CSC

ICT Solutions for Brilliant Minds

Studying Elbencho benchmark on multiple GPU architectures HPC-IODC Workshop

July 2nd, 2021

George S. Markomanolis

Lead HPC Scientist, CSC – IT For Science Ltd.

Outline

- LUMI
- ROCm and HIP
- Benchmarking Elbenco on AMD and NVIDIA GPUs

Disclaimer

- AMD ecosystem is under heavy development, many things can change without notice
- I am **NOT** affiliated with the Elbencho benchmark
- Some results are really fresh and investigating the outcome, this is still under development
- Could not find AMD MI100 node with NVMe
- Preparing the benchmarks but the technology will be different on the actual LUMI
- No GPU Direct Storage

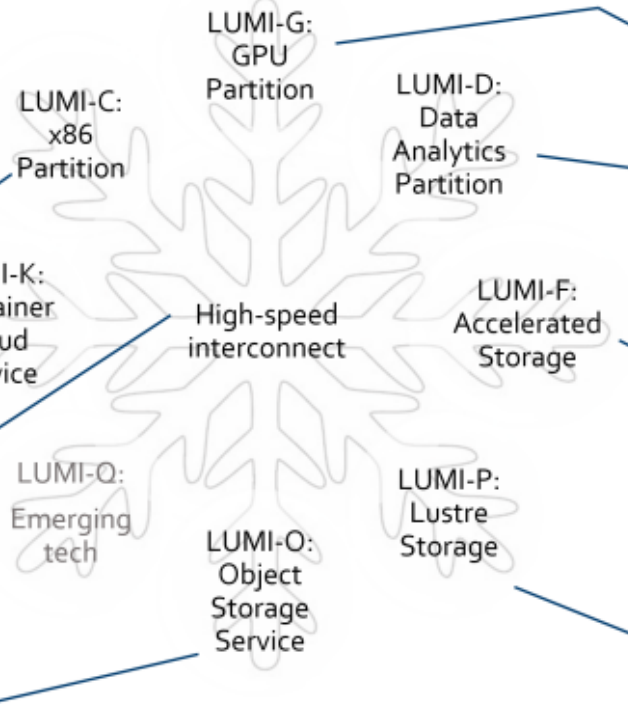
LUMI, the Queen of the North

LUMI is a Tier-0 ***GPU-accelerated supercomputer*** *that enables the convergence of* ***high-performance computing****,* ***artificial intelligence****, and* ***high-performance data analytics.***

- Supplementary CPU partition
- ~200,000 AMD EPYC CPU cores

Possibility for combining different resources within a single run. HPE Slingshot technology.

30 PB encrypted object storage (Ceph) for storing, sharing and staging data



Tier-0 GPU partition: over 550 Pflop/s powered by AMD Instinct GPUs

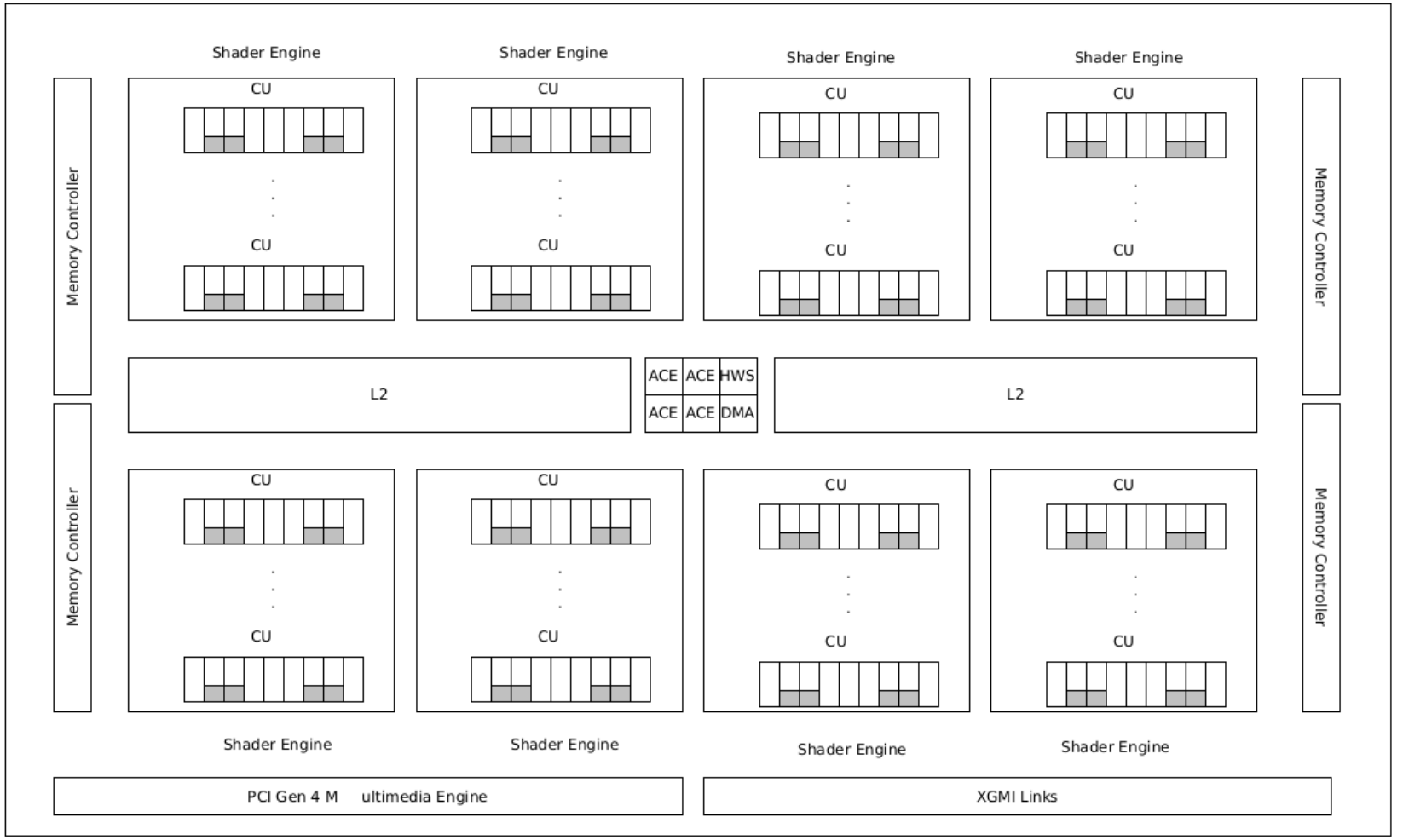
Interactive partition with 32 TB of memory and graphics GPUs for data analytics and visualization

7 PB Flash-based storage layer with extreme I/O bandwidth of 2 TB/s and IOPS capability. Cray ClusterStor E1000.

80 PB parallel file system

www.lumi-supercomputer.eu #lumisupercomputer #lumieurohpc

AMD GPUs (MI100 example)



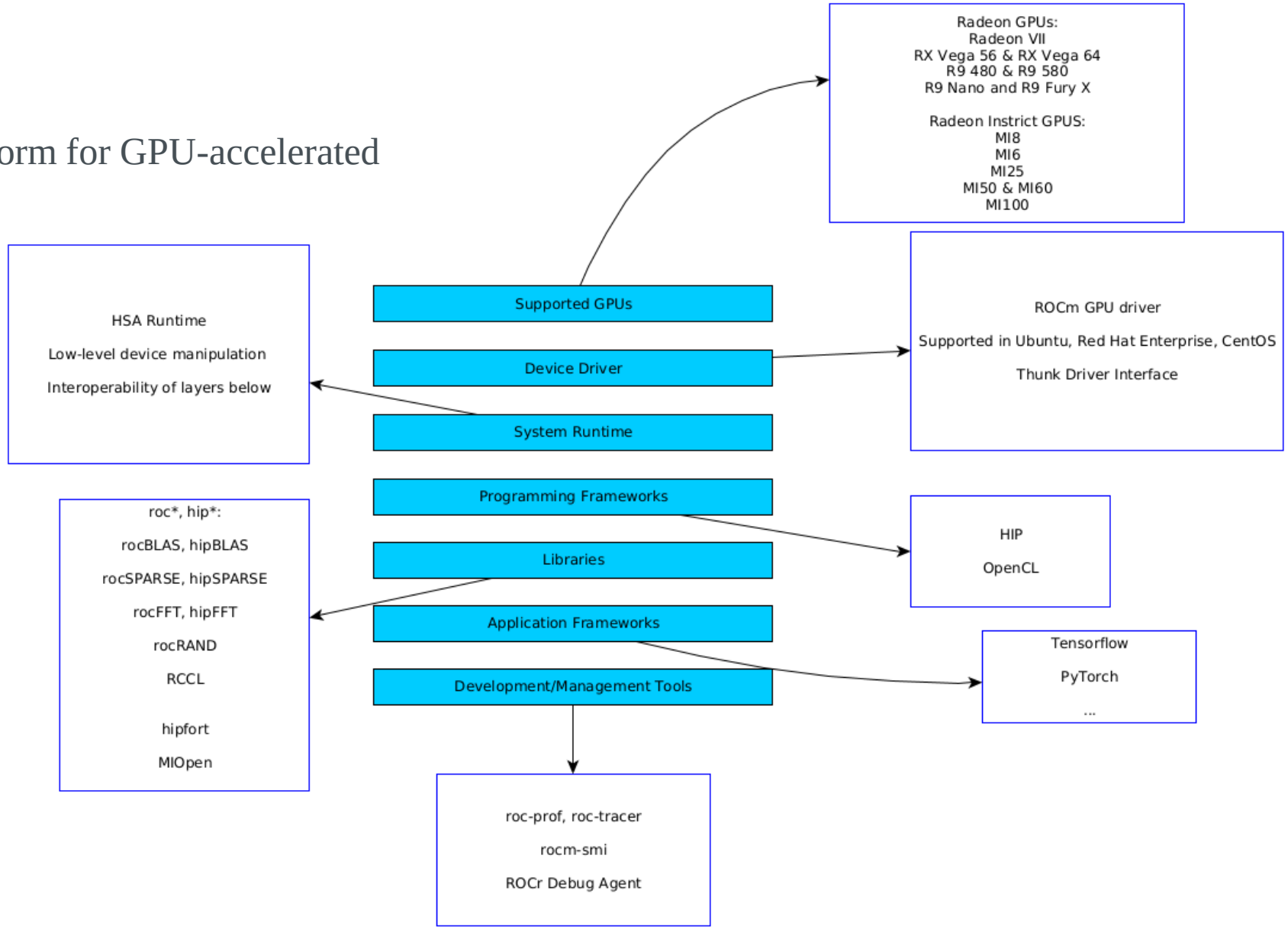
AMD MI100

Systems

- Remote node with AMD MI100 and local SATA home storage in RAID
- CSC Puhti supercomputer, 4 x NVIDIA V100 GPUs per node with access to Lustre parallel filesystem. Some nodes have local NVMe Intel SSD DC P4600 (2.85/1.9 GB/s read/write)

ROCm

- Open Software Platform for GPU-accelerated Computing by AMD



Introduction to HIP

- HIP: Heterogeneous Interface for Portability is developed by AMD to program on AMD GPUs
- It is a C++ runtime API and it supports both AMD and NVIDIA platforms
- HIP is similar to CUDA and there is no performance overhead on NVIDIA GPUs
- Many well-known libraries have been ported on HIP
- New projects or porting from CUDA, could be developed directly in HIP

https://github.com/ROCm-Developer-Tools/HIP

Differences between CUDA and HIP API

CUDA

```
#include ”cuda.h”

cudaMalloc(&d_x, N*sizeof(double));

cudaMemcpy(d_x,x,N*sizeof(double),
                cudaMemcpyHostToDevice);

cudaDeviceSynchronize();
```

HIP

```
#include ”hip/hip_runtime.h”

hipMalloc(&d_x, N*sizeof(double));

hipMemcpy(d_x,x,N*sizeof(double),
                hipMemcpyHostToDevice);

hipDeviceSynchronize();
```

Hipify

- Hipify tools convert automatically CUDA codes
- It is possible that not all the code is converted, the remaining needs the implementation of the developer
- Hipify-perl: text-based search and replace
- Hipify-clang: source-to-source translator that uses clang compiler
- Porting guide: https://github.com/ROCm-Developer-Tools/HIP/blob/main/docs/markdown/hip_porting_guide.md

Hipify Elbencho Benchmark

$ git clone https://github.com/breuner/elbencho.git
$ cd elbencho

$ *hipconvertinplace-perl.sh* .

warning: ./source/workers/LocalWorker.cpp:#336 : &LocalWorker::cudaMemcpyGPUToHost : &LocalWorker::noOpCudaMemcpy;

warning: ./source/workers/LocalWorker.cpp:#340 : funcPreWriteCudaMemcpy = &LocalWorker::cudaMemcpyHostToGPU;

info: TOTAL-converted 38 CUDA->HIP refs …
warn:9 LOC:11633
warning: unconverted cudaMemcpyGPUToHost : 5
warning: unconverted cudaMemcpyHostToGPU : 4
kernels (0 total) :

- Adjust Makefile

Elbencho Options (not exhaustive)

Option	Explanation
-t arg	Threads
-b arg	Block size
-s arg	File size
-w/-r	Write/read
--iodepth arg	Asyncrhonous I/O queue per thread
--timelimit arg	Limit in seconds per benchmark
--gpuids="0,1"	Use GPUs with ID 0 and 1
--zones="0,1"	List of NUMA zones to bind a process
--allelapsed	Show the elapsed time to complete for each I/O worker
--rand	Random write/read offsets

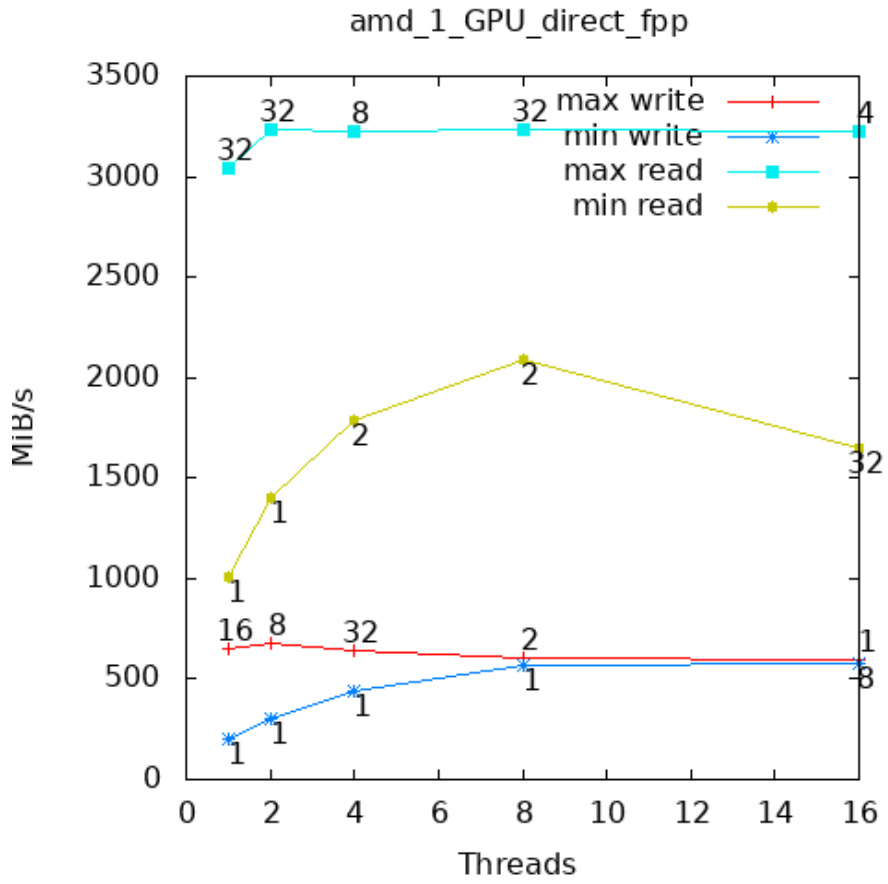
Elbencho Example

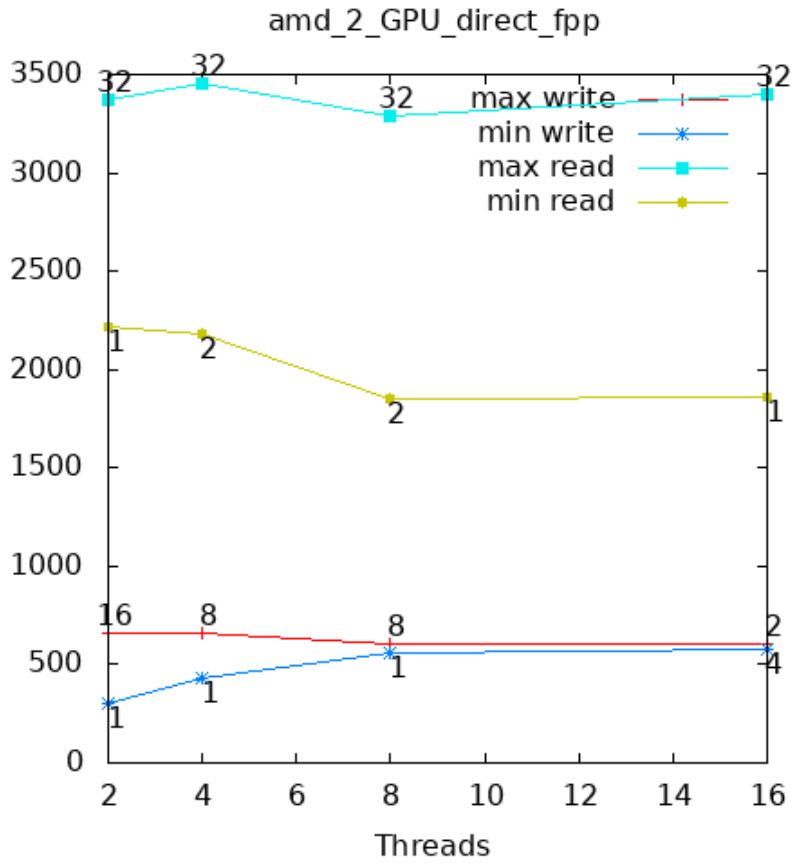
```
./elbencho -w -t1 --iodepth 1  -b 1M -s 10G --gpuids="0" --log 2 --direct --allelapsed --
timelimit 10 /global/scratch/file

OPERATION RESULT TYPE        FIRST DONE  LAST DONE
========= ================  ==========  =========
WRITE    Elapsed ms          :      10007      10007
         IOPS                :        207        207
         Throughput MiB/s :        207        207
         Total MiB           :       2077       2077
         IOs total           :       2077       2077
         Time ms each        : [ 10007 ]
```

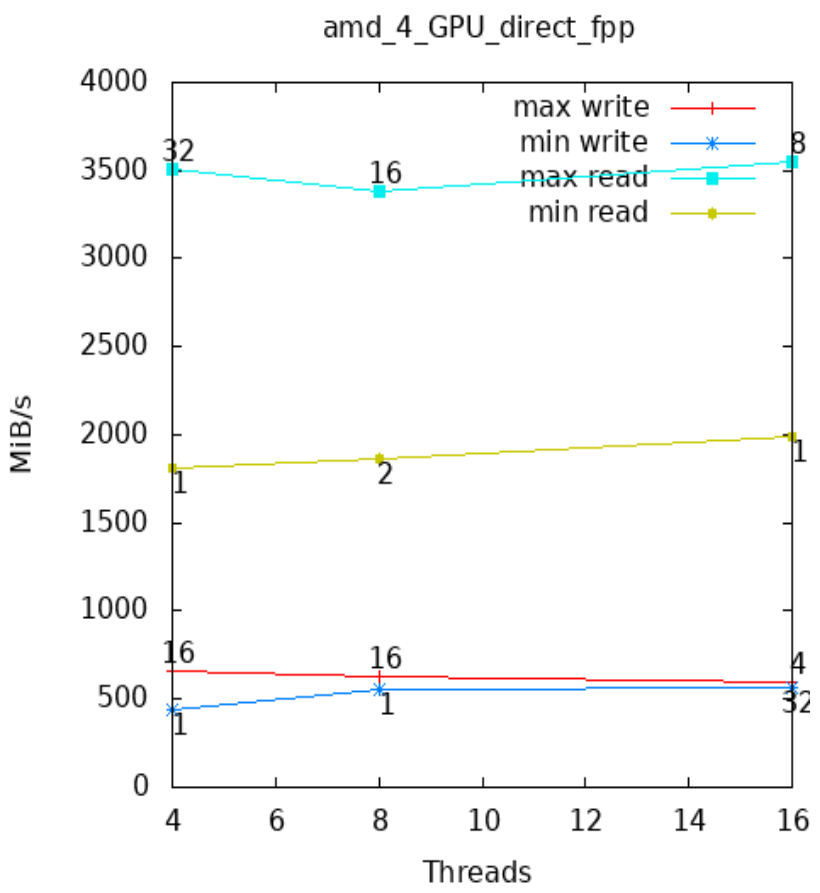
Studying cases

- Single shared file (SSF) (write/read 1M, random write/read 4k)
- One file per process (FPP) (write/read)
- Comparing I/O latency between CPU and GPU (no GDS support)
- If it is not mentioned otherwise, block size is 1MB
- Parameters studied:
  - Number of GPUS
  - Number of threads used across all the used GPUs
  - IO depth
  - Direct IO, PIN memory, PIN processes
- We write/read 3 GB per GPU device, ~10% of the memory size

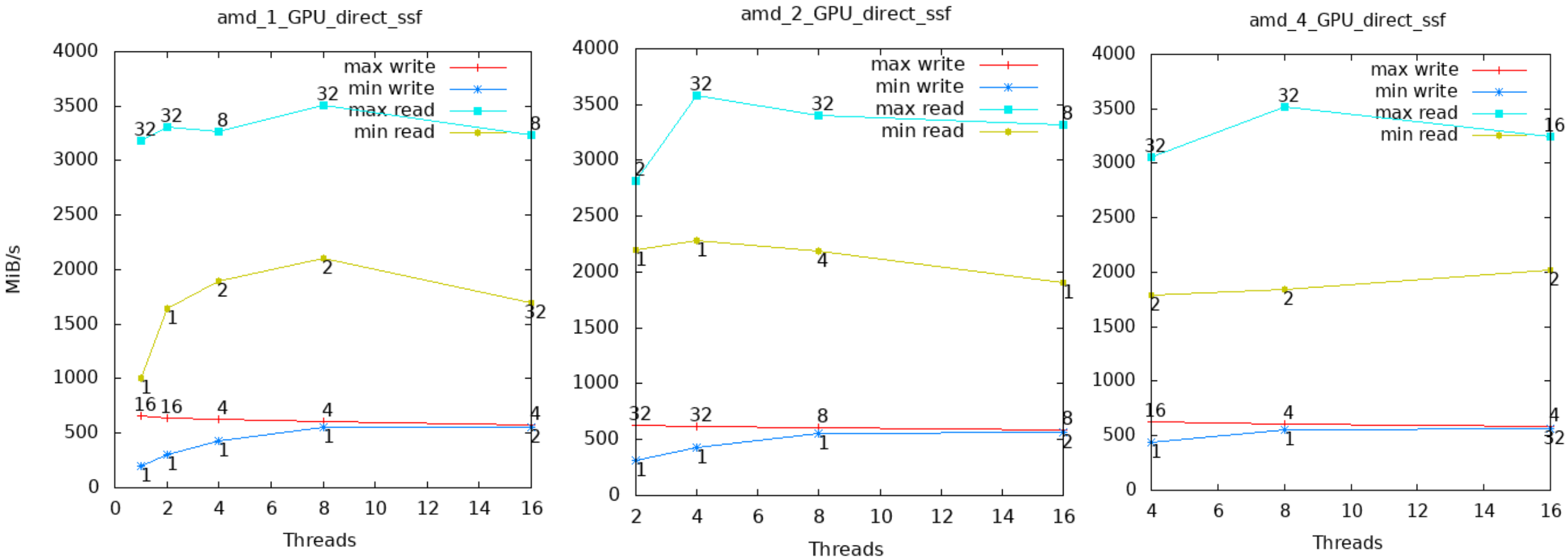
AMD GPU – Direct IO – FPP



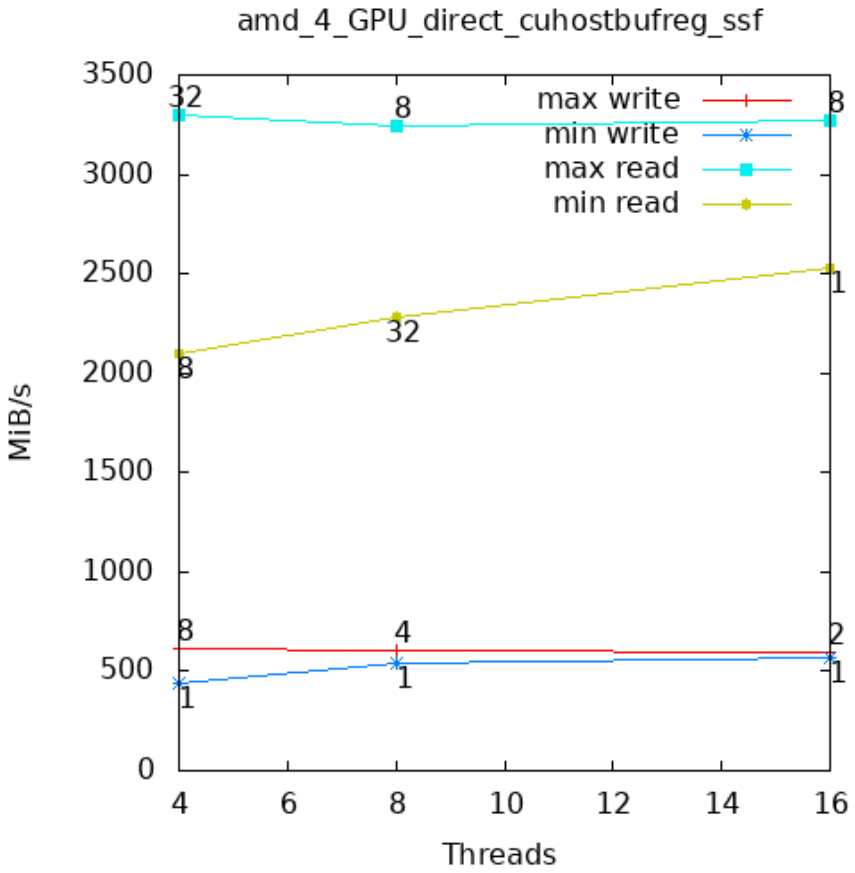


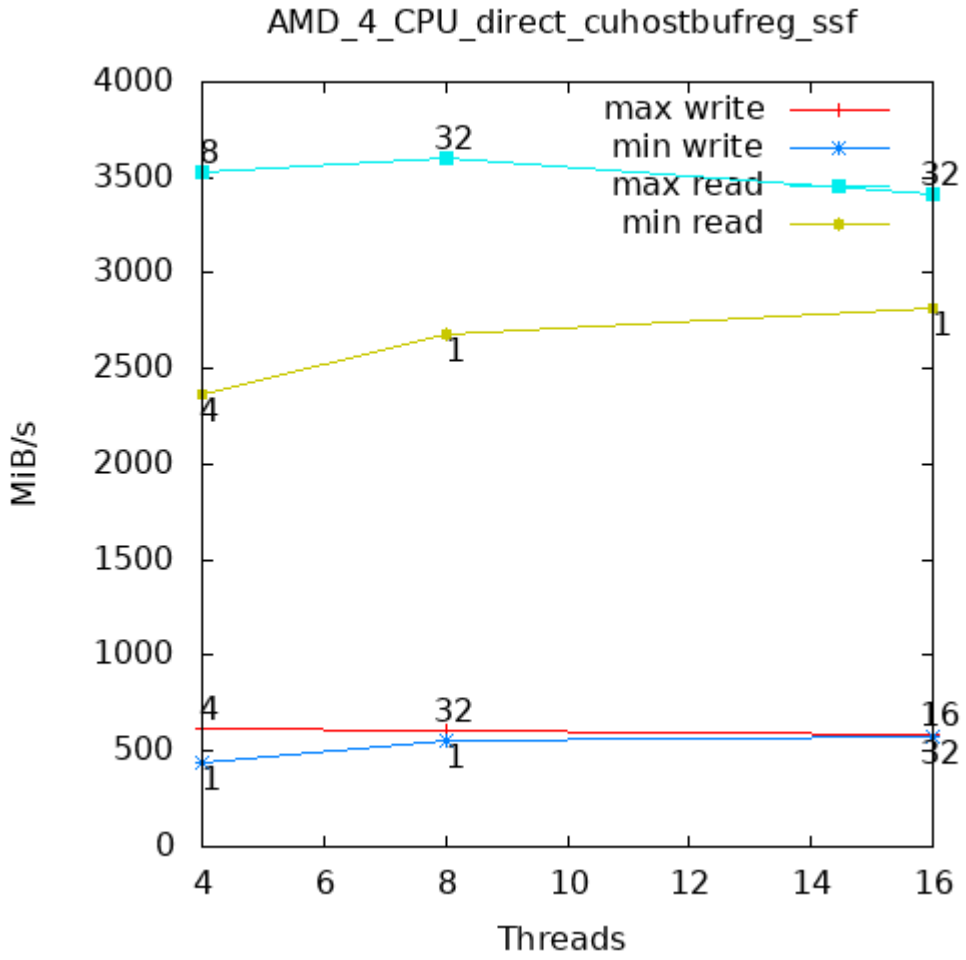


AMD GPU – Direct IO – SSF



AMD GPU – Direct IO - Register Memory– GPU vs CPU SSF



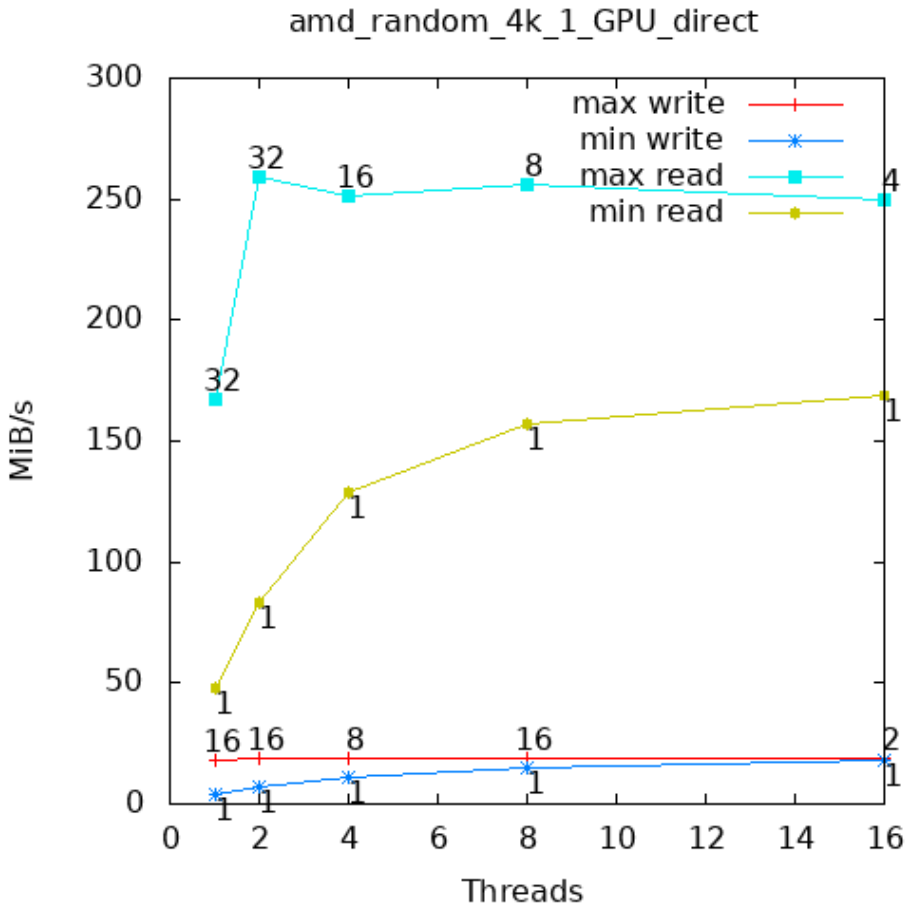


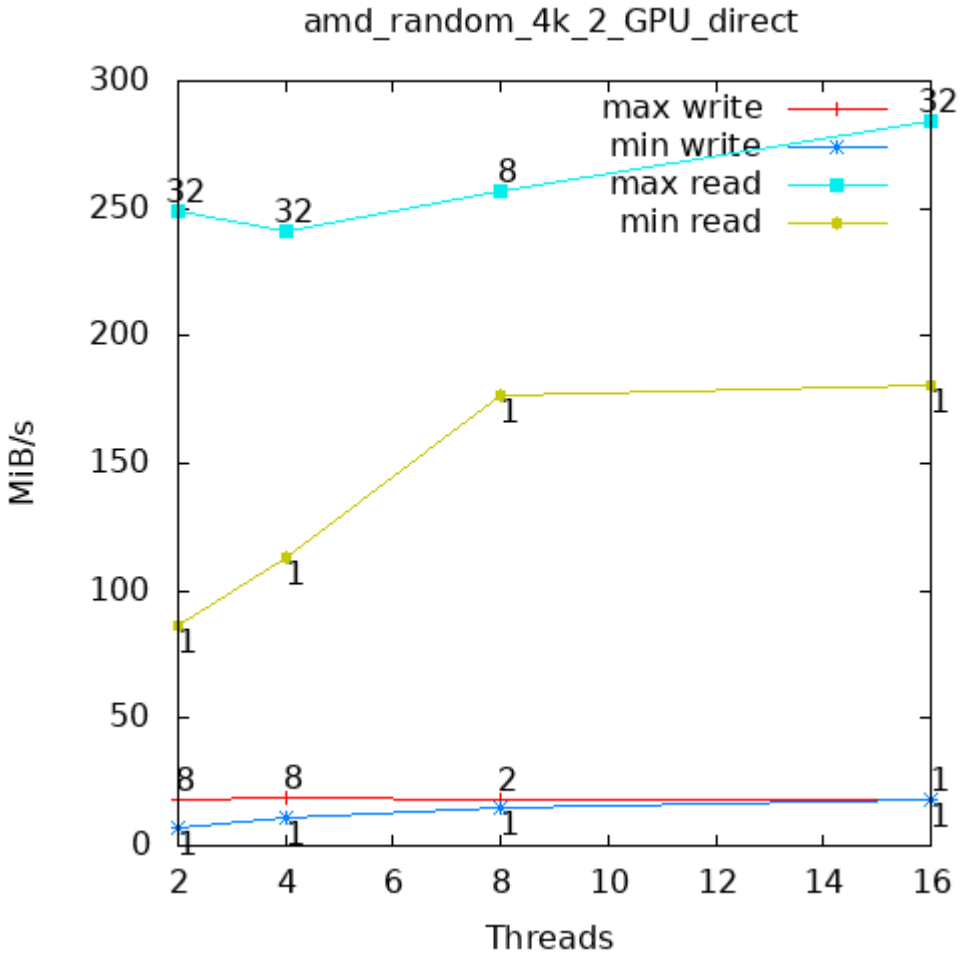
Heat map, max IO latency (in us): AMD GPU vs CPU

Maximum IO latency per IO depth (max_GPU_lat-max_CPU_lat), then selecting the min and max across all the IO depth and threads

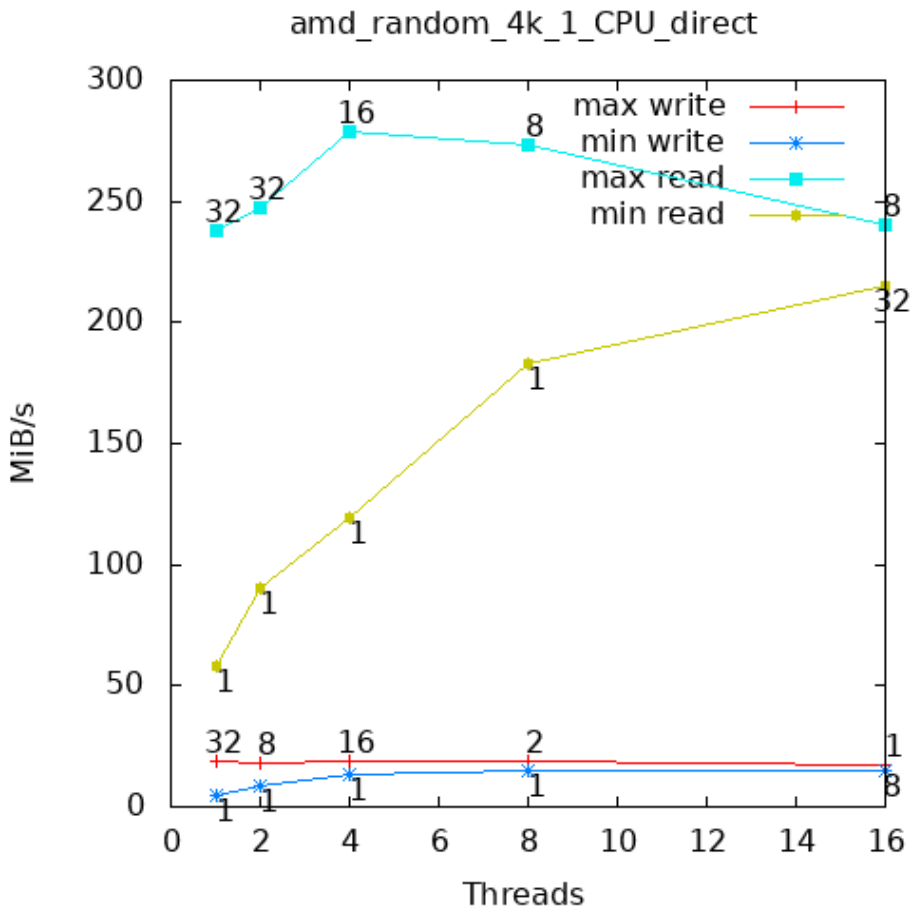
	min(max_latency)					max(max_latency)			
	WRITE		READ			WRITE		READ	
GPU	SSF	FPP	SSF	FPP		SSF	FPP	SSF	FPP
1	3671	3288	823	462	Direct IO	249840	385961	707027	741920
1	3561	4367	474	575	Direct IO- PIN Memory	102822	116971	20259	34359
2	5065	8359	2505	4274	Direct IO	65601	123294	518360	214045
2	4621	4248	243	316	Direct IO- PIN Memory	72673	93066	27235	71675
4	10359	8021	7230	5539	Direct IO	387078	312569	208727	71423
4	7862	5432	5231	789	Direct IO- PIN Memory	301216	414656	208374	42268

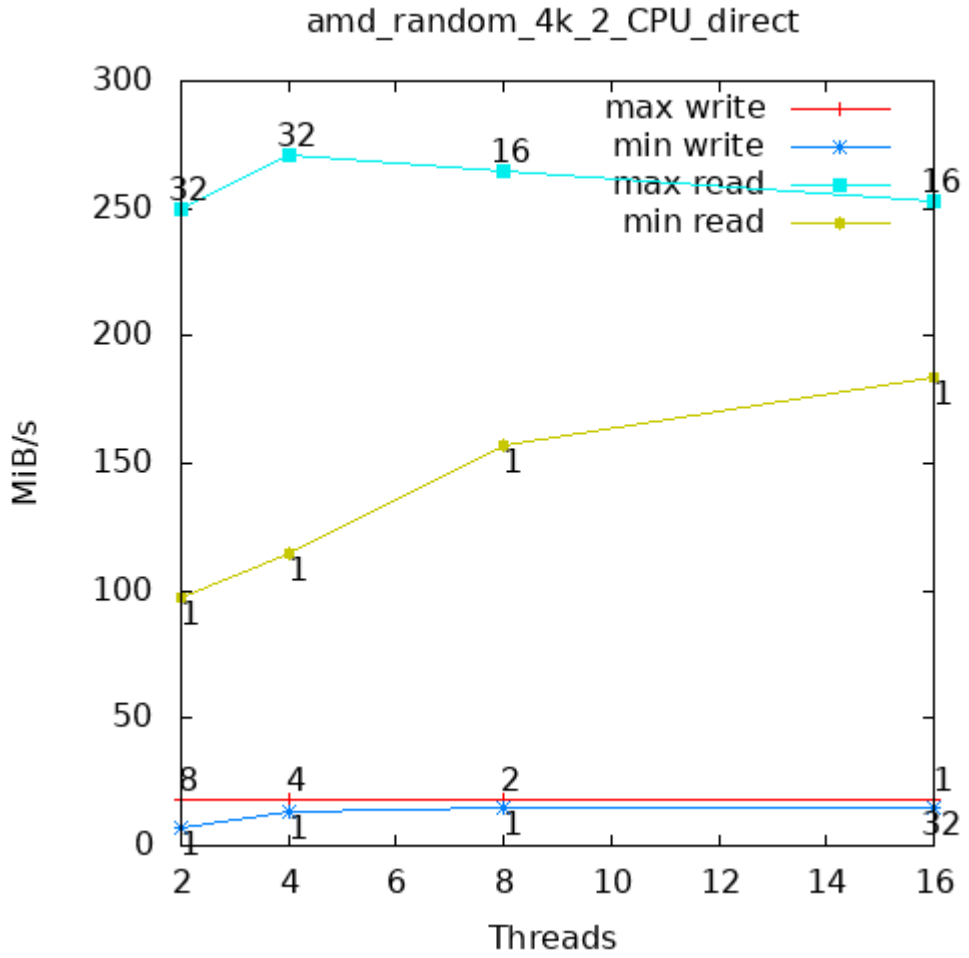
AMD GPU- Random 4k





Random 4k from the AMD CPU

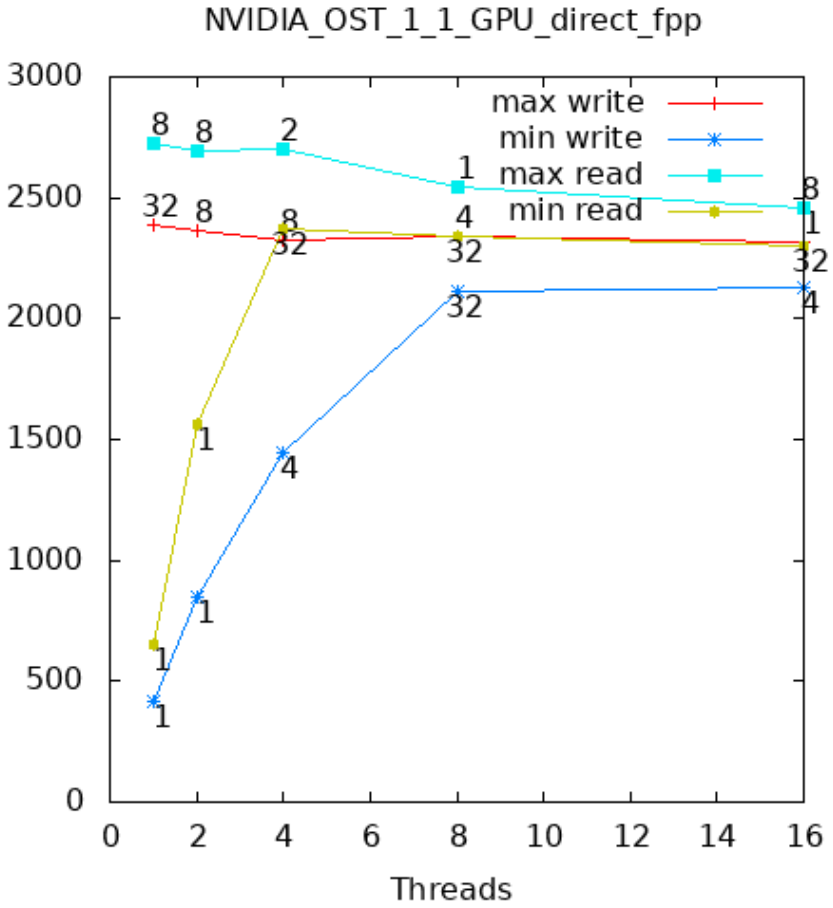


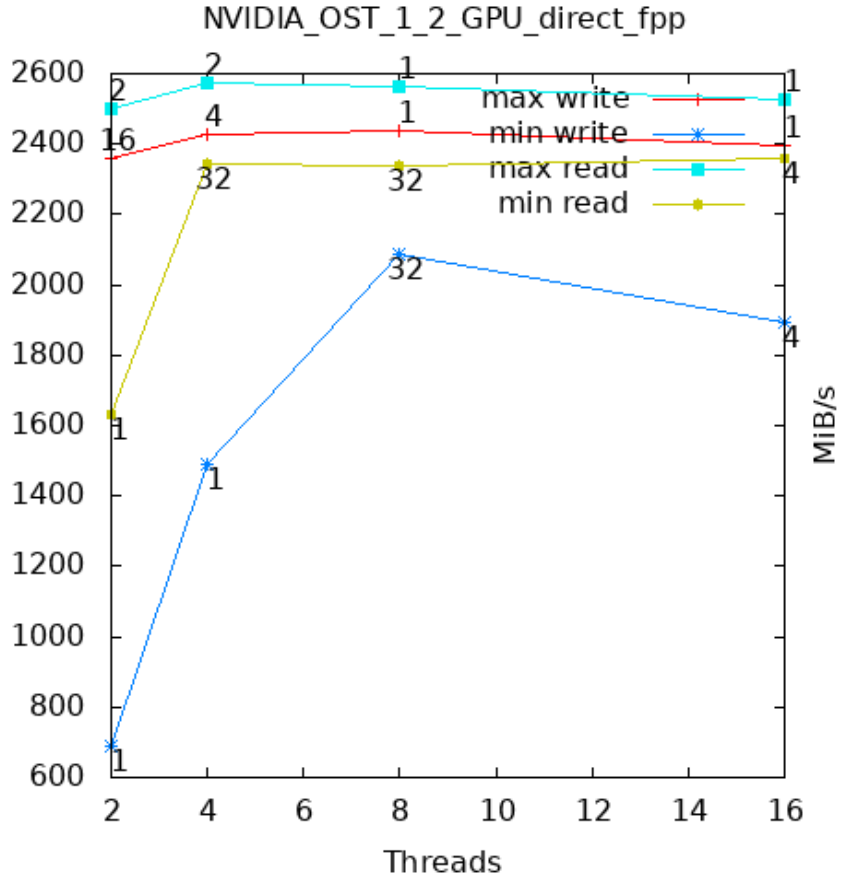


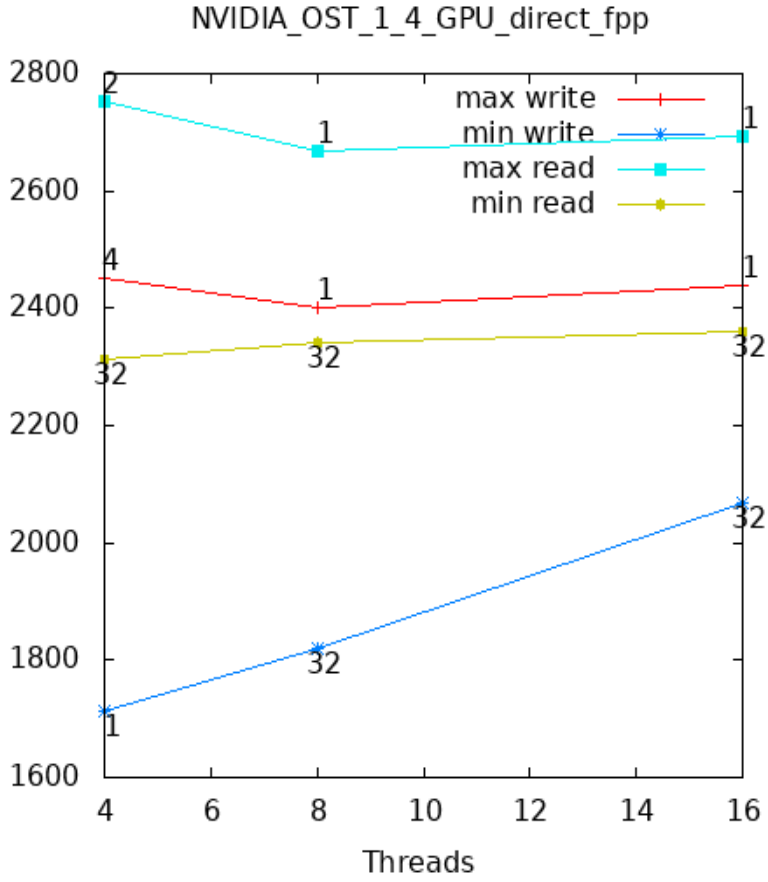
Latency – Random 4k – GPU vs CPU

	min(max_latency)			max(max_latency)	
	WRITE	READ		WRITE	
GPU	SSF			SSF	
1	2593	8151	Direct IO	16954	19316
1	3025	8222	Direct IO- PIN Memory	63120	31340
2	1764	8252	Direct IO	18576	12498
2	3499	8156	Direct IO- PIN Memory	15274	15203
4	5561	8335	Direct IO	64620	13966
4	5220	8223	Direct IO- PIN Memory	14582	26204

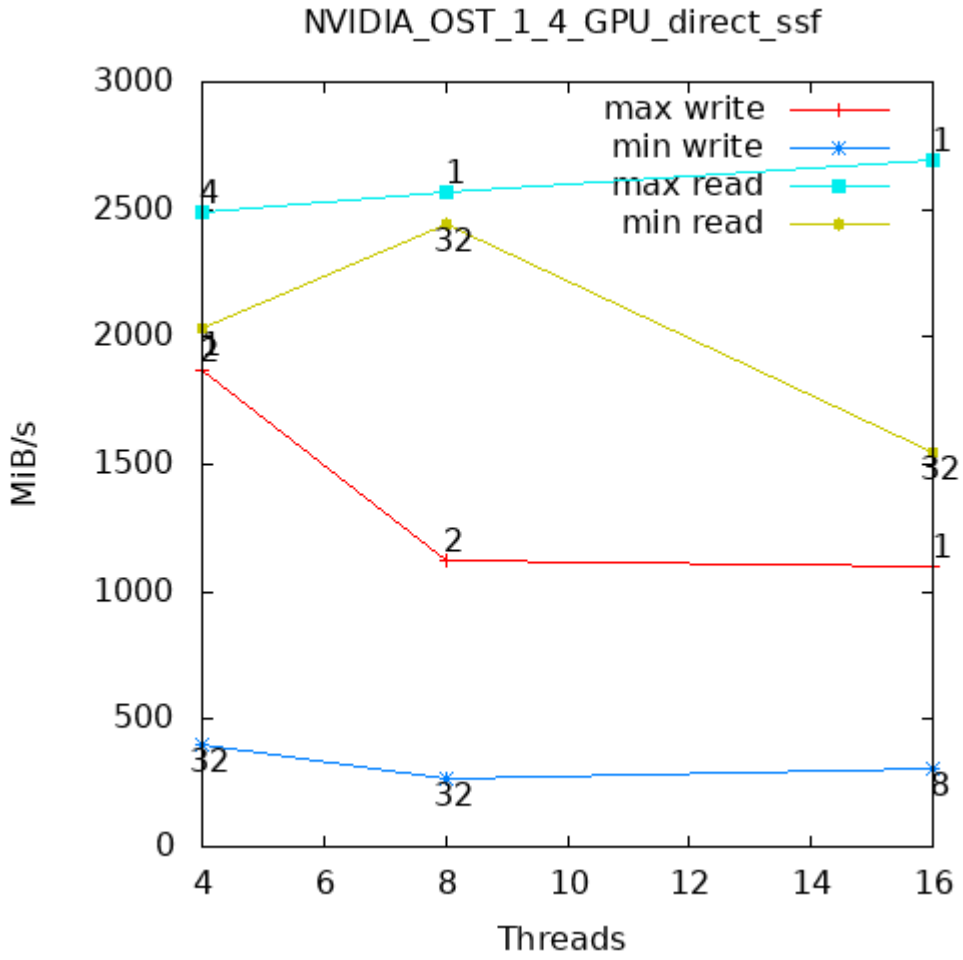
NVIDIA – V100 – Direct IO – FPP – Increasing GPUs (Lustre)

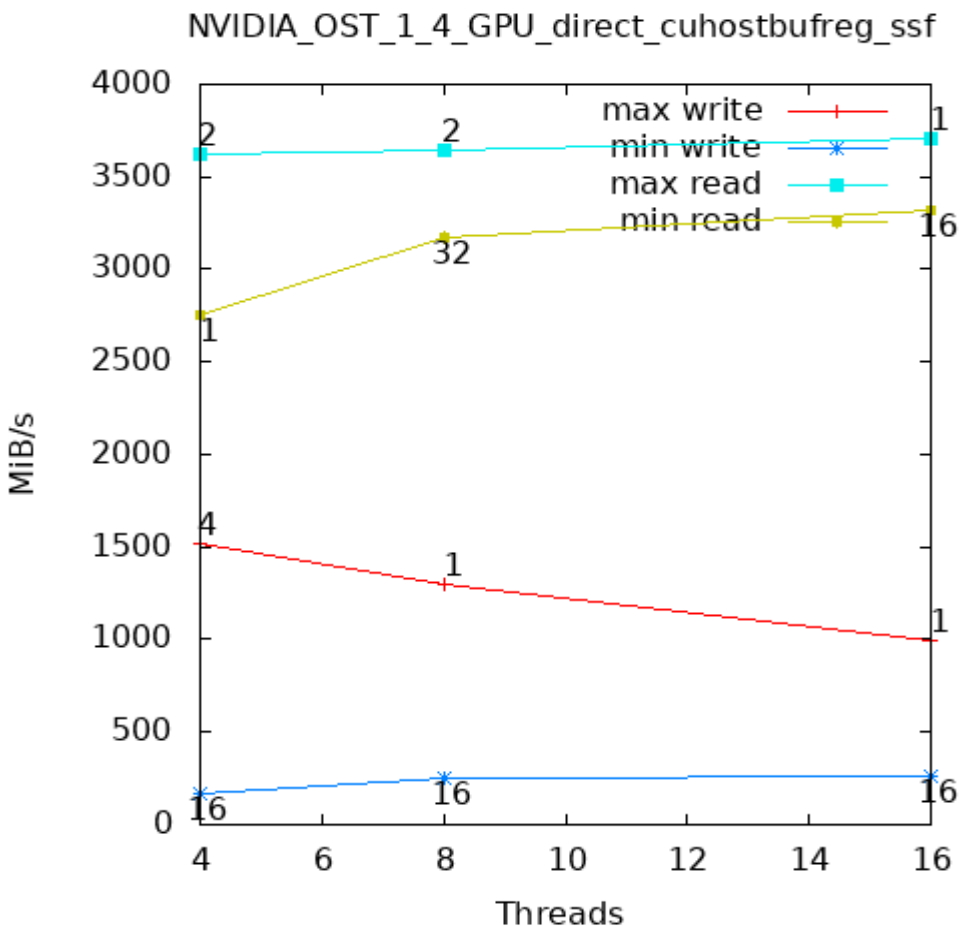




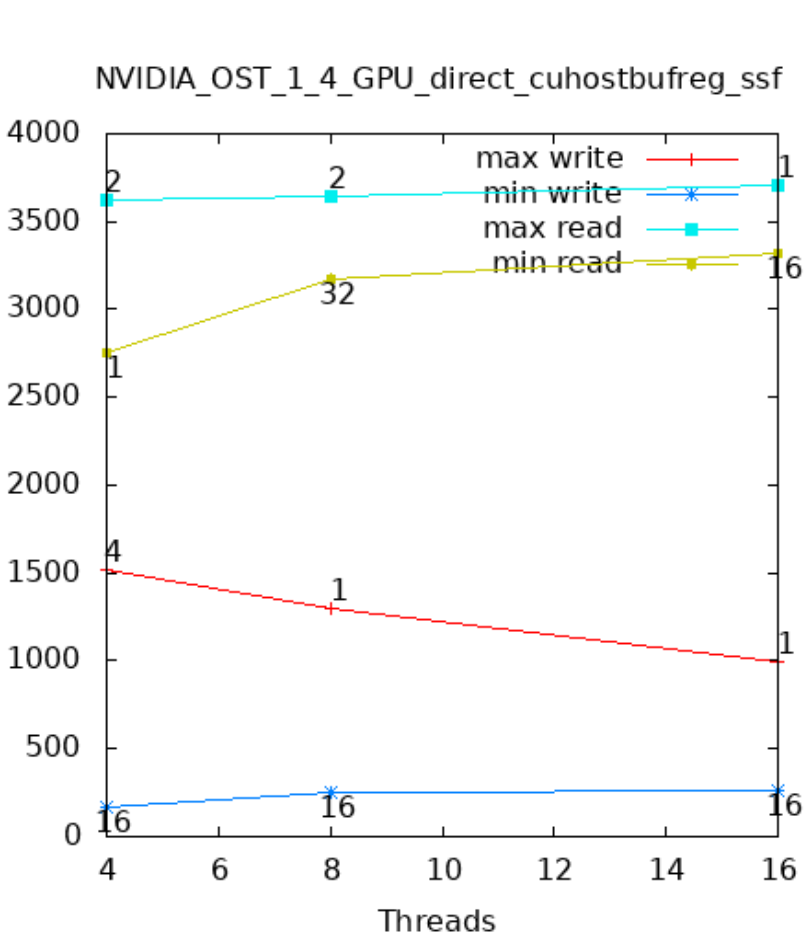


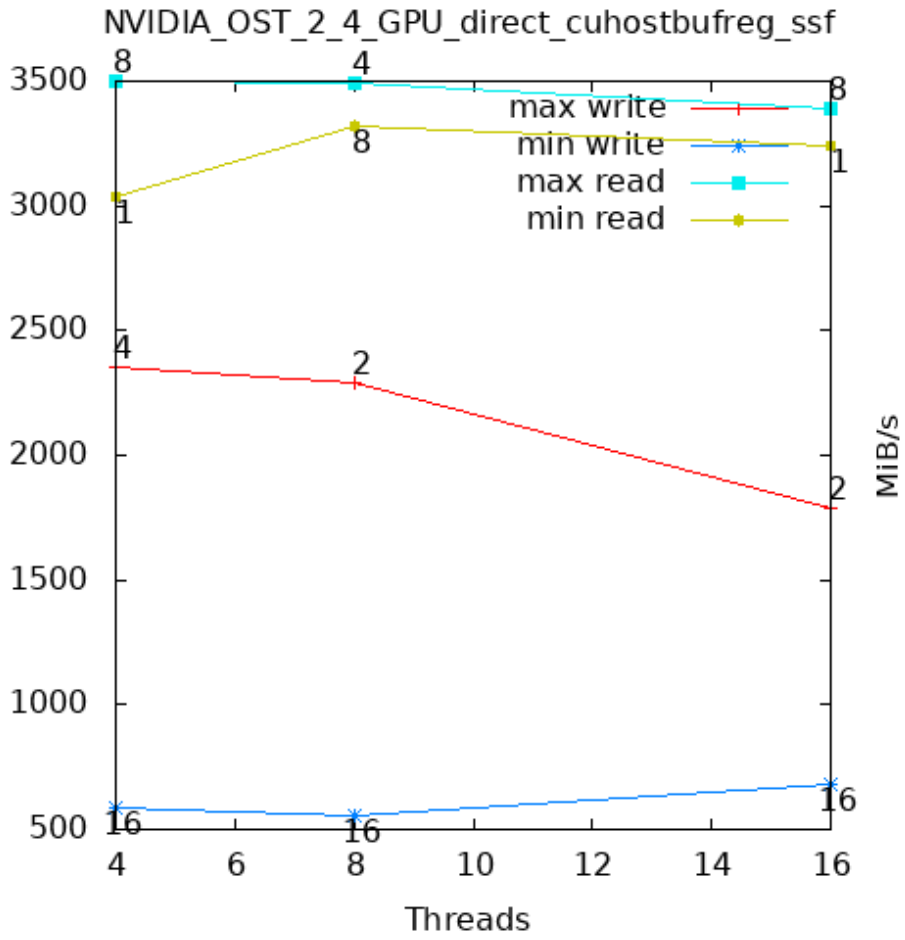
NVIDIA – 4 GPUs – Direct IO – PIN memory

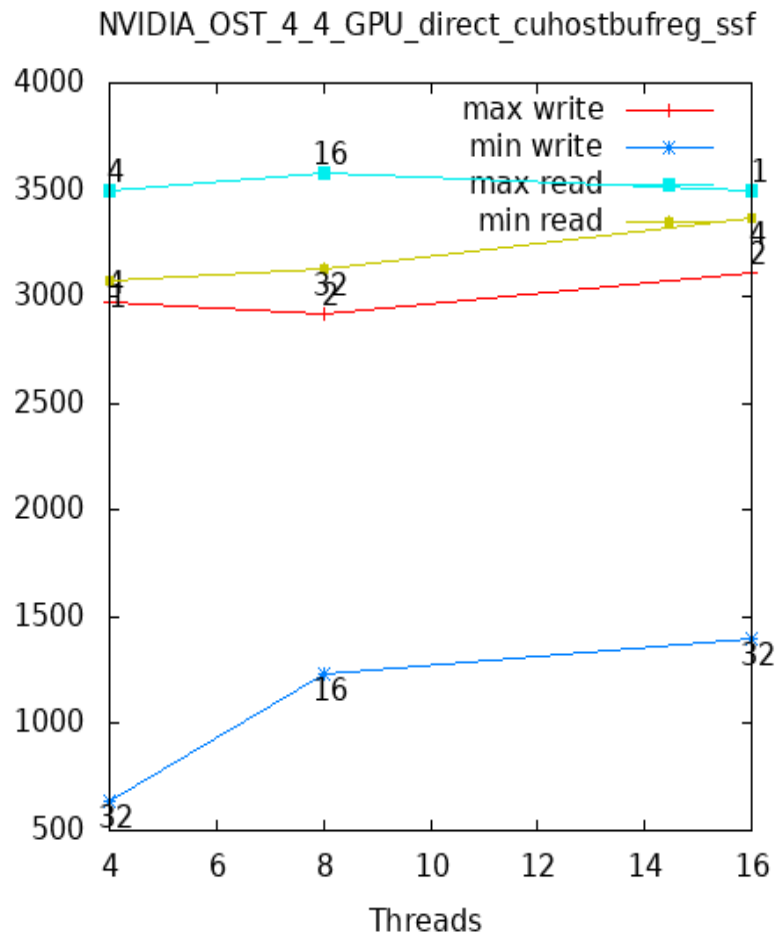




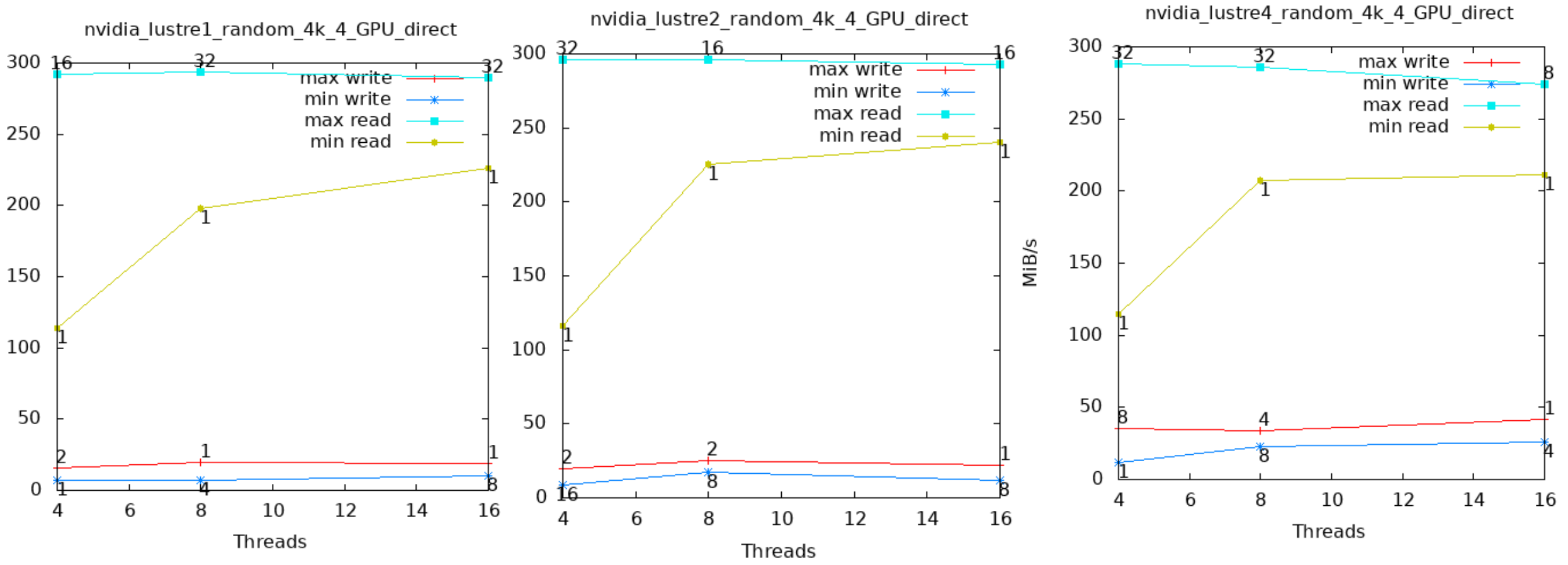
NVIDIA – Direct IO - PIN Memory – Increase OSTs



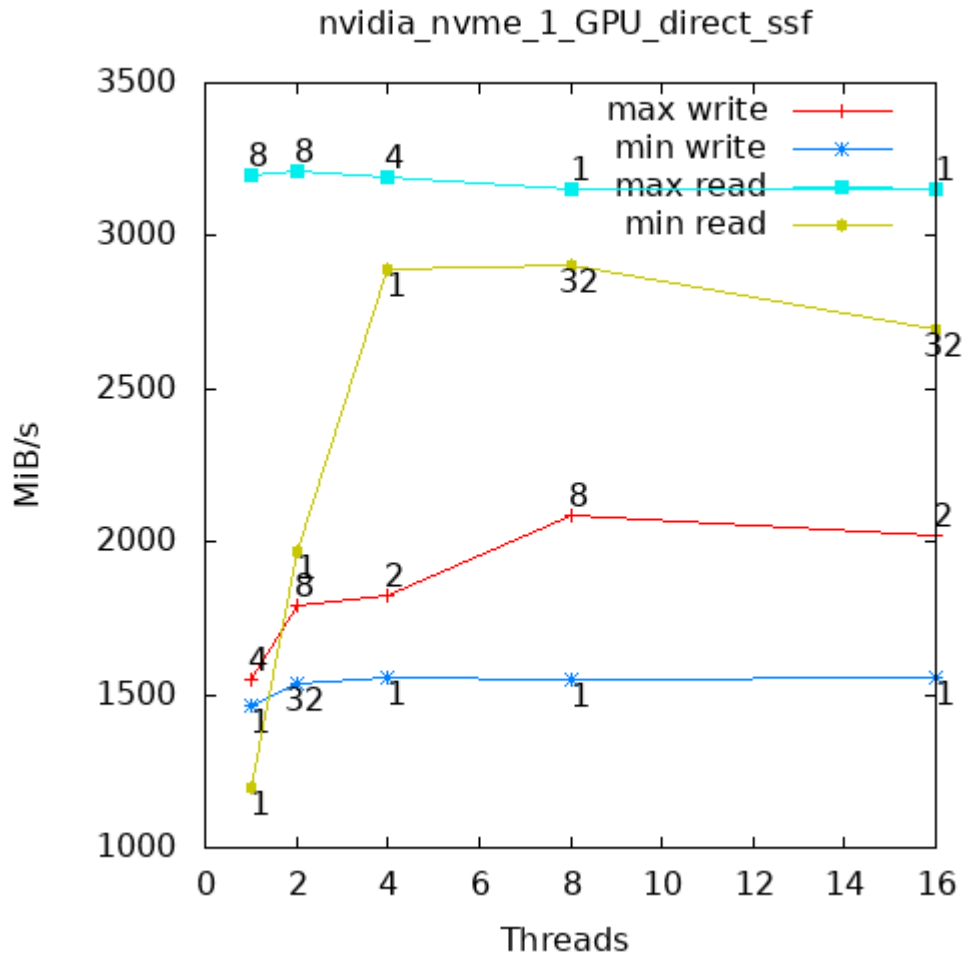


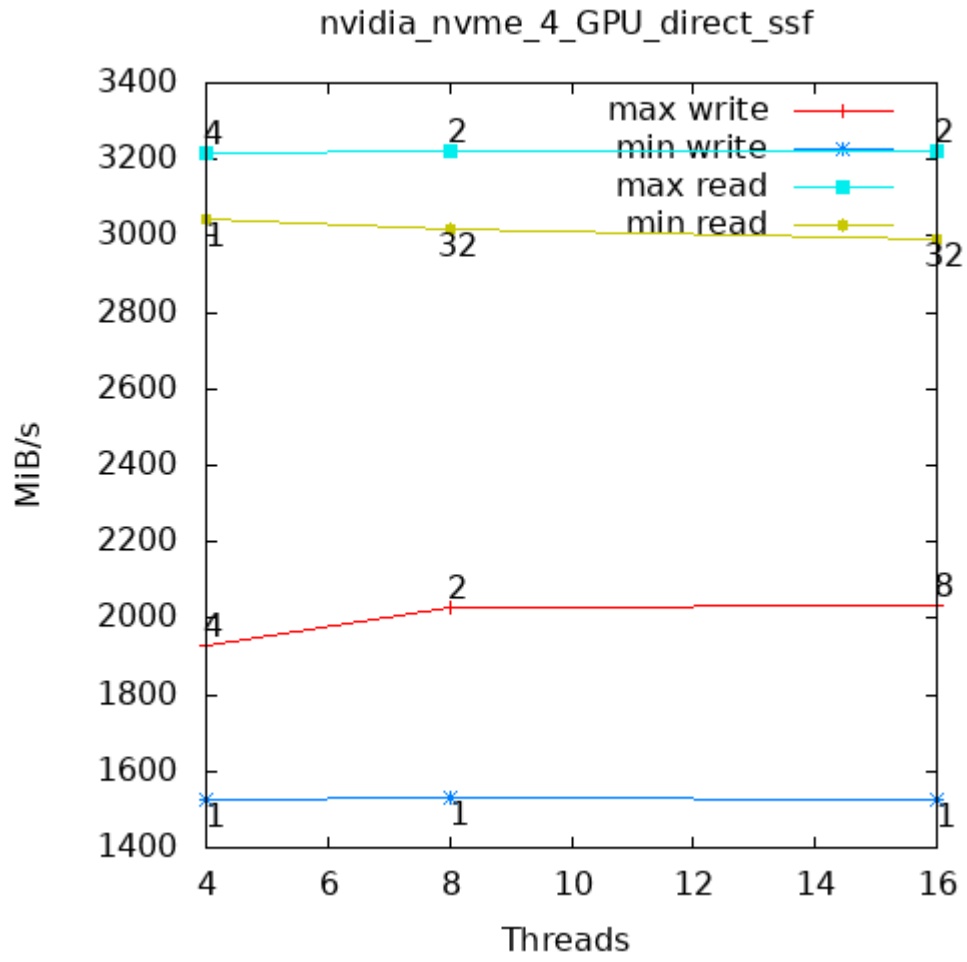


NVIDIA – Random 4k – Increasing OSTs

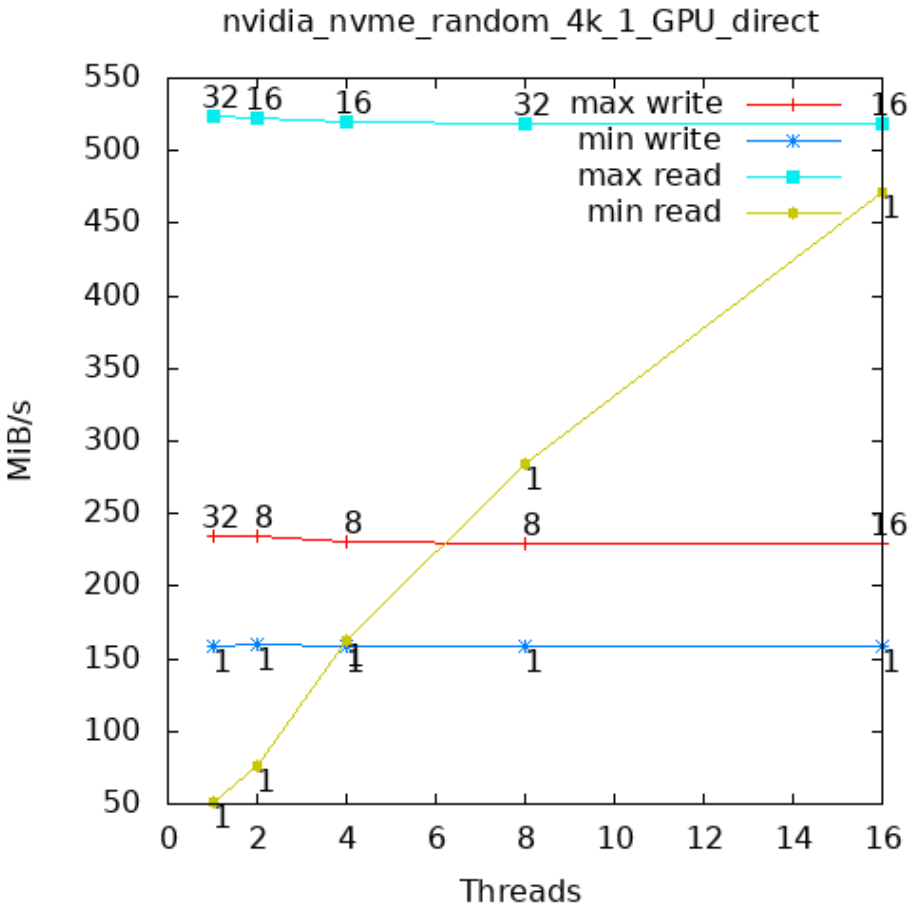


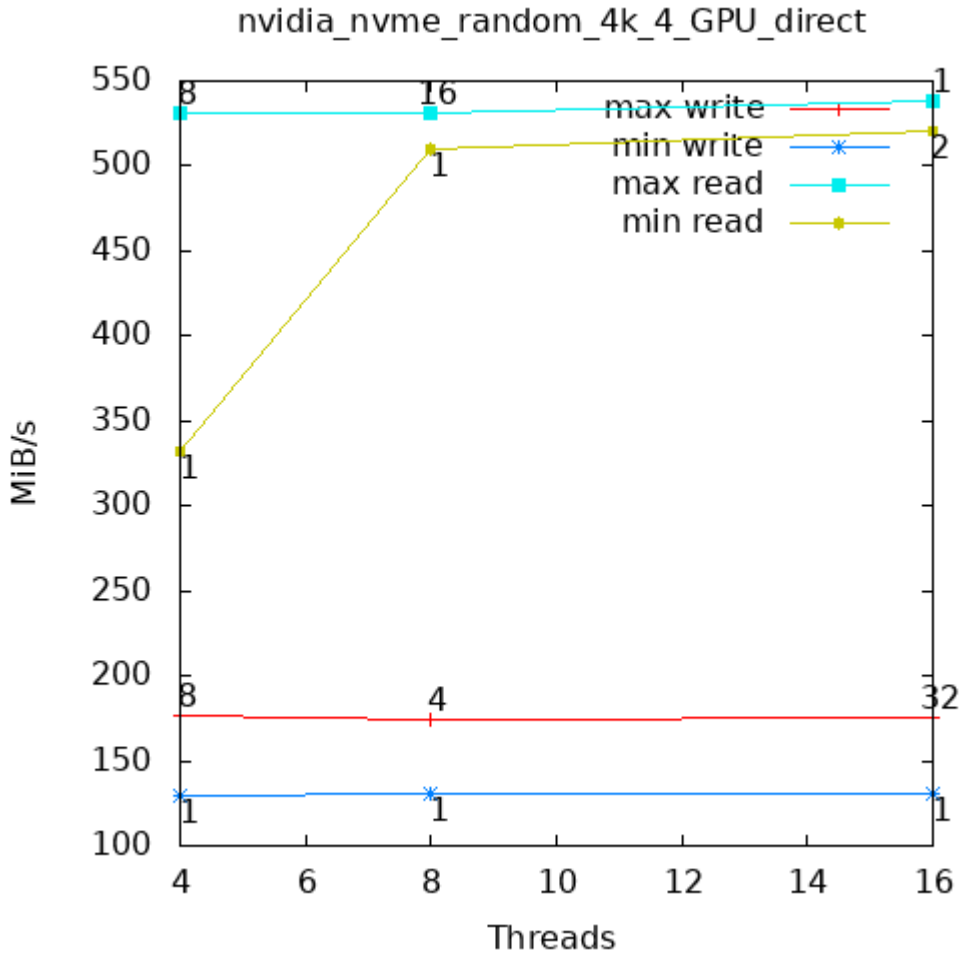
NVIDIA – NVME – Direct IO – Increasing GPUs



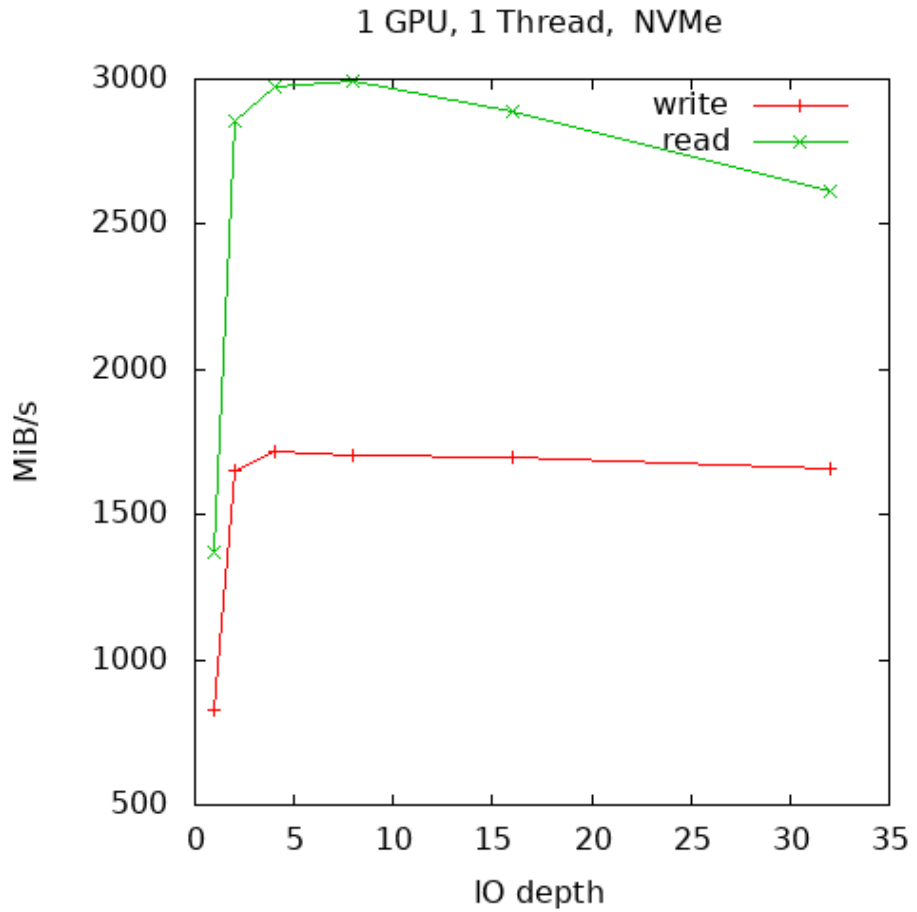


NVIDIA – NVME – 4K





No restrictions, cache allowed



Rocprof

"Name", "Calls", "TotalDurationNs", "AverageNs", "Percentage"

hipMemcpy, 108, 1259402245, 11661131, 53.96194443272036

hipHostUnregister, 4, 1054625049, 263656262, 45.18780121079821

hipHostRegister, 4, 16863223, 4215805, 0.7225430207825078

hipMalloc, 4, 2091897, 522974, 0.08963206959582197

hipFree, 4, 883041, 220760, 0.037835893625720686

hipSetDevice, 2, 5680, 2840, 0.00024337247737544858

Profiling on Lustre

==128352== Profiling application: ./elbencho -w -t 1 --iodepth 1 -b 4k -s 2G --gpuids=0 --log 2 --allelapsed --direct **/scratch/project_2002078/markoman/elbencho/filecheck**
==128352== Profiling result:

	Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:		100.00%	896.10ms	524288	1.7090us	1.6630us	2.9120us	[CUDA memcpy DtoH]
		0.00%	1.9520us	1	1.9520us	1.9520us	1.9520us	[CUDA memcpy HtoD]
API calls:		**98.49%**	**13.0881s**	524289	24.963us	12.744us	52.863ms	cudaMemcpy
		1.50%	199.17ms	1	199.17ms	199.17ms	199.17ms	cudaMalloc
		0.00%	607.96us	1	607.96us	607.96us	607.96us	cuDeviceTotalMem

Profiling on NVMe

==128151== Profiling application: ./elbencho -w -t 1 --iodepth 1 -b 4k -s 2G --gpuids=0 --log 2 --allelapsed --direct **/run/nvme/job_6636988/data/filecheck**
==128151== Profiling result:

	Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:		100.00%	895.86ms	524288	1.7080us	1.6630us	3.1680us	[CUDA memcpy DtoH]
		0.00%	2.2720us	1	2.2720us	2.2720us	2.2720us	[CUDA memcpy HtoD]
API calls:		**97.24%**	**7.01101s**	524289	13.372us	12.433us	50.903ms	cudaMemcpy
		2.74%	197.47ms	1	197.47ms	197.47ms	197.47ms	cudaMalloc

Conclusions/Future work

- We need more than one IO process on the GPU for more efficient IO, how good is that for real applications though? Need to find a balance
- IO depth plays an important role
- Experiment on AMD GPU node with NVMe
- Decide the amount of memory to flush to storage
- Check other block sizes
- Prepare cases for extensive analysis on LUMI
- Discuss with AMD for other solutions and possible bug. Seems NVIDIA GPU has less overhead regarding the cudaHostUnregister
- What about hipMemcpyAsync?