

# Practical Tips for Running on SUMMIT

David Appelhans

ORNL SUMMIT Training Workshop  
Feb 11, 2019



# OUTLINE

- Talk is a collection of hands on experience in readying applications for SUMMIT.
- Lightweight MPI profiler.
- Useful jsrun/MPI flags.
- LSF+jsrun example submission scripts
  - 1 rank per socket, core, or GPU example scripts.
  - Using multiple ranks per GPU with MPS.
- Profiling of 1 rank with nvprof at scale.
- Checking on job submission
  - Useful lsf commands.
  - ssh to compute node, get stack traces, nvidia smi.

## BEST PRACTICE: LIGHTWEIGHT MPI TRACING

A lightweight MPI tracing library can easily be linked against at runtime<sup>1</sup>.

- No performance penalty—we ran this library with all CORAL benchmarks during acceptance.
- Very useful in identifying node imbalance at scale.
- How to use it
  - Official version shipped with SMPI:

---

```
export OMPI_LD_PRELOAD_POSTPEND=$OLCF_SPECTRUM_MPI_ROOT/lib/libmpitrace.so
```

---

It writes several profile files with format of `mpi_profile.jobid.rank`

---

<sup>1</sup>Vampire is another available profiling tool

## USING THE MPI PROFILER

Can see where time is spent, calculate achieved communication bandwidth, average message size, etc.

---

MPI Routine	calls	avg. bytes	time(sec)
MPI_Comm_rank	320850	0.0	0.010
MPI_Comm_size	4	0.0	0.000
MPI_Isend	84	540688.0	0.036
MPI_Send_init	168	8404992.0	0.000
MPI_Recv_init	168	8404992.0	0.000
MPI_Irecv	84	540688.0	0.001
MPI_Wait	3672	0.0	18.112
MPI_Waitall	42	0.0	0.007
MPI_Start	3588	0.0	0.038
MPI_Bcast	24	37.2	0.000
MPI_Barrier	411	0.0	34.302
MPI_Reduce	1	4.0	0.001
MPI_Allreduce	696	20.0	8.602
MPI_Gather	1	4.0	0.000
MPI_Gatherv	2	288.0	0.000

---

total communication time = 61.108 seconds.  
total elapsed time = 239.719 seconds.  
user cpu time = 2211.329 seconds.  
system time = 372.080 seconds.  
max resident set size = 37583.938 MBytes.

---

## IDENTIFYING WORK IMBALANCE

Ranks taking the most time in computation spend the **least** amount of time in MPI calls because all other ranks have been waiting for them. Example rank 0 output info:

---

Histogram of times spent in MPI

time-bin	ranks
30.231	1
32.611	0
34.991	0
37.371	0
39.751	1
42.131	1
44.511	1
46.891	6
49.271	55
51.651	261
54.030	264
56.410	188
58.790	67
61.170	15
63.550	4

<= Rank deserving closer look

---

Histogram of times spent in MPI (Balanced Run)

time-bin	ranks
18.413	1
19.261	1
20.110	1
20.958	9
21.807	28
22.655	38
23.504	30
24.352	48
25.201	39
26.049	26
26.898	17
27.746	11
28.595	3
29.444	3
30.292	1

# USING THE MPI PROFILER

Rank 0 summary file also provides rank to node correlation:

---

MPI timing summary for all ranks:

taskid	hostname	cpu	comm(s)
...			
290	sierra3358	88	52.04
291	sierra3358	128	52.29
292	sierra3800	0	43.48
293	sierra3800	40	30.23
294	sierra3800	88	54.34
295	sierra3800	128	51.69
...			

---

## USEFUL JSRUN FLAGS

- Send kill signal to processes on a MPI failure (helps kill your job instead of hanging it).

---

```
jsrun -X 1 <further commands>
```

---

- Prepend the rank id to the output

---

```
jsrun --stdio_mode prepended
```

---

- If Spectrum MPI arguments are needed (e.g. async argument when using 1-sided communication)

---

```
jsrun --smpiargs="--async"
```

---

## EXAMPLE SUBMISSION SCRIPTS

- ORNL training material is very good ([link](#) + [visualizer](#) ).
- Using scripts for lsf job submission and jsrun configuration allows repeatable experiments.
- See talk by Chris Fuson for detailed explanation of lsf and jsrun.
- Key to jsrun is deciding what a resource set will represent (e.g. 1 rs per socket, or per GPU).
- The following are some examples of common use cases. Links to full submission scripts are on each page.

# 1 RANK PER CORE (CLICKABLE LINK)

---

```
nodes=1
gpus_per_socket=3 # number of gpus to use per socket
application_cores=21 # cores available to the application per socket
threads_per_core=4 # Each core can go up to smt4 for 4 hardware threads.
# user sets rank_per_socket, calculate other quantities from this:
ranks_per_socket=21 # needs to be evenly divisible by gpus_per_socket (if using GPUs)

# calculated from input:
let num_sockets=2*$nodes
let cores_per_rank=$application_cores/$ranks_per_socket # avail cores divided into the ranks.
let cores_per_socket=$cores_per_rank*$ranks_per_socket # this is used cores per socket (not
necessarily equal to application cores
let threads_per_rank=$threads_per_core*$cores_per_rank

... LSF submission stuff ...

jsrun --stdio_mode=prepend -D CUDA_VISIBLE_DEVICES \
-E OMP_NUM_THREADS=${threads_per_rank} \
--nrs ${num_sockets} \
--tasks_per_rs ${ranks_per_socket} \
--cpu_per_rs ${cores_per_socket} \
--gpu_per_rs ${gpus_per_socket} \
--bind=proportional --packed:${cores_per_rank} \
-d plane:${ranks_per_socket} \
./ print --affinity .sh
```

---

# 1 RANK PER GPU (CLICKABLE LINK)

---

```
nodes=1
gpus_per_socket=3 # number of gpus to use per socket ( 3 for summit, 2 for sierra )
gpus_per_rs=1 # 1 res set per GPU (one to one gpu to rs mapping).
threads_per_core=2 # Each core can go up to smt4 for 4 hardware threads .
ranks_per_rs=1 # If using more than 1 rank per gpu, need to enable mps through lsf .

# derived quantities :
let rs_per_socket=$gpus_per_socket/$gpus_per_rs
let ranks_per_socket=$ranks_per_rs*$rs_per_socket
# There are 21 (of 22) cores available to the application per socket (on Summit)
let cores_per_rank=21/$ranks_per_socket # 21 avail cores divided into the ranks .
let cores_per_rs=$cores_per_rank*$ranks_per_rs
let nrs=2*$rs_per_socket*$nodes # total number of resource sets :
let threads_per_rank=$threads_per_core*$cores_per_rank

... LSF submission stuff ...

jsrun --stdio_mode=prepend -D CUDA_VISIBLE_DEVICES \
-E OMP_NUM_THREADS=${threads_per_rank} \
--nrs ${nrs} --tasks_per_rs ${ranks_per_rs} \
--cpu_per_rs ${cores_per_rs} \
--gpu_per_rs ${gpus_per_rs} \
--bind=proportional--packed:${cores_per_rank} \
-d plane:${ranks_per_rs} \
./ print -- affinity .sh
```

---

# SHARING GPU AMONG 2 RANKS [CLICK FOR FULL EXAMPLE](#)

One resource set per GPU, multiple tasks per resource set.

---

```

gpus_per_socket=3 # number of gpus to use per socket ( 3 for summit, 2 for sierra )
gpus_per_rs=1 # 1 res set per GPU (one to one gpu to rs mapping).
threads_per_core=4 # Each core can go up to smt4 for 4 hardware threads .
ranks_per_rs=2 # If using more than 1 rank per gpu, need to enable mps through lsf .

```

```

# derived quantities :
let rs_per_socket=$gpus_per_socket/$gpus_per_rs
let ranks_per_socket=$ranks_per_rs*$rs_per_socket
# There are 21 (of 22) cores available to the application per socket (on Summit)
let cores_per_rank=21/$ranks_per_socket # 21 avail cores divided into the ranks.
let cores_per_rs=$cores_per_rank*$ranks_per_rs
let nrs=2*$rs_per_socket*$nodes # total number of resource sets :
let threads_per_rank=$threads_per_core*$cores_per_rank

```

---

Must enable MPS in lsf submission<sup>2</sup>:

---

```
#BSUB -alloc_flags gpumps
```

---

```

jsrun --stdio_mode=prepend -D
      CUDA_VISIBLE_DEVICES \
      -E OMP_NUM_THREADS=${threads_per_rank} \
      --nrs ${nrs} \
      --tasks_per_rs ${ranks_per_rs} \
      --cpu_per_rs ${cores_per_rs} \
      --gpu_per_rs ${gpus_per_rs} \
      --bind=proportional --packed:${cores_per_rank} \
      -d plane:${ranks_per_rs} \
      ./ print --affinity .sh

```

---



---

<sup>2</sup>If specifying multiple flags, must do it in one line. e.g. #BSUB -alloc\_flags “smt2 gpumps”

## CHECKING ON YOUR SUBMISSION

- *bjobs* will show a list of your current jobs and their status.
- To check pending reason of a specific job, *bjobs -p <jobid>*
- Is the que busy/open? *bqueues*

---

```
[dappelh@login5.summit]$ bqueues
```

QUEUE_NAME	PRIO	STATUS	MAX JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
test	5	Open:Inact	45	—	—	—	0	0	0
storage	4	Open:Inact	—	—	—	—	0	0	0
tested	3	Open:Active	—	—	—	—	0	0	0
new	2	Open:Inact	—	—	—	—	43	43	0
batch	1	Open:Active	—	—	—	—	85	0	85

---

- What jobs are scheduled on the machine? *bjobs -u all*

---

```
[dappelh@login1.summit]$ bjobs -u all
```

JOBID	USER	STAT	SLOTS	QUEUE	START_TIME	FINISH_TIME	JOB_NAME
220841	bykov	RUN	86017	batch	Nov 28 14:15	Nov 29 02:15	lsdalton
220932	jaharris	RUN	43	batch	Nov 28 17:37	Nov 28 18:07	Not_Specified
220933	ngawande	RUN	2017	batch	Nov 28 17:40	Nov 28 19:40	d48_c
220908	chochia	PEND	—	tested	—	—	Not_Specified
220926	chochia	PEND	—	tested	—	—	Not_Specified

---

## CHECKING ON A RUNNING JOB

- *bpeek* will show you tail of standard out and standard error (jobid is optional):

---

```
bpeek <jobid> | less
```

---

- Something seems wrong, can I check into a job further?

---

```
bjobs -WP <jobid> | less
```

---

---

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME	%COMPLETE
220933	ngawand	RUN	batch	login1	batch1	d48_c	Nov 28 17:40	32.62% L
					g36n12			
					g36n12			
					...			

---

- Can ssh to a compute node (only while job is running)

---

```
ssh g36n12
```

---

## TROUBLESHOOTING A JOB ON A COMPUTE NODE

- Use `gstack` to get call stack of each thread (use `top` to get pid)

---

```
gstack <pid>
```

---

- See if the GPUs have something currently running:

---

```
nvidia -smi
```

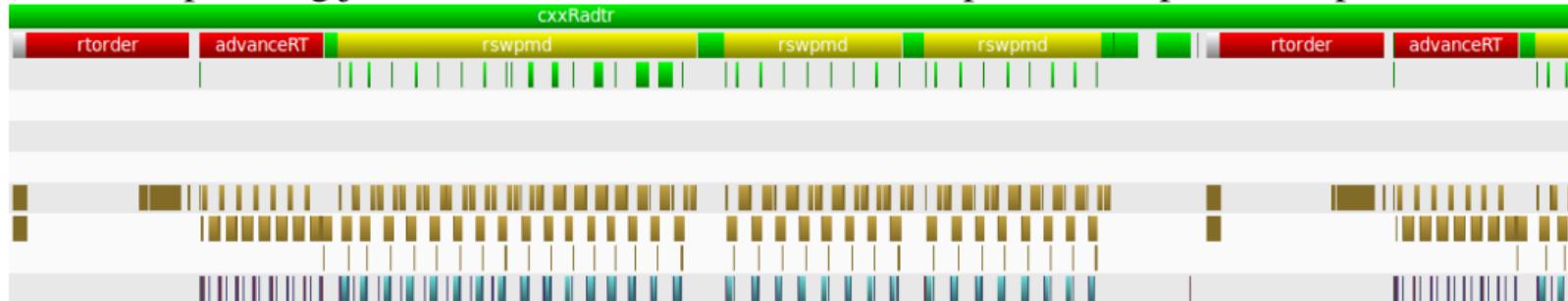
---

### Example idle GPUs:

```
+-----+
| NVIDIA-SMI 396.58                Driver Version: 396.58                |
+-----+-----+-----+-----+-----+
| GPU  Name            Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+
|   0   Tesla V100-SXM2...    On      | 000000004:04:00.0 Off  |            0         |
| N/A   34C    P0     36W / 300W |      0MiB / 16128MiB |          0%      E. Process |
+-----+-----+-----+-----+-----+
+-----+
| Processes:
| GPU      PID   Type   Process name                      GPU Memory
|-----|-----|-----|-----|-----|
| No running processes found
+-----+
```

# PROFILING AT SCALE

- Running nvprof on every rank of a large scale job will slow to a crawl.
- However, profiling just one of the ranks at scale with nvprof/visual profiler is possible.



- See talk later this week on detailed usage of nvprof and visual profiler as well as Score-P/Vampire.

# SCRIPT FOR 1 RANK TO LAUNCH NVPROF

Have jsrun launch this script which executes profiler+application if rank matches profile rank, otherwise just launches application without profiling:

```
export PROFILE_RANK=1
export PROFILE_PATH="/gpfs/path_to_your_directory"
jsrun <flags> profile_helper.sh a.out
```

Contents of profile\_helper.sh:

---

```
#!/bin/bash
if [ $PMIX_RANK == $PROFILE_RANK ]; then
    nvprof -f -o $PROFILE_PATH "$@"
else
    "$@"
fi
```

---

# CLOSING REMARKS

Hopefully you have learned how to

- Use the lightweight mpi profiler even if not focusing on MPI performance.
- Use scripts for lsf/jsrun submissions.
- Check on your job submission after submitting.
- Launching a profiler at scale on a single MPI rank.

Questions?

David Appelhans - [dappelh@us.ibm.com](mailto:dappelh@us.ibm.com)

Github script location: <https://github.com/dappelha/summit-scripts>