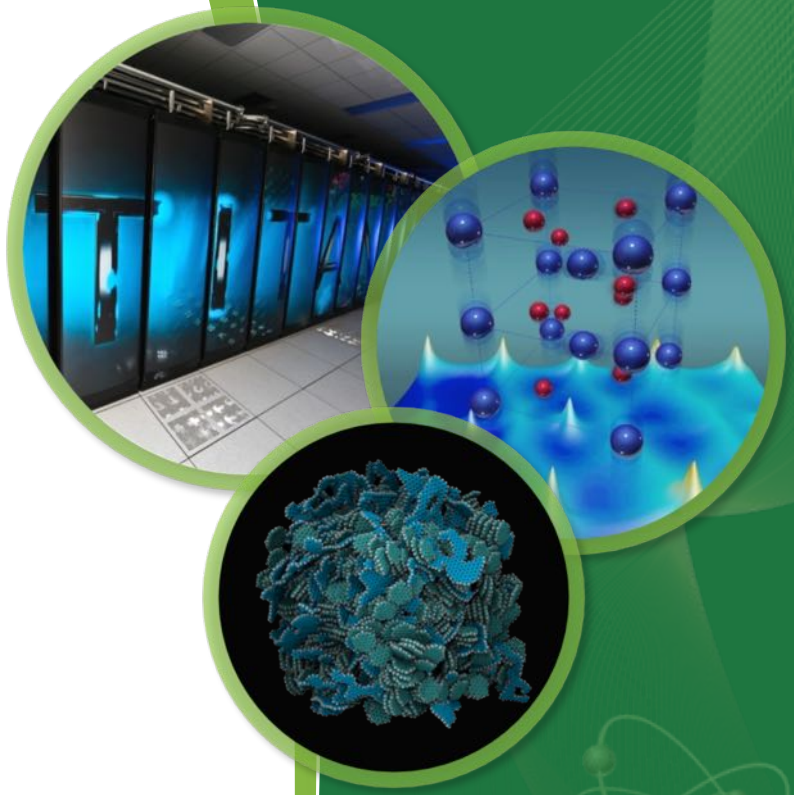
Summit Node Bandwidths: Performance Results



Wayne Joubert

Scientific Computing Group
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory

Summit Workshop
Feb 11-13, 2019

These slides contain work excerpted from Vazhkudai et al., “The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems,” presented at SC18.

ORNL is managed by UT-Battelle
for the US Department of Energy

Motivation

- IDEAL WORLD: we would like to be able to say:
  - *"For optimizing code, flops are all that matter"*
  - *"threading is all that matters"*
  - *"GPU performance is all that matters"*
- REAL WORLD:
  - Node bandwidths between components (CPUs, GPUs, memories, nodes) *decisively affect performance*, for many (maybe most?) applications
  - A high CPU or GPU flop rate is *almost entirely useless* if the data paths cannot feed the processors/GPUs fast enough
  - This is only getting worse with each successive system, as memory and interconnect bandwidths are improving more slowly than flop rates

Motivation (2)

- This was predicted as far back as 2008—DARPA Exascale report predicted the “memory wall” (and related “power wall,” power cost of moving data) would be fundamental challenges to reaching exascale -- these predictions have largely come true
- Since we started working with GPUs in 2009, at the OLCF we have tried to restructure our codes to both increase thread parallelism and reduce memory traffic, to get ahead of the problem
- This was motivated by a picture of an extremely powerful processor connected to the rest of the system (memory, interconnect, etc.) by an *extremely thin straw*, requiring codes to heavily reuse data in registers and caches to reach high performance (cf. paper, *Accelerated application development: The ORNL Titan experience*)
- Many examples of optimizing data motion in the broad community, e.g.,
  - ECP CEED codesign center – work on high order tensor product finite elements, to convert sparse linear algebra to high computational intensity operations
  - Communication-avoidant Krylov solver methods
  - MSM multilevel methods replacing FFTs for molecular dynamics long range force computations
- Additionally the nodes of our systems are becoming more complex, with more bandwidth issues that can affect code performance. Understanding these is critical to optimizing our codes

Overview

- Will present experimental data on speeds and feeds in the Summit node (and also the LLNL Sierra node)
  - Theoretical peak performance as baseline
  - Actual achievable performance for (somewhat idealized) kernel benchmarks
  - (note performance in complex applications may be yet different)
- Most of this material is excerpted from the paper, Vazhkudai et al., “The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems,” presented at SC18. Please see this paper for more details.
- These experiments were run in March 2018. Since then, software / firmware updates have improved node performance and may improve results for some of these tests

Summit, Sierra Node Architecture

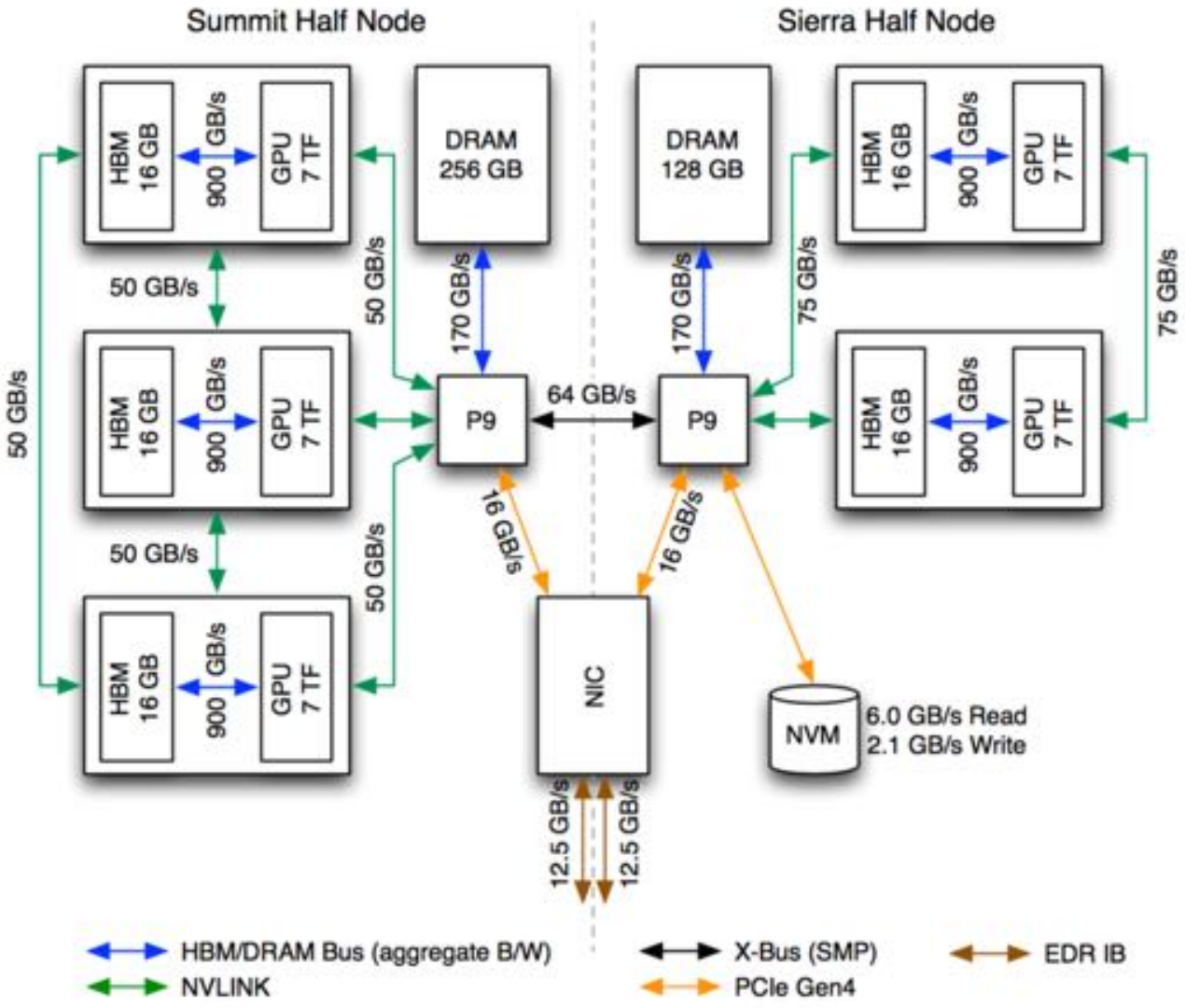


Fig. 1: Block Diagram of Summit and Sierra Half Nodes

CPU Stream Benchmark

- Measures speed of CPU memory for four operations: for double precision vectors `x`, `y`, `z` and scalar `a`,
  - Copy: `y = x`
  - Scale: `y = a * x`
  - Add: `y = z + x`
  - Triad: `y = z + a * x`
- Performance measured in GB/sec, where both load and store data transfer rates are counted together
- Run using GCC compiler with OpenMP threads, one thread per core (44 cores/node)
- Core isolation is enabled on Summit / Sierra: several cores of each CPU socket set aside to offload OS tasks
- Tests run on Summit (or TDS Peak) or Sierra (or TDS Butte)
- All tests are run on a single arbitrary node; no attempt to identify possible small performance variations between individual nodes of a system

CPU Stream Benchmark

TABLE III: CPU stream rates on Peak and Sierra (GB/s)

system cores	Peak/ci 40	Peak/ci 42	Peak 40	Peak 44	Sierra 40	Sierra 44
Copy	272.9	273.5	273.1	274.6	277.3	278.3
Scale	269.6	270.6	269.5	271.4	274.4	275.7
Add	268.8	269.8	268.7	270.6	273.5	274.9
Triad	273.0	273.9	273.5	275.3	277.7	279.0

We first present results for several memory microbenchmarks. The stream code, compiled with GCC measures CPU memory bandwidth under OpenMP threading. Table III shows the best result in 1,000 trials for Peak with core isolation (ci) (its normal operating mode), Peak without core isolation, and Sierra (without core isolation). Performance is similar for both systems and slight differences between Peak and Sierra may be partly due to slightly different system memory configurations [11] or inherent performance variability. Multiple benchmark trials reveal runtime variation as high as 9%. Also, performance was up to 4% higher if the benchmark was run after the POWER9 was idle for several minutes prior to the experiment. While we are investigating this issue, one possible explanation is aggressive frequency throttling.

SUMMIT: peak 170 X 2 = 340, actual ~ 275
actual: ~ 82%

TITAN: peak 25.6 X 2 = 51.2, actual ~ 34
actual ~ 67%

JAGUARPF: peak 25.6, actual ~ 19
actual ~ 75%

```
TITAN:
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           17382.1     0.014840     0.014728     0.015088
Scale:          17663.7     0.014607     0.014493     0.014795
Add:            16853.9     0.022983     0.022784     0.023280
Triad:          16899.2     0.022930     0.022723     0.023176
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           17320.7     0.014916     0.014780     0.015109
Scale:          17622.3     0.014629     0.014527     0.014908
Add:            16790.5     0.023040     0.022870     0.023418
Triad:          16797.2     0.023020     0.022861     0.023416
-------------------------------------------------------------
```

GPU Stream Benchmark

We use a variant [12] of stream to measure GPU HBM2 bandwidth on all GPUs of 15 Peak and Butte nodes. Best values for Peak were 789 (Copy), 788 (Mul) and 831 (Add and Triad) GB/s; values for Butte differed by less than 1%, confirming the expected result that node architecture differences do not impact GPU memory performance. These figures represent 88% and 92% of the 900 GB/s peak, a much higher fraction of peak than Titan GDDR memory. Most trials in 1,000 vary in performance less than 10%, although a few outliers were up to 16X slower than the best case.

```
TITAN
Function     MBytes/sec  Min (sec)   Max         Average   %peak
Copy         181435.808  0.00296     0.00296     0.00297    72%
Mul          181202.206  0.00296     0.00296     0.00298    72%
Add          179750.930  0.00448     0.00448     0.00449    72%
Triad        179636.213  0.00448     0.00448     0.00449    72%
```

Comments:

- GPU firmware updates were installed in late 2018 to fix performance issues that were discovered, these may remedy the performance irregularity issue shown here

NVLINK Benchmark

- Measure connection speed of NVLINK connection between CPU and GPUs
- Tests are modified from NVIDIA CUDA Samples
- First test: run 1 MPI rank on 1 core of the node, to access each single GPU on the node in isolation (some on-socket, some off-socket). This is not realistic for applications but is done for illustration purposes

NVLINK Benchmark: Single GPU

TABLE IV: Single Node Single GPU NVLink Rates (GB/s)

GPU	0	1	2	3	4	5	(peak)
Peak htod	45.93	45.92	45.92	40.63	40.59	40.64	50
Peak dtoh	45.95	45.95	45.95	36.60	36.52	35.00	50
Peak bidir	86.27	85.83	77.36	66.14	65.84	64.76	100
GPU	0	1		2	3		(peak)
Butte htod	68.64	68.47	—	40.44	40.47	—	75
Butte dtoh	68.33	68.69	—	36.85	35.63	—	75
Butte bidir	128.98	114.99	—	64.79	64.60	—	150

We measure achieved CPU-GPU NVLink rates with a modified bandwidthTest from the NVIDIA CUDA Samples. As described earlier, peak NVLink rates between a CPU and a directly connected GPU are 50 GB/s (Summit, Peak) and 75 GB/s (Sierra, Butte). Table IV shows host to device (htod), device to host (dtoh) and bidirectional (bidir) transfer rates between core 0 and each GPU. Multiple trials show little variability. On-socket (Peak GPUs 0, 1 and 2; Butte GPUs 0 and 1) unidirectional and bidirectional bandwidths are 92% and 86% of theoretical peak, although bidirectional bandwidth to the final GPU of the socket is unexpectedly about 10% lower compared to the other on-socket GPUs when accessed from core 0. We are currently investigating possible affinities between cores and each GPU. Unsurprisingly, off-socket bandwidths are significantly lower, due to the intervening X-Bus. Thus, we expect users to avoid off-socket GPU access.

Comments:

- The X-Bus performance peak values of 128 GB/s bidir, 64 GB/s unidir, are highly idealized values and do not count substantial protocol overheads (not unlike PCIe-2 8 GB/s peak, ~5 GB/s actual). The numbers here are within 10% of what IBM has measured internally.
- The lower NVLINK rate achieved to one of the GPUs is a known behavior related to GPU address translation, believe will not affect real application use cases in production

NVLINK Benchmark

- Second test: multiple MPI ranks evenly spread across the 2 CPUs, each accessing a GPU on its socket, all at the same time (represents common application use case)

NVLINK Benchmark: Multiple GPU

TABLE V: NVLink Rates with MPI Processes (GB/s)

MPI Process Count	1	2	3	4	5	6
Peak htod	45.93	91.85	137.69	183.54	229.18	274.82
Peak dtoh	45.95	91.90	137.85	183.80	225.64	268.05
Peak bidir	85.70	172.59	223.54	276.34	277.39	278.07
Butte htod	68.66	137.39	206.05	275.47	—	—
Butte dtoh	68.91	137.48	203.80	271.12	—	—
Butte bidir	126.06	255.47	270.72	283.08	—	—

Table V shows the more typical use case of multiple MPI processes evenly spread between CPU sockets each simultaneously using one GPU. Multiple trials exhibit run-to-run variability under about 3%. For a saturated node with the largest MPI process count, for the unidirectional case the expected NVLink rate (300 GB/s peak, $6 \times 46 = 276$ GB/s actual on Peak, $4 \times 69 = 276$ GB/s actual on Butte) nearly matches the CPU stream performance of about 275 GB/s, thus CPU memory bandwidth does not limit the transfers. However, attainable bidirectional bandwidth is reduced by 46% compared to the sum of rates for individual GPUs (600 GB/s peak, $6 \times 86 = 516$ GB/s actual on Peak, $4 \times 129 = 516$ GB/s actual on Butte), due to bandwidth limits of CPU memory. Thus, overlapped host-device and device-host transfers (as opposed to in sequence) will provide little performance benefit in some cases. In either case, since attainable NVLink speeds for a saturated node are roughly the same for both systems, Summit's additional GPUs may provide little performance benefit for applications highly bound by NVLink bandwidth.

NVLINK Peer-to-peer Benchmark

- Transfer speed between GPUs
- Test code is modified from NVIDIA CUDA Samples
- Test of GPU data transfer between GPUs –
  - between 2 GPUs that are connected to CPU socket 0,
  - between 2 GPUs that are connected to CPU socket 1, and
  - between 2 GPUs that are connected to different CPUs on the node (through XBus)
- Tested with and without the “peer-to-peer” feature enabled in the CUDA call
- (note: for typical users, special syntax is required to enable p2p transfers, because of cgroups (otherwise transfers will go through the CPU))

NVLINK Peer-to-peer Benchmark

TABLE VI: NVLink Rates for GPU-GPU Transfers (GB/s)

	no P2P			P2P			
	socket 0	socket 1	cross-socket	socket 0	socket 1	cross-socket	(peak)
Peak unidir	33.18	25.84	30.32	46.33	46.55	25.89	50
Peak bidir	54.48	27.91	49.02	93.02	93.11	21.63	100
Butte unidir	41.27	24.72	31.04	69.49	69.49	31.05	75
Butte bidir	58.63	25.55	49.17	139.15	124.30	49.15	150

Table VI shows NVLink transfer rates between GPUs (within a socket and across them), using p2pBandwidthLatencyTest from CUDA Samples. We show the average of ten trials on a single node; the maximum deviance across different trials and GPU-GPU connections was 8.7%. The peer-to-peer (P2P) access feature yield performance that approaches NVLink theoretical peak bandwidth; results are much lower without it (no P2P). Predictably, cross-socket bandwidth is much lower than that between GPUs attached to the same CPU socket. GPUs on socket 1 without peer-to-peer access underperform compared to socket 0, possibly due to the benchmark running on socket 0 controlling GPUs attached to socket 1. Socket 1 peer-to-peer bidirectional performance on Butte is also low by about 12%. Otherwise, on-socket performance with peer-to-peer access enabled is roughly 93% of theoretical peak.

Comments:

- The lower Peak cross-socket performance may be related to the lower number of NVLINK connections available per GPU. Users wanting to do CPU-GPU transfers off-socket may want to experiment with transfers with/without P2P enabled cross-socket to evaluate performance

Interconnect Performance

- Infiniband point-to-point message performance
- Tested using the IMB Intel MPI Benchmarks suite
- Ping-pong test measures unidirectional bandwidth (peak 25 GB/sec) and zero-byte latency
- SendRecv test measures bidirectional bandwidth (peak 50 GB/sec)
- Achieved bandwidth measured as a function of message size
- 2 MPI ranks, 2 arbitrary nodes
- Summit fat tree interconnect with adaptive routing is less sensitive to node placement than Titan 3-D torus interconnect

Interconnect Performance

Benchmarking PingPong
#processes = 2

#bytes	#repetitions	t[usec]	Mbytes/sec
0	1000	1.21	0.00
1	1000	1.21	0.79
2	1000	1.13	1.69
4	1000	1.14	3.33
8	1000	1.14	6.68
16	1000	1.14	13.33
32	1000	1.19	25.68
64	1000	1.20	50.93
128	1000	1.28	95.14
256	1000	1.67	146.50
512	1000	1.83	266.85
1024	1000	2.02	482.76
2048	1000	2.92	669.44
4096	1000	3.39	1152.07
8192	1000	5.07	1542.09
16384	1000	6.26	2497.91
32768	1000	8.80	3552.76
65536	640	12.87	4857.82
131072	320	14.61	8557.85
262144	160	19.80	12629.17
524288	80	30.70	16289.25
1048576	40	51.92	19262.19
2097152	20	95.21	21006.54
4194304	10	178.86	22363.95

Benchmarking Sendrecv
#processes = 2

#bytes	#repetitions	t_min[usec]	t_max[usec]	t_avg[usec]	Mbytes/sec
0	1000	1.73	1.73	1.73	0.00
1	1000	1.72	1.72	1.72	1.11
2	1000	1.66	1.66	1.66	2.30
4	1000	1.66	1.66	1.66	4.60
8	1000	1.67	1.67	1.67	9.15
16	1000	1.71	1.71	1.71	17.86
32	1000	1.71	1.71	1.71	35.78
64	1000	1.65	1.65	1.65	73.84
128	1000	1.73	1.74	1.74	140.60
256	1000	1.94	1.94	1.94	251.31
512	1000	2.00	2.00	2.00	487.74
1024	1000	2.26	2.26	2.26	863.14
2048	1000	3.40	3.41	3.41	1146.80
4096	1000	3.80	3.80	3.80	2054.11
8192	1000	5.89	5.89	5.89	2652.01
16384	1000	7.47	7.47	7.47	4183.07
32768	1000	10.19	10.19	10.19	6130.49
65536	640	14.00	14.00	14.00	8926.57
131072	320	15.41	15.41	15.41	16218.11
262144	160	21.44	21.44	21.44	23322.26
524288	80	32.08	32.10	32.09	31155.20
1048576	40	53.43	53.43	53.43	37434.40
2097152	20	95.68	95.75	95.72	41773.45
4194304	10	180.60	180.61	180.60	44294.80

Conclusions / Recommendations

- High speeds achievable for CPU and GPU memories, esp. compared to Titan
- Avoid transfers through the XBus – ok to use for MPI communications since XBus is faster than the NIC, but CPU transfers to off-socket GPU are much slower than on-socket GPUs. This generally favors 2 or 6 MPI ranks per node instead of 1 rank.
- Enable peer-to-peer access if necessary to transfer directly between GPUs at high speed
- It may not be of benefit to overlap GPU transfers in the two directions with each other for this architecture, though this may still be desirable for performance portability, also still definitely useful to overlap transfers with compute
- Performance variations for operations (e.g., GPU memory access) and other unexpected performance variances may have a negative impact—this topic merits further study
- Advisable to maximize message size for point-to-point messages (e.g., > 4 MB), to approach asymptotic best behavior for the network. Also note if using one communication thread per node (uncommon), must use special syntax to use both NIC communication paths

Conclusions / Recommendations (2)

- Caveat: benchmark kernels provide performance results for idealized operations which may be different from the execution patterns in an application (e.g., nonuniform memory access, memory access mixed with other operations, GPU memory access with different threadblock configurations, etc.). Sometimes hard to troubleshoot causes of performance behaviors for complex code
- Yet kernel benchmarks are still useful for understanding how close you are to the "mark on the wall" of what peak value is realistically achievable
- Kernel benchmark results and performance models can also be useful for algorithm design – e.g., to understand tradeoffs for different ways to restructure an algorithm – e.g., to exclude a particular algorithm restructuring choice if the hardware speeds disallow it from be performant

Questions?

Wayne Joubert
joubert@ornl.gov

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Supplementary slides

Aside: impact of idle time on benchmark performance