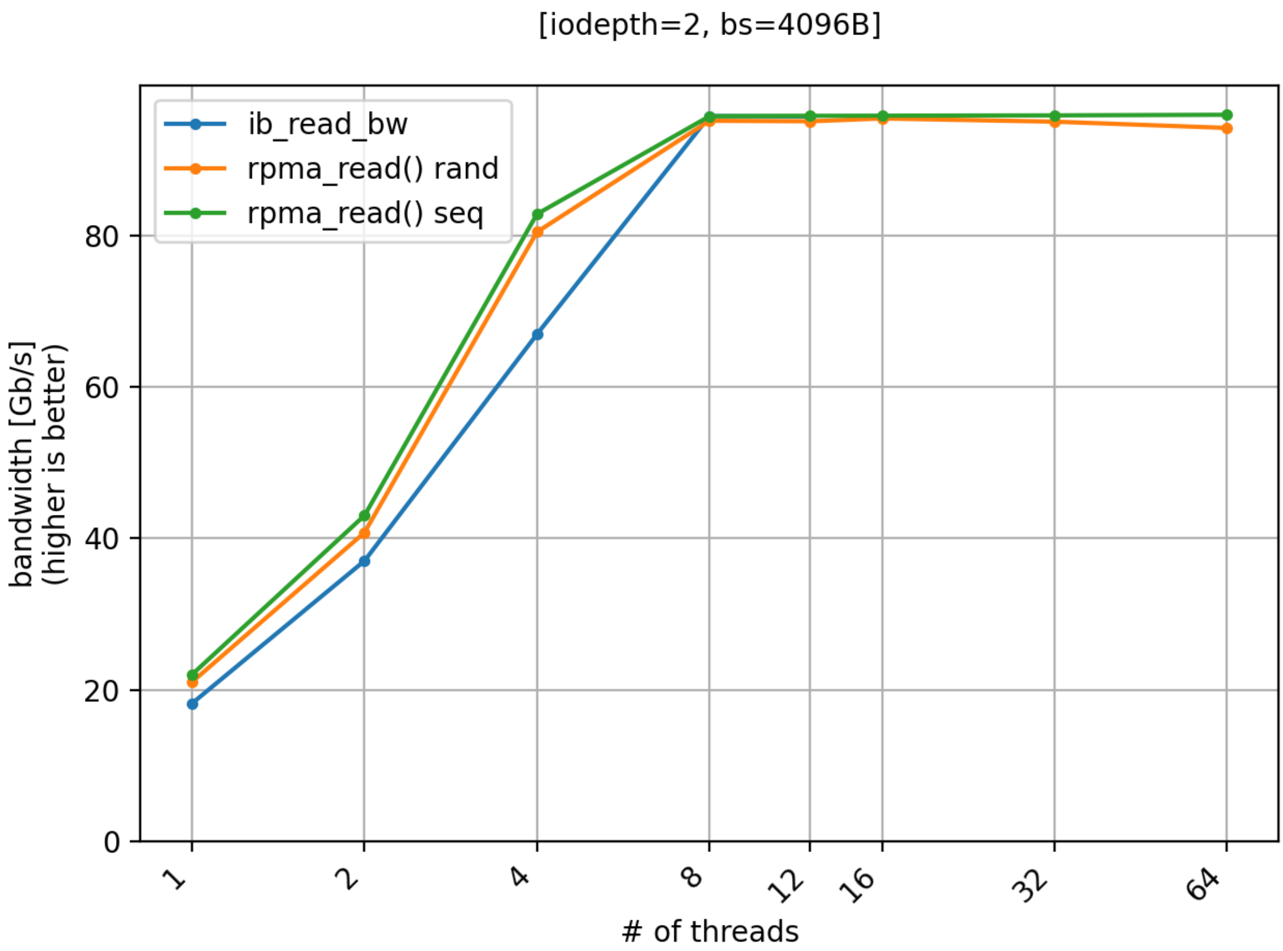
Item	Description
Client - FIO engine configuration	[global] ioengine=librpma_apm_client create_serialize=0 serverip=$serverip port=7204 thread disable_clat=1 lat_percentiles=1 percentile_list=99.0:99.9:99.99:99.999 [client] numjobs=$numjobs group_reporting=1 iodepth=2 readwrite={read, randread} blocksize=$blocksize ramp_time=15 time_based runtime=60

Figure 7. Bandwidth (bs): ib_read_bw() vs rpma_read() from DRAM (both non-DW2PMem)



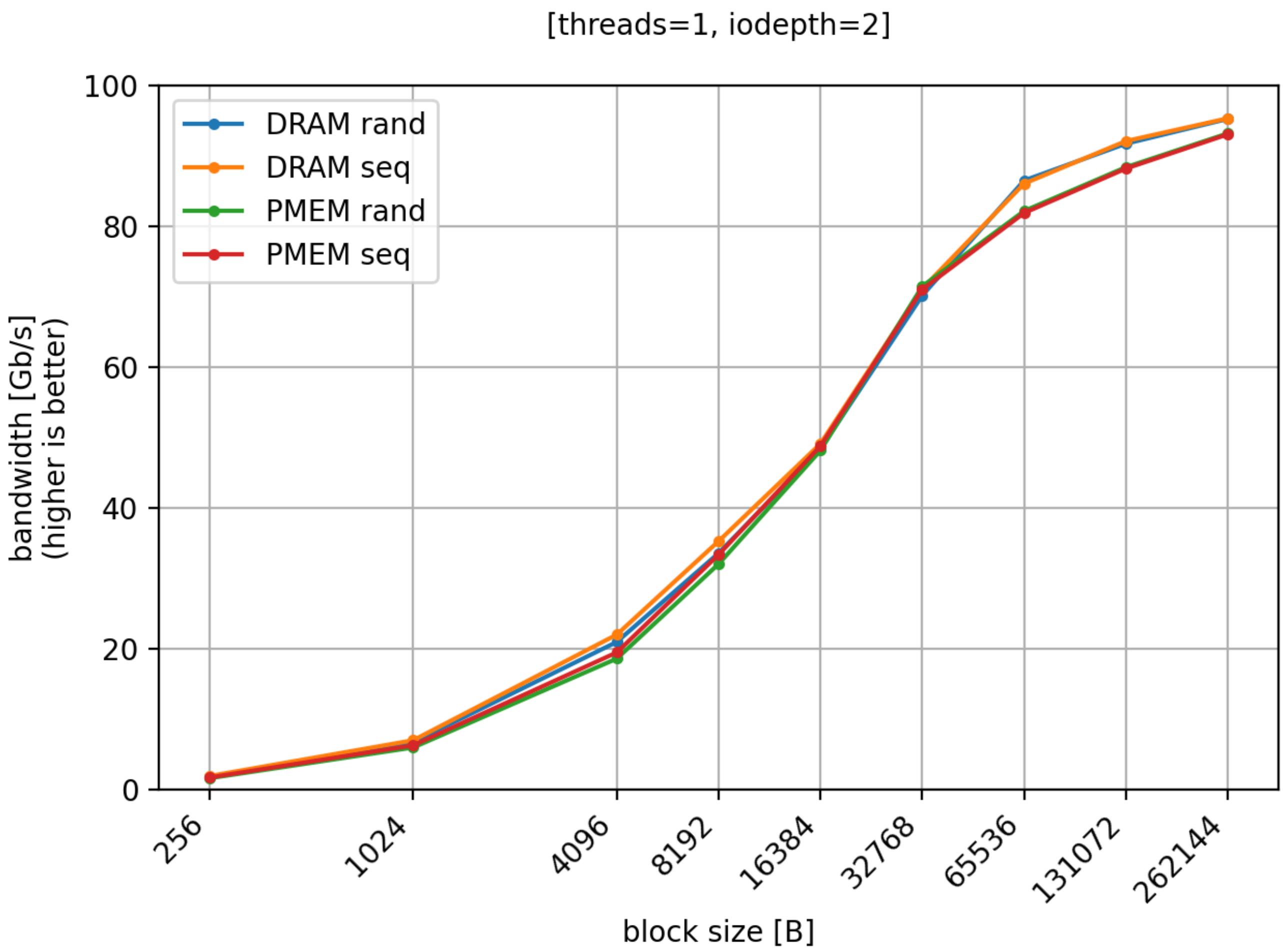
	256	1024	4096	8192	16384	32768	65536	131072	262144
ib_read_bw	1.78	6.54	19.21	28.93	58.90	72.78	88.25	95.86	95.96
rpma_read() rand	1.81	6.73	20.97	33.59	48.39	70.19	86.52	91.75	95.27
rpma_read() seq	1.91	6.99	22.01	35.29	49.13	71.16	86.08	92.12	95.35

Figure 8. Bandwidth (threads): ib_read_bw() vs rpma_read() from DRAM (both non-DW2PMem)



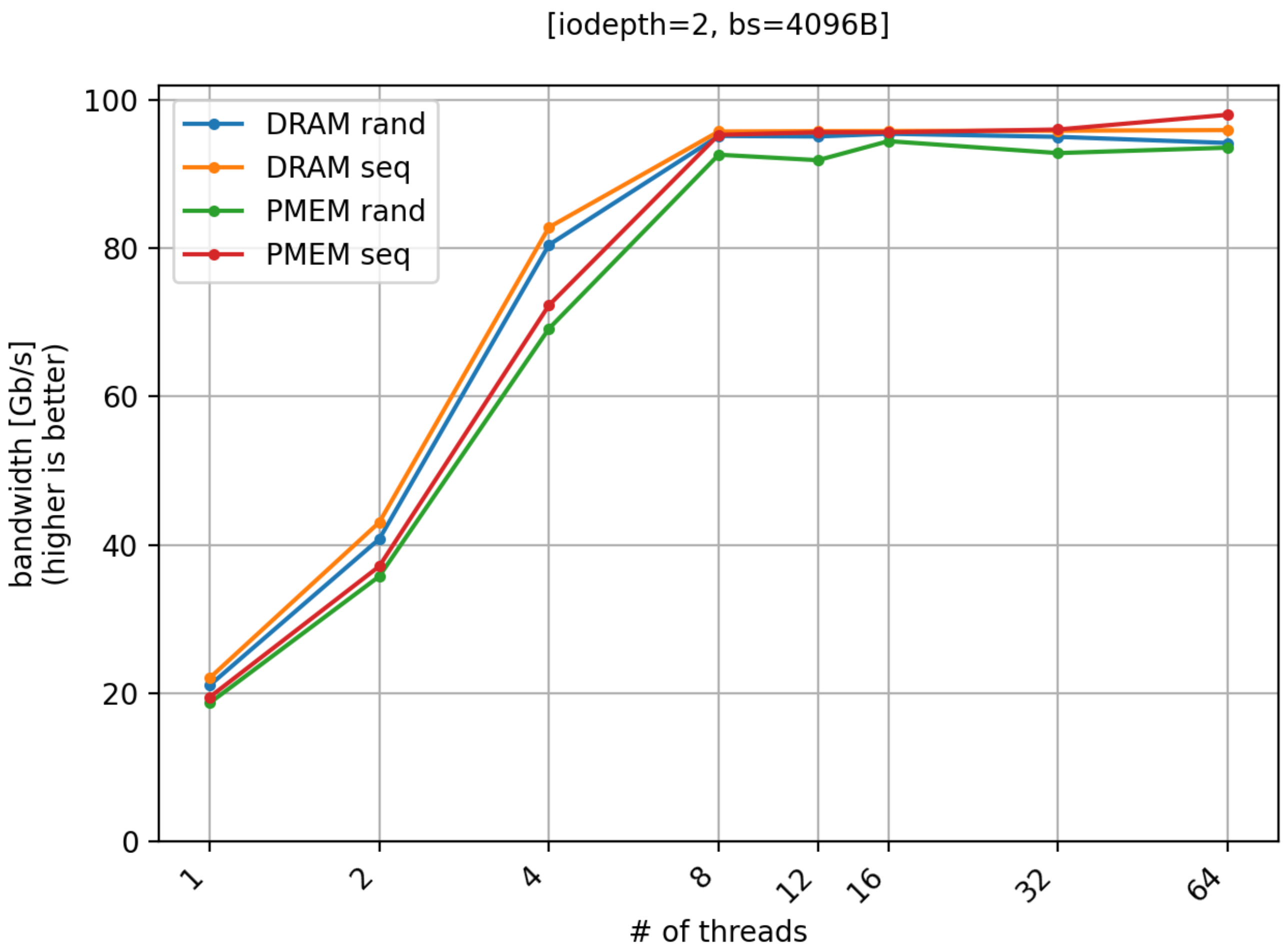
	1	2	4	8	12	16	32	64
ib_read_bw	18.20	37.03	66.96	95.77	95.72	-	-	-
rpma_read() rand	21.02	40.77	80.46	95.15	95.08	95.45	95.03	94.20
rpma_read() seq	22.01	43.02	82.83	95.75	95.80	95.82	95.86	95.94

Figure 9. Bandwidth (bs): rpma_read() from DRAM (non-DW2PMem) vs from PMEM (DW2PMem)



	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	1.81	6.73	20.97	33.59	48.39	70.19	86.52	91.75	95.27
DRAM seq	1.91	6.99	22.01	35.29	49.13	71.16	86.08	92.12	95.35
PMEM rand	1.61	5.97	18.60	32.05	48.17	71.49	82.19	88.41	93.21
PMEM seq	1.70	6.27	19.48	33.38	48.80	71.05	81.91	88.22	93.08

Figure 10. Bandwidth (threads): rpma_read() from DRAM (non-DW2PMem) vs from PMEM (DW2PMem)



	1	2	4	8	12	16	32	64
DRAM rand	21.02	40.77	80.46	95.15	95.08	95.45	95.03	94.20
DRAM seq	22.01	43.02	82.83	95.75	95.80	95.82	95.86	95.94
PMEM rand	18.64	35.74	69.16	92.62	91.88	94.45	92.84	93.57
PMEM seq	19.43	37.09	72.32	95.30	95.64	95.64	96.02	98.01

Write to PMem

Benchmarking two ways of writing data persistently to **the RPMA Target**: *Appliance Persistency Method* (**APM**) and *General Purpose Persistency Method* (**GPSPM**). Where:

- **APM** uses `rpma_flush()` following a sequence of `rpma_write()` operations to provide the remote persistency. This method requires **the RPMA Target** to be capable of *Direct Write to PMem* (**DW2PMem**; for details please see Direct Write to PMem).
- **GPSPM** uses `rpma_send()` and `rpma_recv()` operations for sending requests to **the RPMA Target** to assure persistency of the data written using `rpma_write()`. This method does not require **the RPMA Target** to be capable of *Direct Write to PMem* (**non-DW2PMem**). **The RPMA Target** has to provide a thread handling these requests e.g. persisting the data using the `pmem_persist()` operation. When the persistency of the data is assured the response is sent back using also `rpma_send()` and `rpma_recv()`. Depending on how the thread polls for incoming requests you may distinguish two modes:
  - **GPSPM-RT** where the thread polling for incoming requests busy-wait for them (`busy_wait_polling=1`) and
  - **GPSPM** where the thread schedule to be wakened up when a request will appear (`busy_wait_polling=0`). Picking one of these polling modes over another introduces specific challenges and benefits.

For more details on **APM** and **GPSPM** please see: "Persistent Memory Replication Over Traditional RDMA Part 1: Understanding Remote Persistent Memory" Chapter "Two Remote Replication Methods".

As a baseline is used **APM** to DRAM on **the RPMA Target** with *Direct Write to PMem* capability **disabled** (**non-DW2PMem**). Such a configuration allows benefiting from available caching on **the RPMA Target**. Note that data, in this case, is not written to achieve persistency but to show the limit of what is possible regarding data transmission in general and how big is potential overhead when transmitting data using **APM** or **GPSPM** while both of these provide the remote persistency additionally.

Write to PMem: Latency

Comparing the latency of **APM** to PMem on **the RPMA Target** (**DW2PMem**) vs the latency of **APM** to DRAM on **the RPMA Target** (**non-DW2PMem**) (as a baseline) vs the latency of **GPSPM(-RT)** to PMem on **the RPMA Target** (**non-DW2PMem**).

Item	Description

<table>
<tr><th>Item</th><th>Description</th></tr>
<tr><td>APM Server - FIO engine configuration</td><td><pre>[global]
ioengine=librpma_apm_server
create_serialize=0
kb_base=1000
serverip=$serverip
port=7204
thread

[server]
direct_write_to_pmem={0, 1} # 1 for Device DAX
numjobs=1
size=100MiB
filename={malloc, /dev/daxX.Y} # for DRAM or Device DAX
filename_format=/mnt/pmem/file # for File System DAX</pre></td></tr>
<tr><td>APM Client - FIO engine configuration</td><td><pre>[global]
ioengine=librpma_apm_client
create_serialize=0
serverip=$serverip
port=7204
thread
disable_clat=1
lat_percentiles=1
percentile_list=99.0:99.9:99.99:99.999

[client]
sync=1
readwrite={write, randwrite}
blocksize=$blocksize
ramp_time=15
time_based
runtime=60</pre></td></tr>
</table>

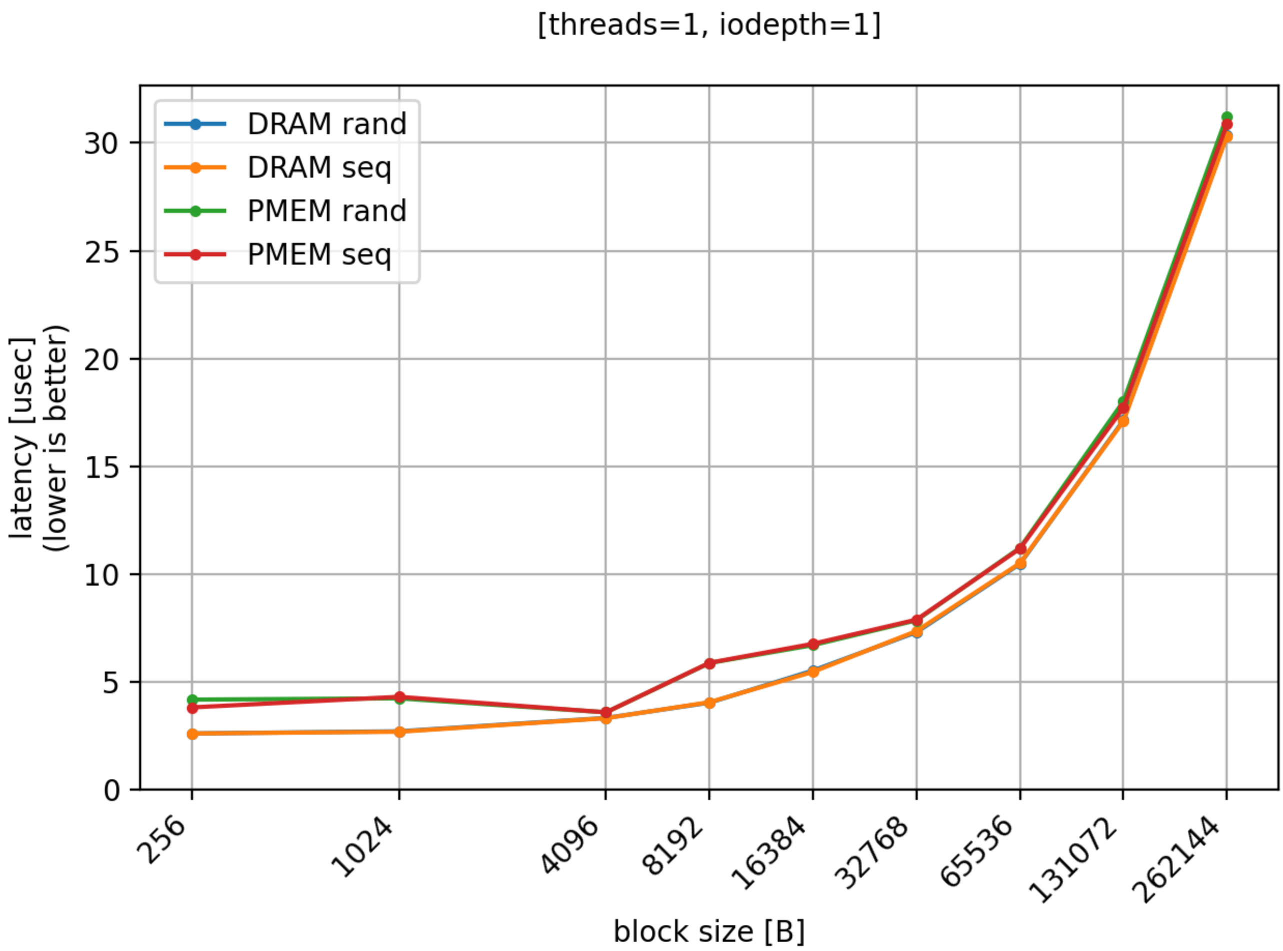
<table>
<tr><th>Item</th><th>Description</th></tr>
<tr><td>GPSPM(-RT) Server - FIO engine configuration</td><td><pre>[global]
ioengine=librpma_gpspm_server
create_serialize=0
kb_base=1000
serverip=$serverip
port=7204
thread

[server]
direct_write_to_pmem=0
numjobs=1
iodepth=1
size=100MiB
filename=/dev/daxX.Y # for Device DAX
filename_format=/mnt/pmem/file # for File System DAX
busy_wait_polling={0, 1} # 1 for GPSPM-RT

time_based
runtime=365d</pre></td></tr>
<tr><td>GPSPM(-RT) Client - FIO engine configuration</td><td><pre>[global]
ioengine=librpma_gpspm_client
create_serialize=0
serverip=$serverip
port=7204
thread
disable_clat=1
lat_percentiles=1
percentile_list=99.0:99.9:99.99:99.999

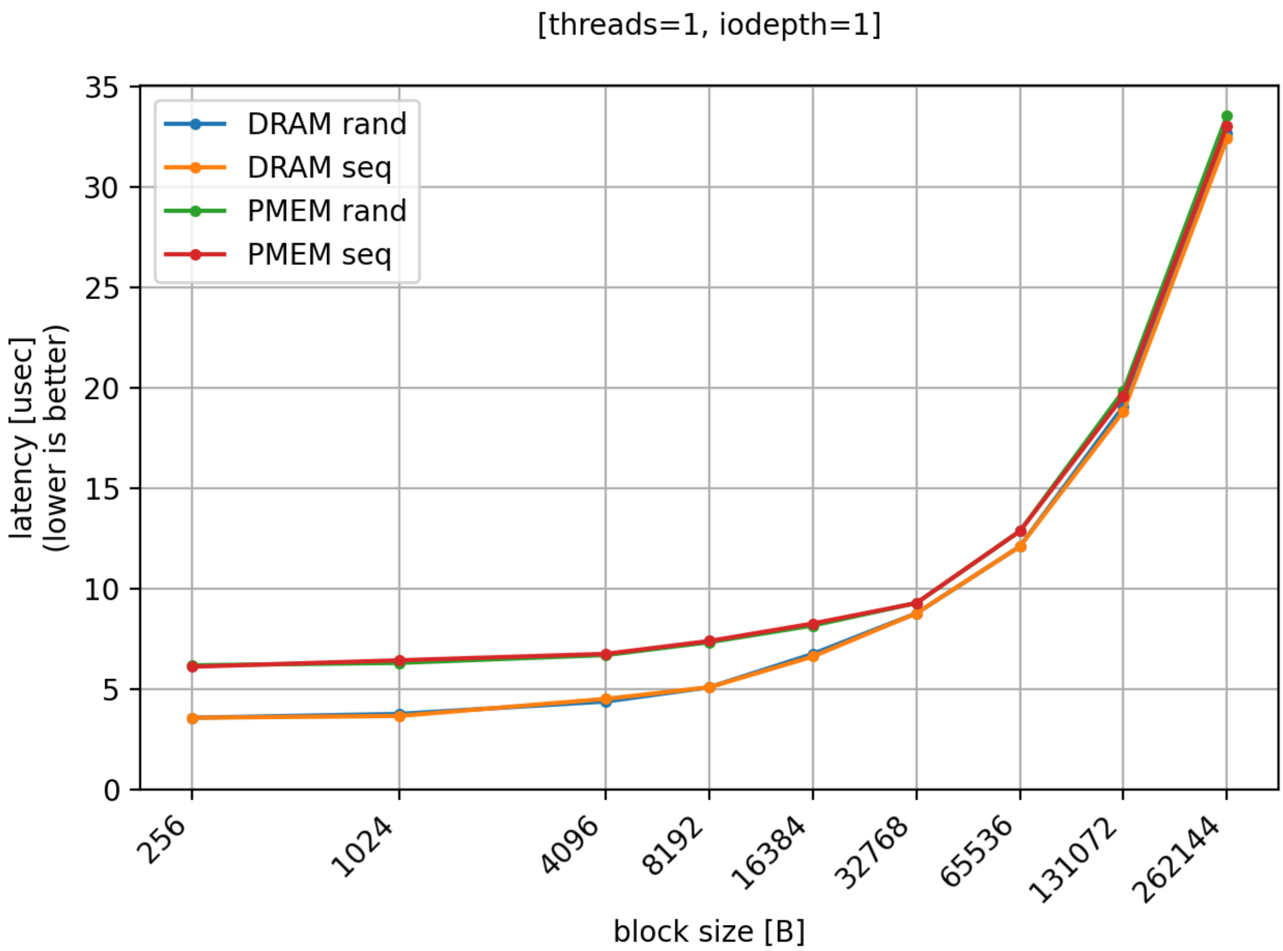
[client]
sync=1
readwrite={write, randwrite}
blocksize=$blocksize
ramp_time=15
time_based
runtime=60</pre></td></tr>
</table>

Figure 11. Latency (lat_avg): APM to DRAM (non-DW2PMem) vs APM to PMEM



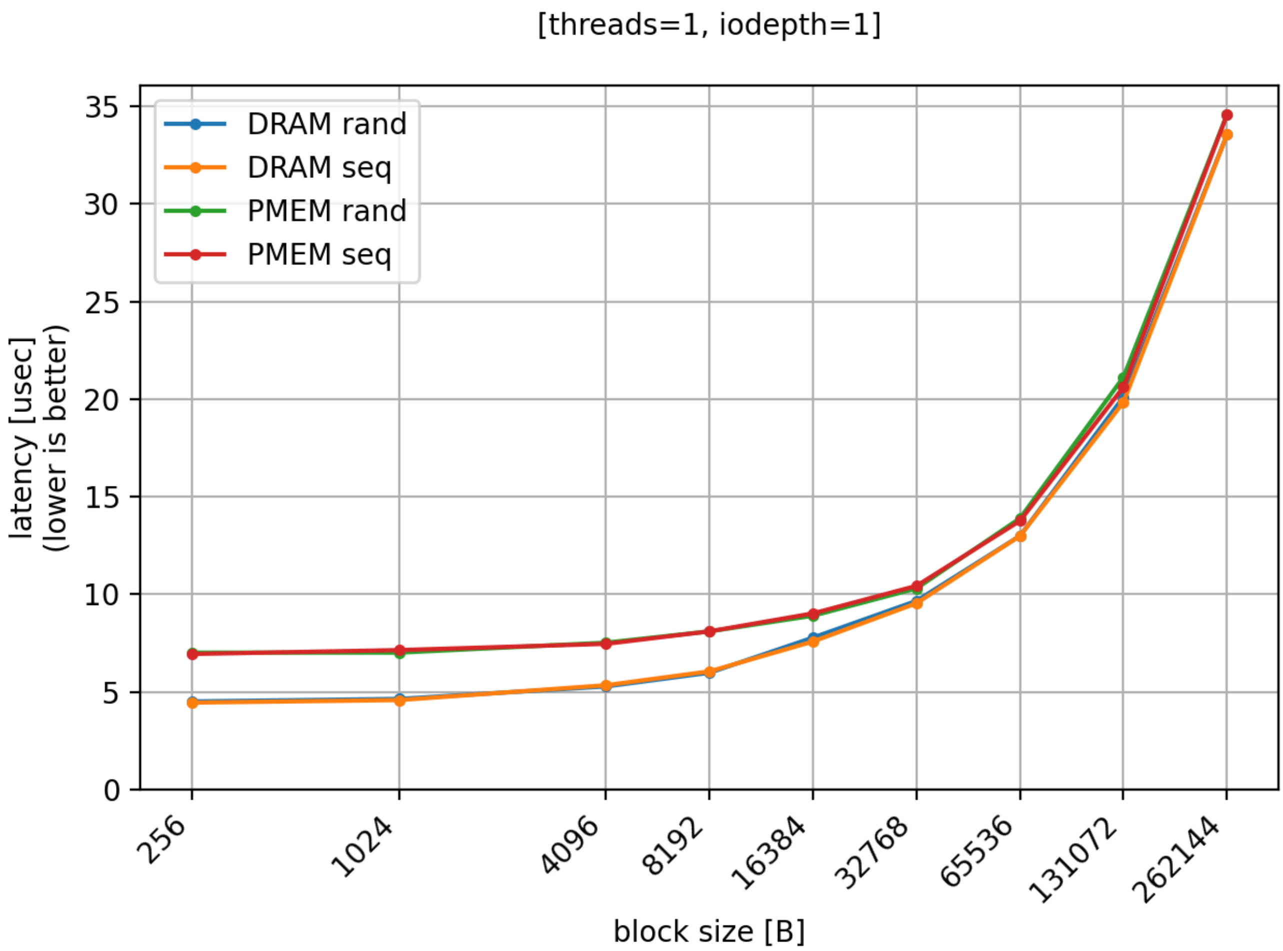
	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	2.60	2.70	3.32	4.03	5.52	7.30	10.47	17.13	30.38
DRAM seq	2.60	2.68	3.31	4.05	5.46	7.35	10.50	17.09	30.33
PMEM rand	4.17	4.23	3.58	5.87	6.70	7.85	11.21	18.01	31.24
PMEM seq	3.81	4.30	3.58	5.88	6.76	7.88	11.18	17.71	30.87

Figure 12. Latency (lat_pctl_99.9): APM to DRAM (non-DW2PMem) vs APM to PMEM



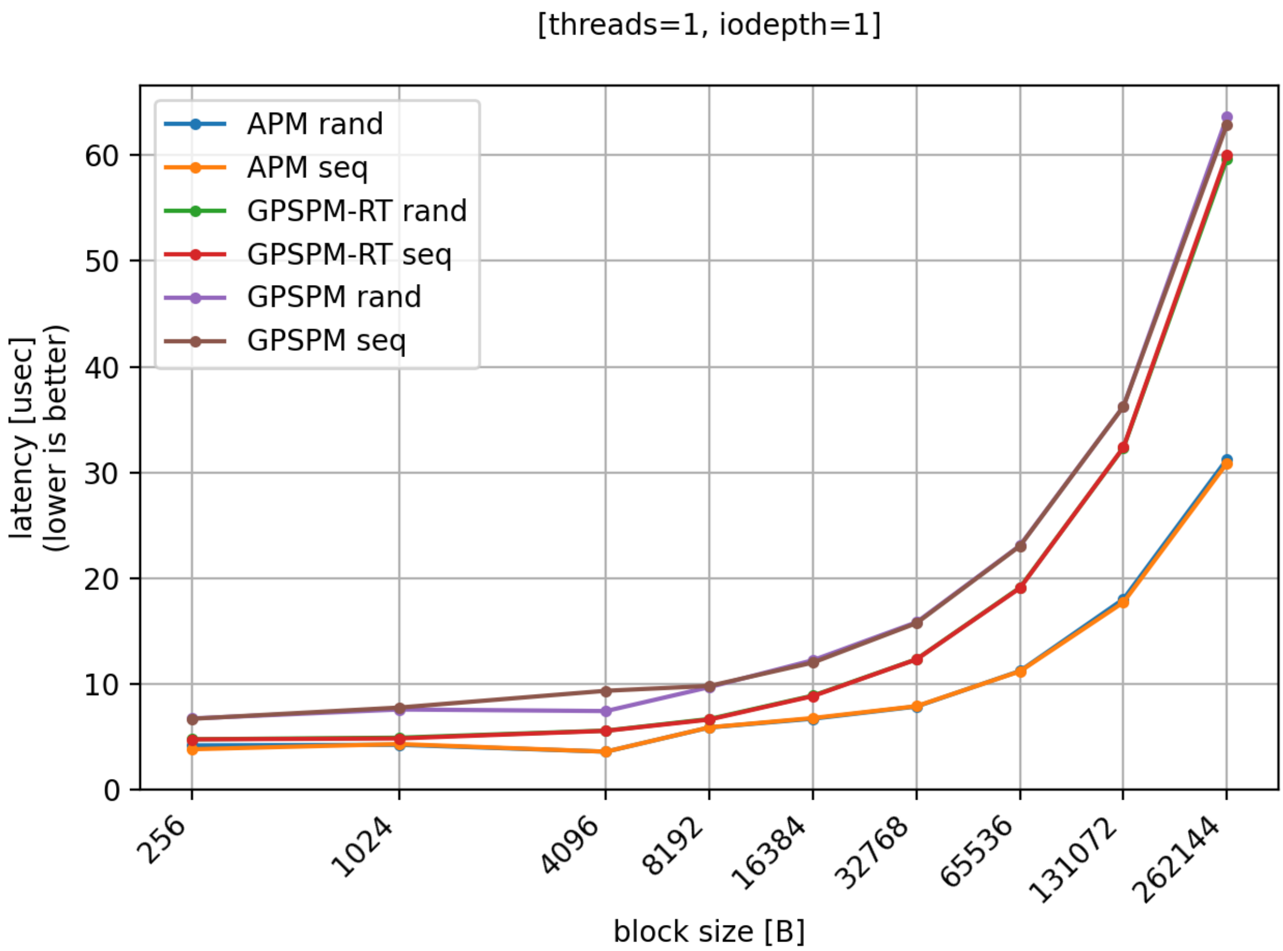
	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	3.57	3.76	4.38	5.09	6.75	8.77	12.10	19.07	32.64
DRAM seq	3.57	3.66	4.51	5.09	6.62	8.77	12.10	18.82	32.38
PMEM rand	6.18	6.30	6.69	7.33	8.16	9.28	12.86	19.84	33.54
PMEM seq	6.11	6.43	6.75	7.39	8.26	9.28	12.86	19.58	33.02

Figure 13. Latency (lat_pctl_99.99): APM to DRAM (non-DW2PMem) vs APM to PMEM



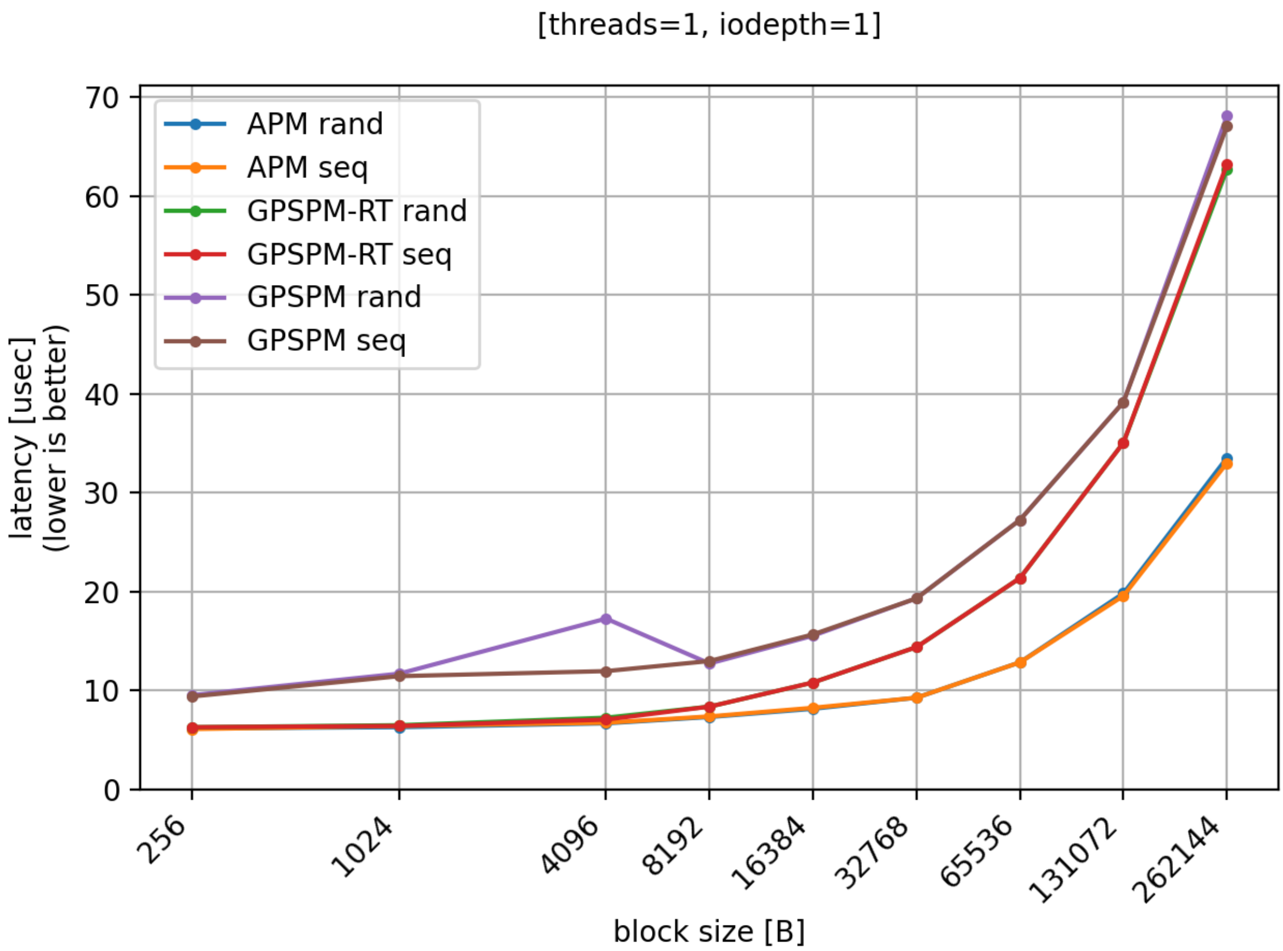
	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	4.51	4.64	5.28	5.98	7.78	9.66	12.99	20.10	33.54
DRAM seq	4.45	4.58	5.34	6.05	7.58	9.54	12.99	19.84	33.54
PMEM rand	7.01	7.01	7.52	8.10	8.90	10.30	13.89	21.12	34.56
PMEM seq	6.94	7.14	7.46	8.10	9.02	10.43	13.76	20.61	34.56

Figure 14. Latency (lat_avg): APM to PMEM vs GPSPM(-RT) to PMEM



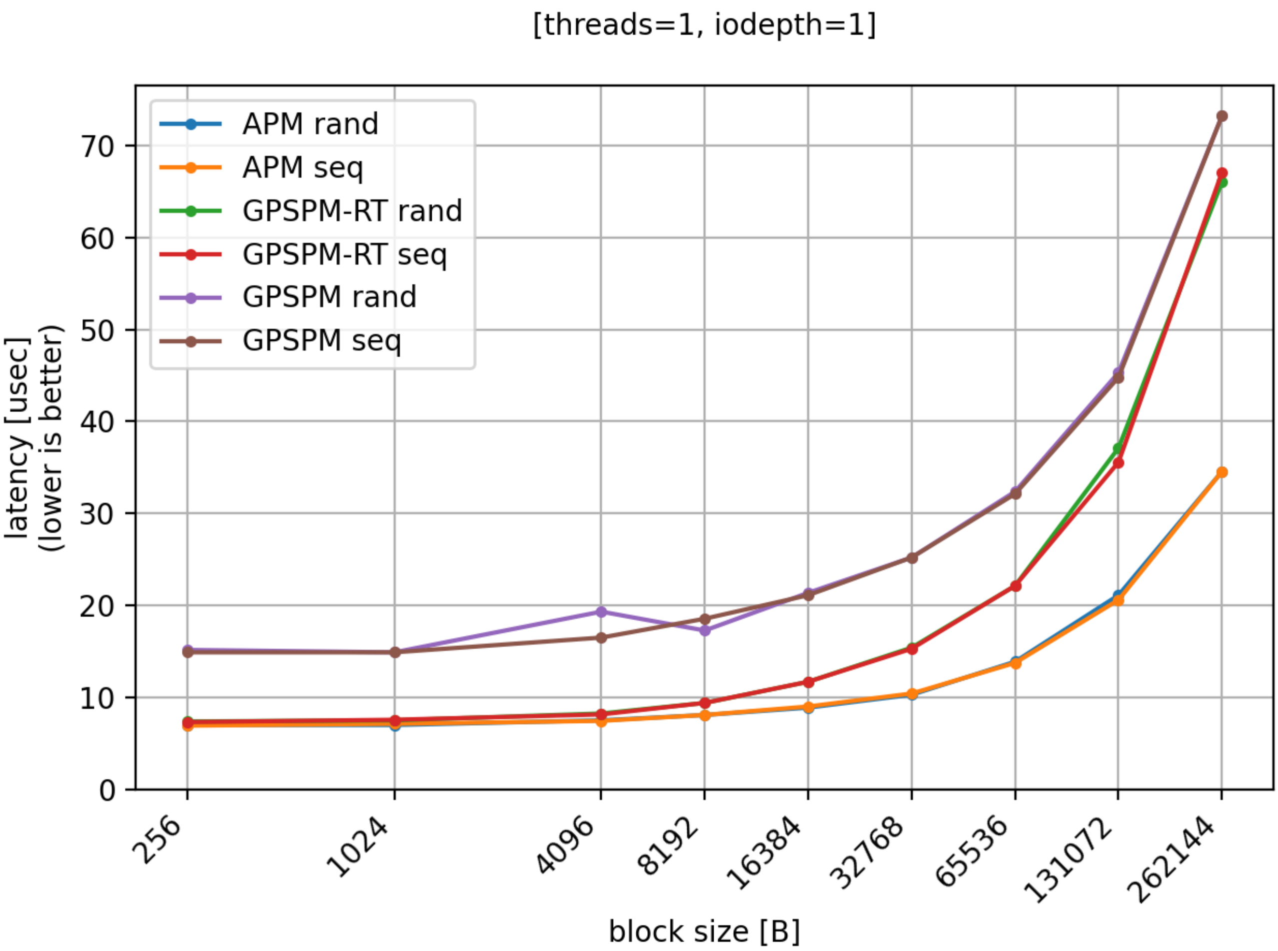
	256	1024	4096	8192	16384	32768	65536	131072	262144
APM rand	4.17	4.23	3.58	5.87	6.70	7.85	11.21	18.01	31.24
APM seq	3.81	4.30	3.58	5.88	6.76	7.88	11.18	17.71	30.87
GPSPM-RT rand	4.75	4.90	5.57	6.67	8.88	12.32	19.10	32.33	59.67
GPSPM-RT seq	4.74	4.83	5.55	6.62	8.84	12.31	19.06	32.37	60.06
GPSPM rand	6.72	7.57	7.42	9.69	12.23	15.82	23.08	36.24	63.61
GPSPM seq	6.69	7.75	9.34	9.80	12.01	15.74	23.04	36.23	62.92

Figure 15. Latency (lat_pctl_99.9): APM to PMEM vs GPSPM(-RT) to PMEM



	256	1024	4096	8192	16384	32768	65536	131072	262144
APM rand	6.18	6.30	6.69	7.33	8.16	9.28	12.86	19.84	33.54
APM seq	6.11	6.43	6.75	7.39	8.26	9.28	12.86	19.58	33.02
GPSPM-RT rand	6.30	6.50	7.26	8.38	10.82	14.40	21.38	35.07	62.72
GPSPM-RT seq	6.30	6.43	7.07	8.38	10.82	14.40	21.38	35.07	63.23
GPSPM rand	9.54	11.71	17.28	12.74	15.55	19.33	27.26	39.17	68.10
GPSPM seq	9.41	11.46	11.97	12.99	15.68	19.33	27.26	39.17	67.07

Figure 16. Latency (lat_pctl_99.99): APM to PMEM vs GPSPM(-RT) to PMEM



	256	1024	4096	8192	16384	32768	65536	131072	262144
APM rand	7.01	7.01	7.52	8.10	8.90	10.30	13.89	21.12	34.56
APM seq	6.94	7.14	7.46	8.10	9.02	10.43	13.76	20.61	34.56
GPSPM-RT rand	7.39	7.52	8.26	9.41	11.71	15.42	22.14	37.12	66.05
GPSPM-RT seq	7.33	7.58	8.16	9.41	11.71	15.30	22.14	35.58	67.07
GPSPM rand	15.17	14.91	19.33	17.28	21.38	25.22	32.38	45.31	73.22
GPSPM seq	14.91	14.91	16.51	18.56	21.12	25.22	32.13	44.80	73.22

Write to PMem: Bandwidth

Comparing the bandwidth of **APM** to PMem on **the RPMA Target** (**DW2PMem**) vs the bandwidth of **APM** to DRAM on **the RPMA Target** (**non-DW2PMem**) (as a baseline) vs the bandwidth of **GPSPM(-RT)** to PMem on **the RPMA Target** (**non-DW2PMem**).

Item	Description

APM Server - FIO engine configuration	`[global]`
	`ioengine=librpma_apm_server`
	`create_serialize=0`
	`kb_base=1000`
	`serverip=$serverip`
	`port=7204`
	`thread`
	
	`[server]`
	`direct_write_to_pmem=1`
	`numjobs=$numjobs`
	`size=100MiB`
	`filename={malloc, /dev/daxX.Y} # for DRAM or Device DAX`
	`filename_format=/mnt/pmem/file # for File System DAX`

APM Client - FIO engine configuration	`[global]`
	`ioengine=librpma_apm_client`
	`create_serialize=0`
	`serverip=$serverip`
	`port=7204`
	`thread`
	`disable_clat=1`
	`lat_percentiles=1`
	`percentile_list=99.0:99.9:99.99:99.999`
	
	`[client]`
	`numjobs=$numjobs`
	`group_reporting=1`
	`iodepth=2`
	`readwrite={write, randwrite}`
	`blocksize=$blocksize`
	`ramp_time=15`
	`time_based`
	`runtime=60`

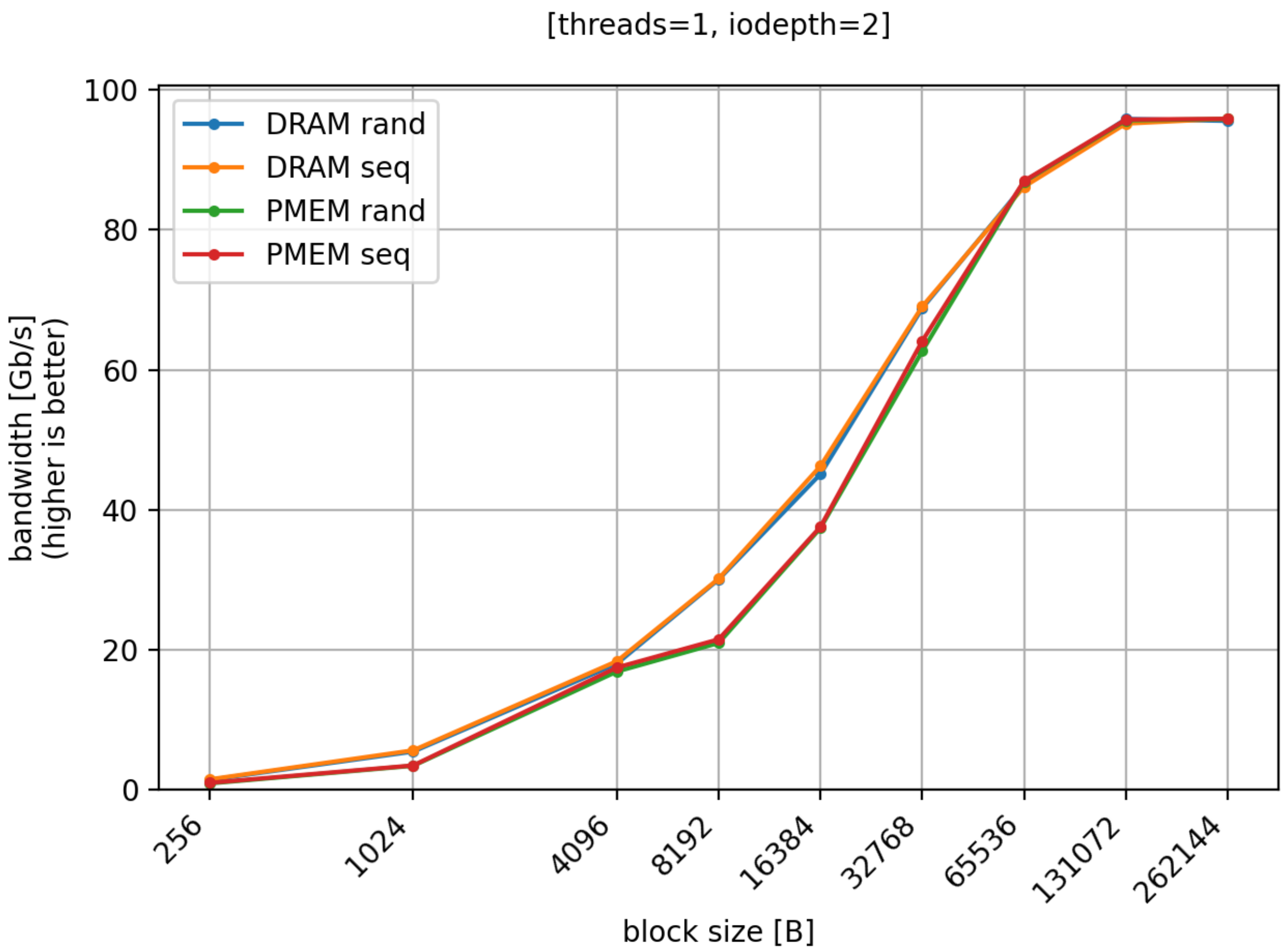
<table>
<tr><th>Item</th><th>Description</th></tr>
<tr><td>GPSPM(-RT) Server - FIO engine configuration</td><td><pre>[global]
ioengine=librpma_gpspm_server
create_serialize=0
kb_base=1000
serverip=$serverip
port=7204
thread

[server]
direct_write_to_pmem=0
numjobs=$numjobs
iodepth=2
size=100MiB
filename=/dev/daxX.Y # for Device DAX
filename_format=/mnt/pmem/file # for File System DAX
busy_wait_polling={0, 1} # 1 for GPSPM-RT

time_based
runtime=365d</pre></td></tr>
<tr><td>GPSPM(-RT) Client - FIO engine configuration</td><td><pre>[global]
ioengine=librpma_gpspm_client
create_serialize=0
serverip=$serverip
port=7204
thread
disable_clat=1
lat_percentiles=1
percentile_list=99.0:99.9:99.99:99.999

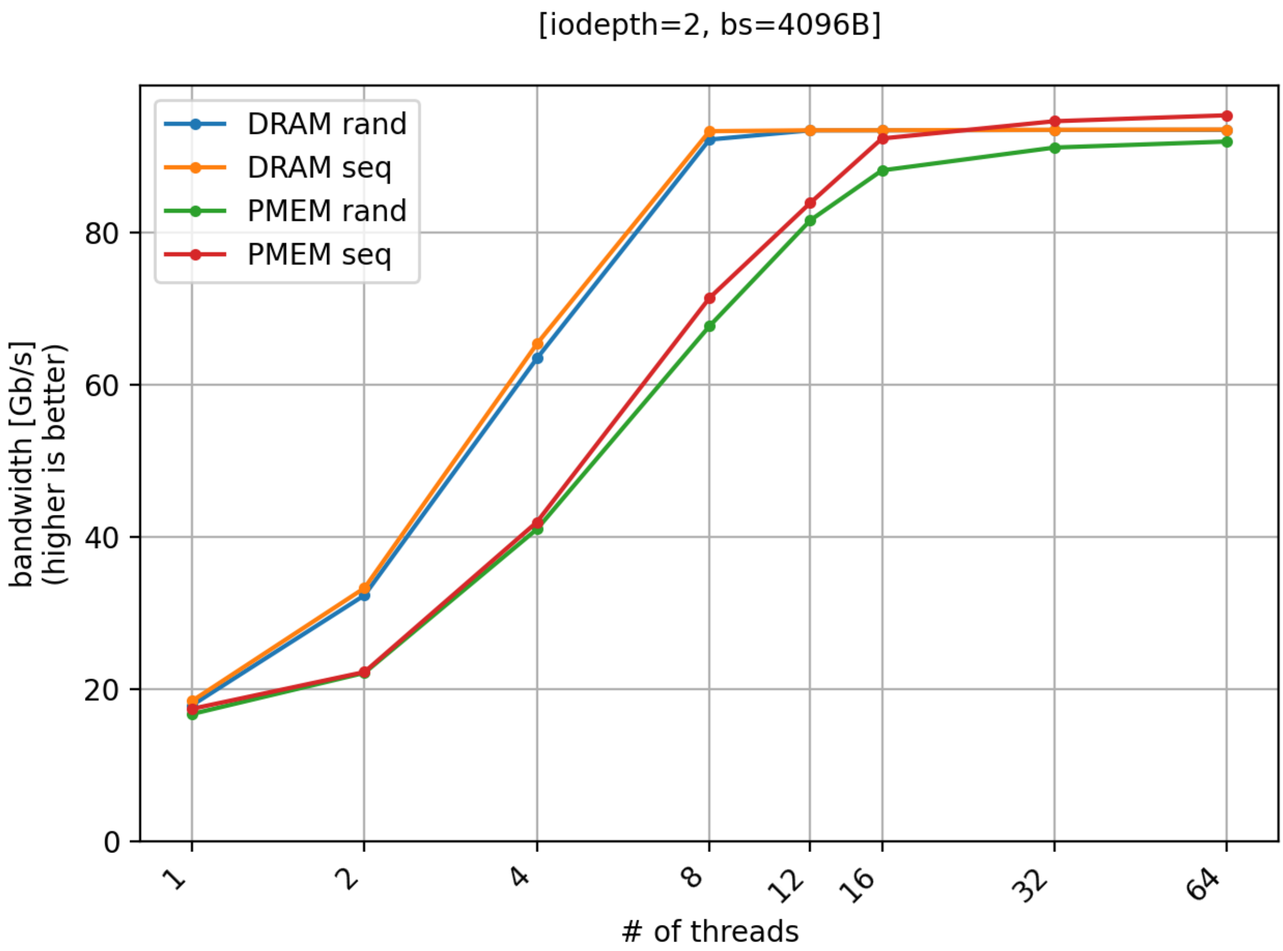
[client]
numjobs=$numjobs
group_reporting=1
iodepth=2
readwrite={write, randwrite}
blocksize=$blocksize
ramp_time=15
time_based
runtime=60</pre></td></tr>
</table>

Figure 17. Bandwidth (bs): APM to DRAM (non-DW2PMem) vs APM to PMEM



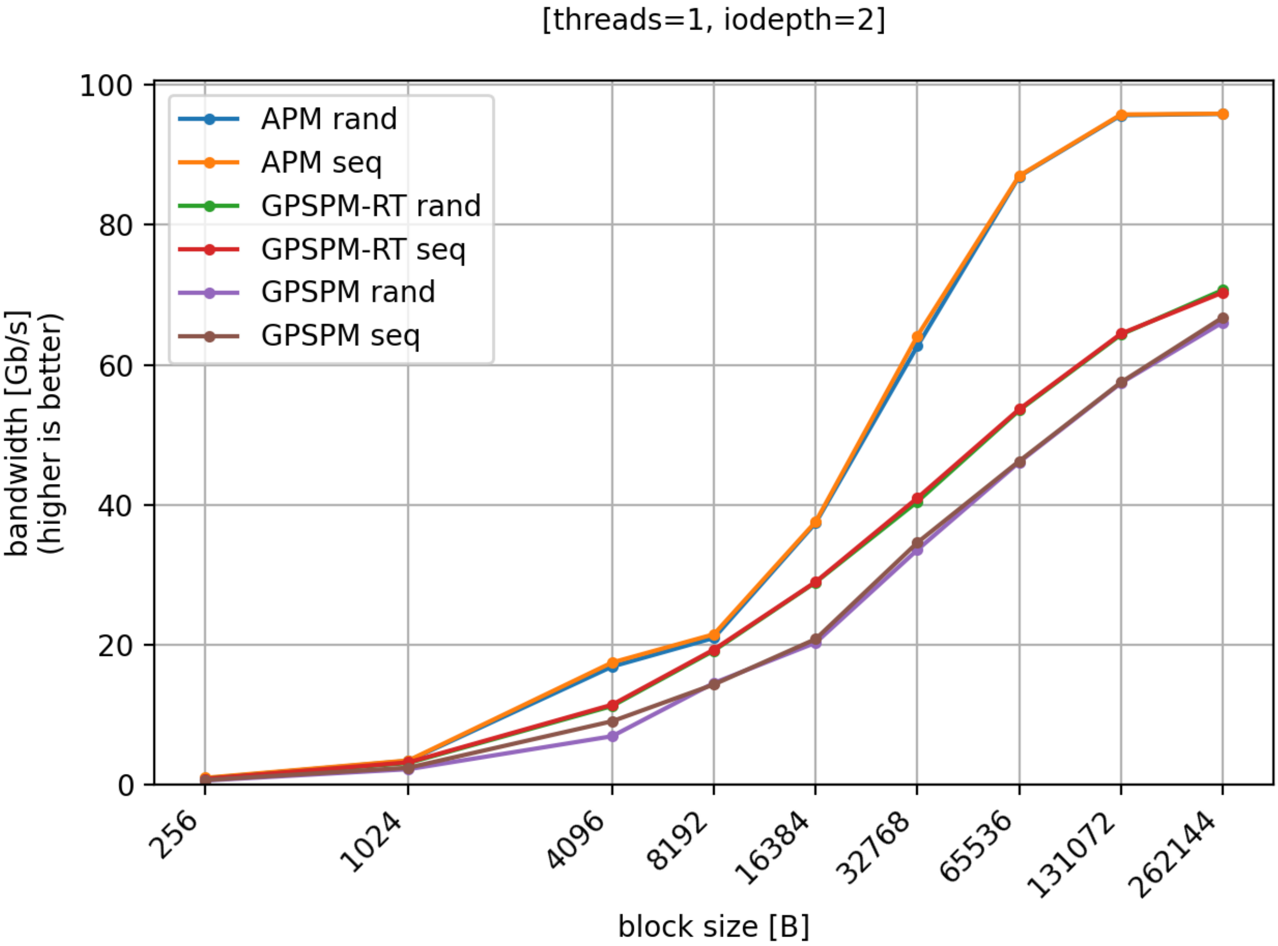
	256	1024	4096	8192	16384	32768	65536	131072	262144
DRAM rand	1.38	5.42	17.99	30.04	45.10	68.83	86.44	95.87	95.59
DRAM seq	1.45	5.60	18.34	30.15	46.25	69.07	86.19	95.20	95.92
PMEM rand	0.87	3.39	16.86	20.95	37.39	62.69	86.94	95.67	95.86
PMEM seq	0.95	3.44	17.44	21.47	37.57	64.14	87.02	95.78	95.89

Figure 18. Bandwidth (threads): APM to DRAM (non-DW2PMem) vs APM to PMEM



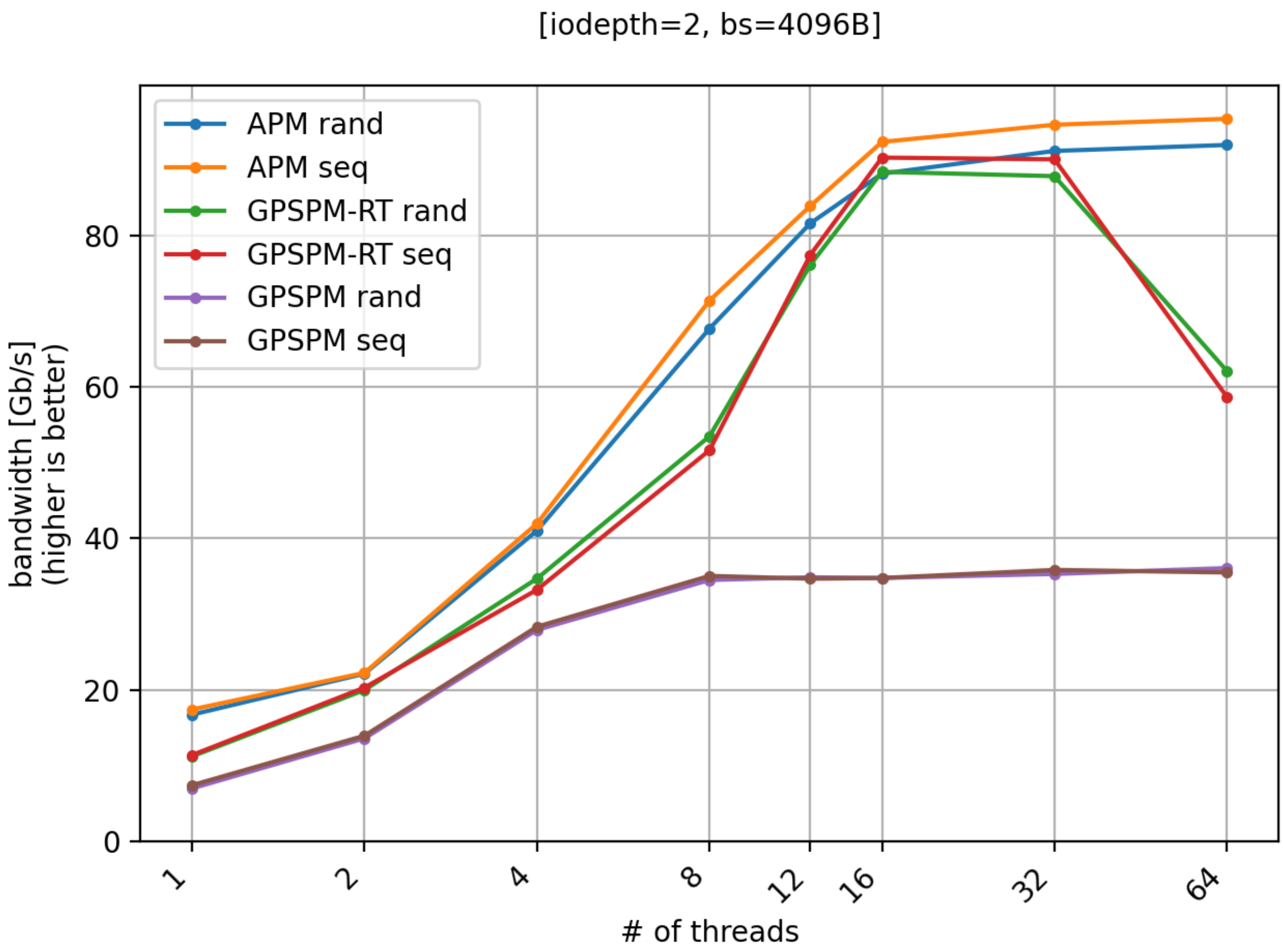
	1	2	4	8	12	16	32	64
DRAM rand	17.86	32.32	63.45	92.19	93.40	93.41	93.45	93.46
DRAM seq	18.43	33.27	65.36	93.29	93.41	93.41	93.47	93.51
PMEM rand	16.71	22.12	41.01	67.68	81.59	88.14	91.14	91.93
PMEM seq	17.39	22.23	41.94	71.38	83.88	92.33	94.60	95.38

Figure 19. Bandwidth (bs): APM to PMEM vs GPSPM(-RT) to PMEM



	256	1024	4096	8192	16384	32768	65536	131072	262144
APM rand	0.87	3.39	16.86	20.95	37.39	62.69	86.94	95.67	95.86
APM seq	0.95	3.44	17.44	21.47	37.57	64.14	87.02	95.78	95.89
GPSPM-RT rand	0.80	3.08	11.22	19.14	28.91	40.45	53.51	64.32	70.69
GPSPM-RT seq	0.80	3.17	11.40	19.29	28.97	40.97	53.65	64.45	70.33
GPSPM rand	0.57	2.19	6.90	14.51	20.27	33.56	46.08	57.40	66.07
GPSPM seq	0.63	2.40	9.04	14.30	20.79	34.59	46.23	57.49	66.77

Figure 20. Bandwidth (threads): APM to PMEM vs GPSPM(-RT) to PMEM



	1	2	4	8	12	16	32	64
APM rand	16.71	22.12	41.01	67.68	81.59	88.14	91.14	91.93
APM seq	17.39	22.23	41.94	71.38	83.88	92.33	94.60	95.38
GPSPM-RT rand	11.20	19.96	34.68	53.46	76.11	88.38	87.81	62.08
GPSPM-RT seq	11.37	20.28	33.20	51.61	77.42	90.26	90.03	58.66
GPSPM rand	6.97	13.58	27.90	34.49	34.85	34.76	35.29	36.06
GPSPM seq	7.39	13.93	28.32	35.03	34.66	34.75	35.82	35.48

Ref: 2022_02_15_CASCADE_LAKE