

Burst Buffer on Summit

George S. Markomanolis,
HPC Engineer
Oak Ridge National Laboratory
Summit New User Training
Knoxville, TN
3 June 2020

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



U.S. DEPARTMENT OF
ENERGY

Burst Buffer on compute node

- Burst Buffers are technologies that provide faster I/O based on new media, on Summit we have on each compute node a Samsung PM1725a NVMe
- 4,608 nodes with local NVMe of 1.6 TB
 - 7.3 PB Total
 - Write performance per BB node: 2.1 GB/s
 - Read performance per BB node : 5.5 GB/s
- By default we can do one file per MPI process or one file per node, no single shared file between different Burst Buffer nodes without using any other Burst Buffer library (check Chris Zimmer's presentation tomorrow).
- Linear scalability by using Burst Buffers across many nodes
- Exclusive usage of the resources, no sharing with other users

Burst Buffer – Use cases

- Periodic burst
- Good for machine learning and deep learning workloads
- Transfer to PFS between bursts
- I/O improvements
- Improves applications with heavy metadata

Burst Buffer

- Burst Buffer can be used through the scheduler, integration with LSF
- What a user has to do?
 - Add the appropriate scheduler option in the submission script
 - Copy any necessary file (if required) on the Burst Buffer (input file, executable)
 - Execute the application and make sure that it reads/writes the files with significant size from/on Burst Buffer
 - Copy the necessary files back to GPFS to save them

Submission script for Burst Buffer – NAS BTIO

GPFS

```
#!/bin/bash
#BSUB -P projid
#BSUB -J nas_btio
#BSUB -o nas_btio.o%J
#BSUB -W 10
#BSUB -nnodes 1

jsrun -n 1 -a 16 -c 16 -r 1 ./btio
```

Burst Buffer

```
#!/bin/bash
#BSUB -P projid
#BSUB -J nas_btio
#BSUB -o nas_btio.o%J
#BSUB -W 10
#BSUB -alloc_flags "nvme"
#BSUB -nnodes 1

jsrun -n 1 cp btio inputbt.data /mnt/bb/$USER/

jsrun -n 1 -a 16 -c 16 -r 1 /mnt/bb/$USER/btio

jsrun -n 1 cp /mnt/bb/$USER/btio.nc
/gpfs/alpine/scratch/...
```

NAS BTIO

- Executing 16 MPI processes on a **single** BB node, blocking PNetCDF with a single shared file

Total I/O amount	:	152.6 GB
Time in sec	:	67.98
I/O bandwidth	:	2.24 GB/s

Understanding the MPI I/O Hints

- Using the command `export ROMIO_PRINT_HINTS=1` in the submission script, we can acquire the following information for 16 MPI processes of one BB node

```
key = cb_buffer_size      value = 16777216
key = romio_cb_read       value = enable
key = romio_cb_write      value = enable
key = cb_nodes           value = 1
...
key = cb_config_list     value = *:1
key = romio_aggregator_list value = 0
```

NAS BTIO - Improved

- Increasing the MPI I/O aggregators to 8
`echo "cb_config_list *:8" > romio_hints`

- Declare the ROMIO_HINTS variable
`export ROMIO_HINTS=$PWD/romio_hints`

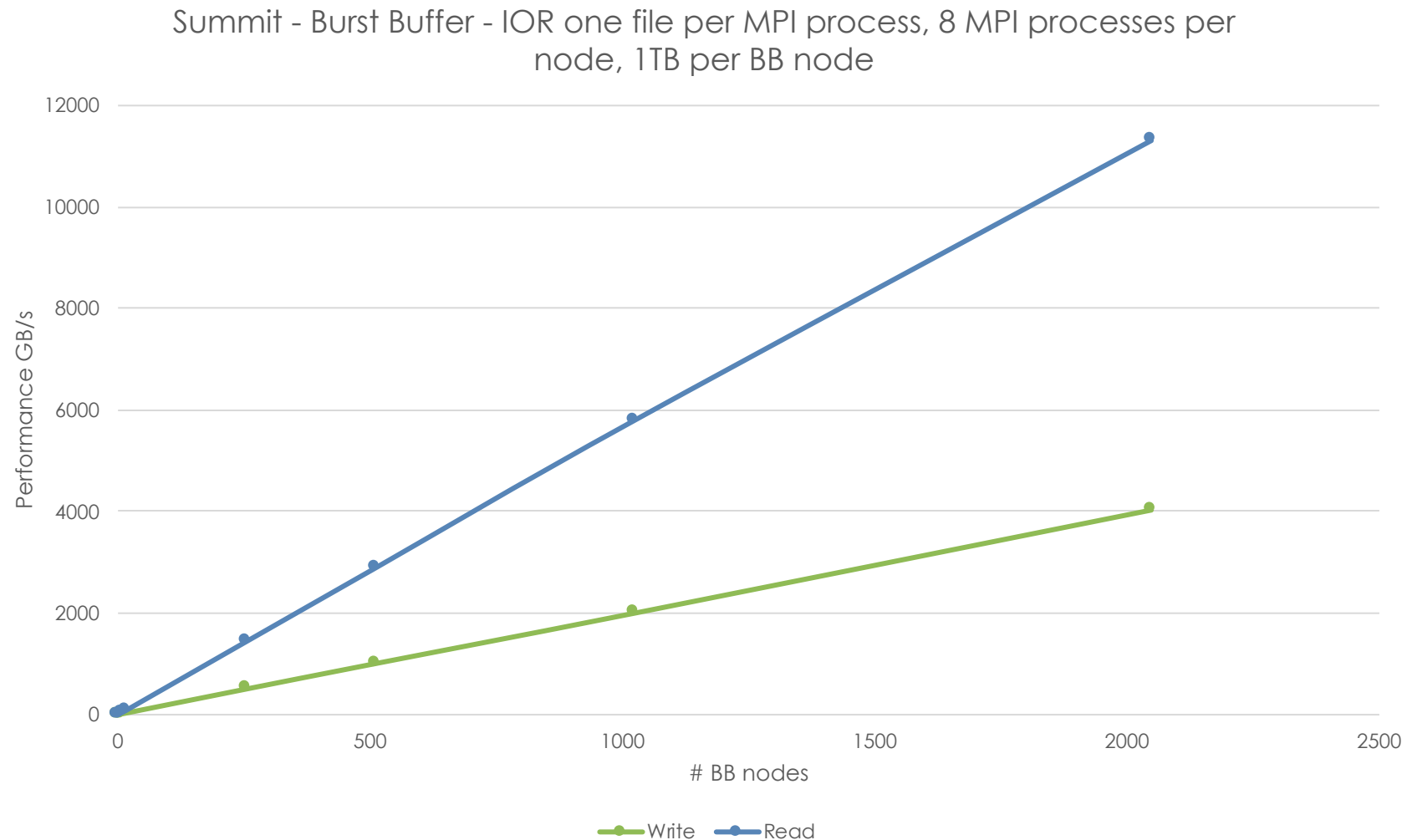
- New performance results

Total I/O amount	:	152.6 GB
Time in sec	:	52.47
I/O bandwidth	:	2.98 GB/s

Almost 23% improvement by using page cache and NVMe

Burst Buffer

- Scalability test with IOR



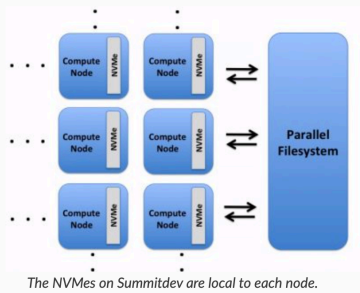
Documentation

Accounts and Projects
Connecting
Systems
Summit User Guide
Summit Documentation Resources
System Overview
NVIDIA V100 GPUs
Connecting
Data and Storage
Burst Buffer
NVMe (XFS)
Current NVMe Usage
Interactive Jobs Using the NVMe
NVMe Usage Example
Spectral Library
Software
Shell & Programming Environments
Compiling
Running Jobs
Debugging
Optimizing and Profiling
Known Issues
Training System (Ascent)
Preparing For Frontier
SummitDev User Guide
Rhea User Guide
Home
Data Transfer Nodes (DTNs)
High Performance Storage System
Ascent

Burst Buffer

NVMe (XFS)

Each compute node on Summit has a 1.6TB Non-Volatile Memory (NVMe) storage device, colloquially known as a "Burst Buffer" with theoretical performance peak of 2.1 GB/s for writing and 5.5 GB/s for reading. 100GB of each NVMe is reserved for NFS cache to help speed access to common libraries. When calculating maximum usable storage size, this cache and formatting overhead should be considered; We recommend a maximum storage of 1.4TB. The NVMe's could be used to reduce the time that applications wait for I/O. Using an SSD drive per compute node, the burst buffer will be used to transfer data to or from the drive before the application reads a file or after it writes a file. The result will be that the application benefits from native SSD performance for a portion of its I/O requests. Users are not required to use the NVMe's. Data can also be written directly to the parallel filesystem.



Current NVMe Usage

Tools for using the burst buffers are still under development. Currently, the user will have access to a writeable directory on each node's NVMe and then explicitly move data to and from the NVMe's with posix commands during a job. This mode of usage only supports writing file-per-process or file-per-node. It does not support automatic "n to 1" file writing, writing from multiple nodes to a single file. After a job completes the NVMe's are trimmed, a process that irreversibly deletes data from the devices, so all desired data from the NVMe's will need to be copied back to the parallel filesystem before the job ends. This largely manual mode of usage will not be the recommended way to use the burst buffer for most applications because tools are actively being developed to automate and improve the NVMe transfer and data management process. Here are the basic steps for using the BurstBuffers in their current limited mode of usage:

1. Modify your application to write to /mnt/bb/\$USER, a directory that will be created on each NVMe.
2. Modify either your application or your job submission script to copy the desired data from /mnt/bb/\$USER back to the parallel filesystem before the job ends.

Using GPFS	Using NVMe
	<code>#!/bin/bash</code>
	<code>#BSUB -P xxx</code>
	<code>#BSUB -J NAS-BTIO</code>
	<code>#BSUB -o nasbtio.o%J</code>
	<code>#BSUB -e nasbtio.e%J</code>
	<code>#BSUB -W 10</code>
	<code>#BSUB -nnodes 1</code>
	<code>#BSUB -alloc_flags nvme</code>
	<code>export BBPATH=/mnt/bb/\$USER/</code>
	<code>jsrun -n 1 cp btio \${BBPATH}</code>
	<code>jsrun -n 1 cp input* \${BBPATH}</code>
<code>jsrun -n 1 -a 16 -c 16 -r 1 ./btio</code>	<code>jsrun -n 1 -a 16 -c 16 -r 1 \${BBPATH}/btio</code>
<code>ls -l</code>	<code>jsrun -n 1 ls -l \${BBPATH}/</code>
	<code>jsrun -n 1 cp \${BBPATH}/* .</code>

https://docs.olcf.ornl.gov/systems/summit_user_guide.html#burst-buffer

Conclusions

- Burst Buffer is the solution for heavy I/O applications
- We need some extra libraries on Summit to support various workflows
- Tuning with MPI I/O hints could provide faster execution time

Thank you!
Questions?