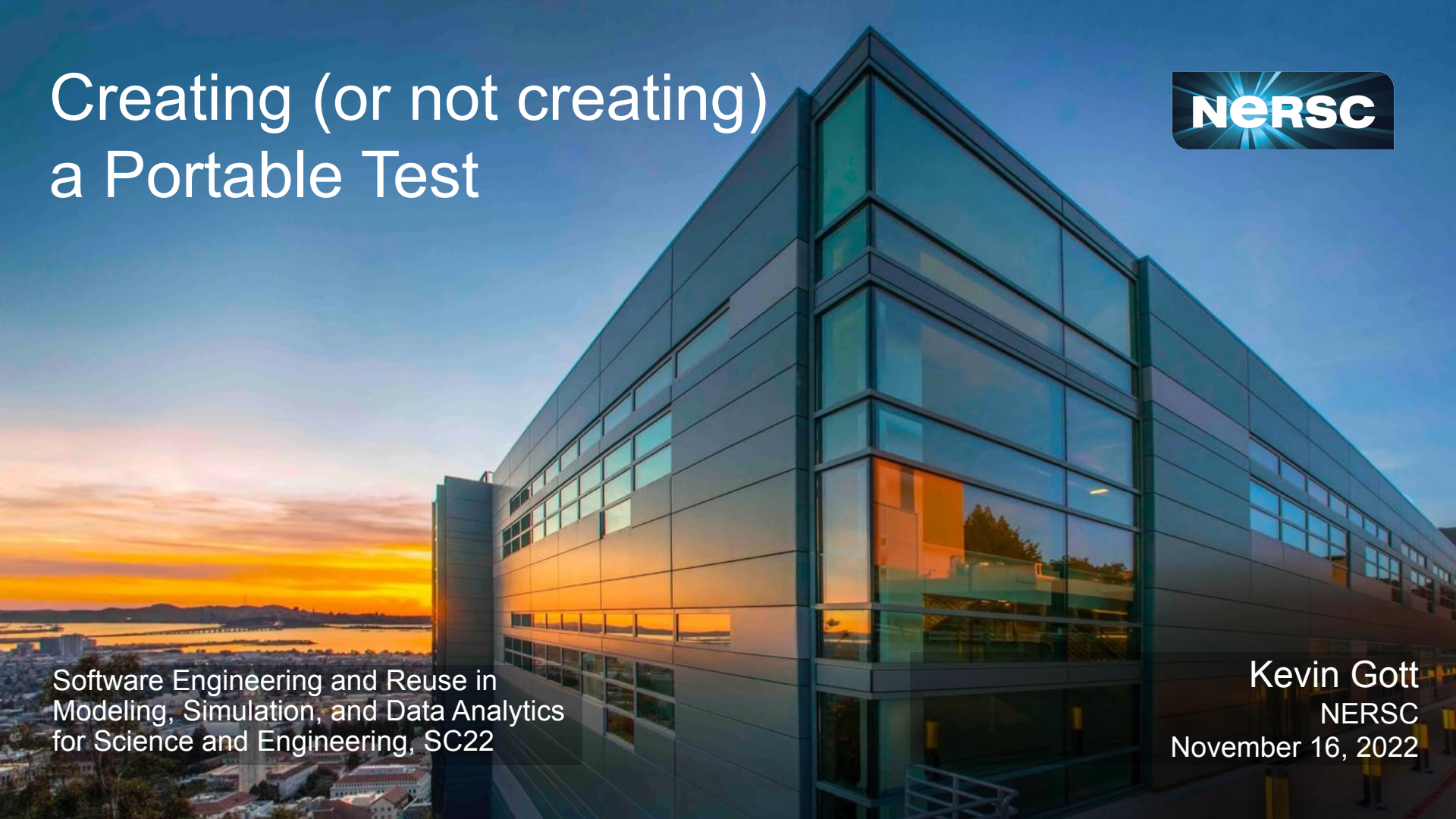
Creating (or not creating) a Portable Test



Software Engineering and Reuse in Modeling, Simulation, and Data Analytics for Science and Engineering, SC22

Kevin Gott
NERSC
November 16, 2022

I'd like to tell you a story:

"Challenges of porting to diverse architectures"

A few weeks ago, I was making a CUDA Aware MPI Test.

Simple test: should we use Cuda Aware MPI or not.

For the application, this test is just two input flags:

```
amrex.the_arena_is_managed=0
amrex.use_gpu_aware_mpi=1
```

Want a portable test (not too much to ask, right?)

- Decide on a default: on or off?
- Give to other users.
- At least Perlmutter & Frontier.

But, building a portable run script isn't so easy.

There's a known issue with Slurm + CUDA Aware MPI dealing with GPU binding:

So, don't use these:	Unless you use this:	Instead, do this:
--gpus-per-task = --gpus-per-socket = --gpu-bind = <stuff> --cpus-per-gpu = --ntasks-per-gpu =	gpu-bind=none ➔ Just turns off bad ones. ➔ Still need to hand-set binding. ➔ And users play with this flag.	--ntasks-per-node=4 --gpus-per-node=4 *Not* --exclusive; some systems alter that. And then, set CUDA_VISIBLE_DEVICES for binding.

So, now you have to hand-tune affinity.

```
Rank 0 out of 4 processes: I see 4 GPU(s).
0 for rank 0: 0000:03:00.0
1 for rank 0: 0000:41:00.0
2 for rank 0: 0000:82:00.0
3 for rank 0: 0000:C1:00.0
Rank 1 out of 4 processes: I see 4 GPU(s).
0 for rank 1: 0000:03:00.0
1 for rank 1: 0000:41:00.0
2 for rank 1: 0000:82:00.0
3 for rank 1: 0000:C1:00.0
Rank 3 out of 4 processes: I see 4 GPU(s).
0 for rank 3: 0000:03:00.0
1 for rank 3: 0000:41:00.0
2 for rank 3: 0000:82:00.0
3 for rank 3: 0000:C1:00.0
Rank 2 out of 4 processes: I see 4 GPU(s).
0 for rank 2: 0000:03:00.0
1 for rank 2: 0000:41:00.0
2 for rank 2: 0000:82:00.0
3 for rank 2: 0000:C1:00.0
```

Source of “truth”

	GPU0	GPU1	GPU2	GPU3	CPU Affinity	NUMA Affinity
GPU0	X	NV4	NV4	NV4	48-63,112-127	3
GPU1	NV4	X	NV4	NV4	32-47,96-111	2
GPU2	NV4	NV4	X	NV4	16-31,80-95	1
GPU3	NV4	NV4	NV4	X	0-15,64-79	0

Confirm mappings

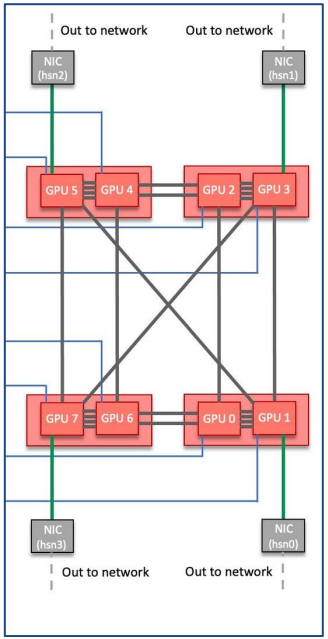
```
Hello from rank 0, on nid001277. (core affinity = 0-15,64-79)
Hello from rank 1, on nid001277. (core affinity = 16-31,80-95)
Hello from rank 2, on nid001277. (core affinity = 32-47,96-111)
Hello from rank 3, on nid001277. (core affinity = 48-63,112-127)
```

After a few hours:

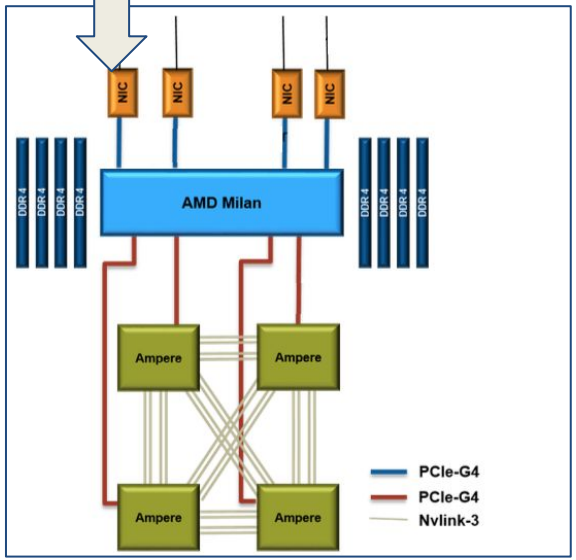
```
export CUDA_VISIBLE_DEVICES=$((3-$SLURM_LOCALID))
export MPICH_OFI_NIC_POLICY=GPU     #Or NUMA
```

Perlmutter

Which, of course, isn't portable:

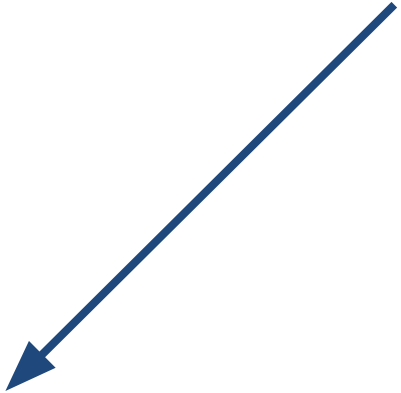


Frontier



Of course… intermission for a meeting:

During which I look up the “deadline” flag in the sbatch man page:



--deadline=<*OPT*>

remove the job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

Then, the meeting goes on…

I get distracted….

And I glance up the page….

Hey, isn't that the version of Slurm that we're upgrading to in 36 hours?

NOTE: Beginning with 22.05, srun will not inherit the --cpus-per-task value requested by salloc or sbatch. It must be requested again with the call to srun or set with the SRUN_CPUS_PER_TASK environment variable if desired for the task(s).

--deadline=<*OPT*>

remove the job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

And…..

Well…that's an important flag

-c, **--cpus-per-task**=<*ncpus*>

Advise the Slurm controller that ensuing job steps will require *ncpus* number of processors per task. Without this option, the controller will just try to allocate one processor per task.

For instance, consider an application that has 4 tasks, each requiring 3 processors. If our cluster is comprised of quad-processors nodes and we simply ask for 12 processors, the controller might give us only 3 nodes. However, by using the --cpus-per-task=3 options, the controller knows that each task requires 3 processors on the same node, and the controller will grant an allocation of 4 nodes, one for each of the 4 tasks.

NOTE: Beginning with 22.05, srun will not inherit the --cpus-per-task value requested by salloc or sbatch. It must be requested again with the call to srun or set with the SRUN_CPUS_PER_TASK environment variable if desired for the task(s).

--deadline=<*OPT*>

remove the job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

And then…

-c, --cpus-per-task=<*ncpus*>

Advise the Slurm controller that ensuing job steps will require *ncpus* number of processors per task. Without this option, the controller will just try to allocate one processor per task.

For instance, consider an application that has 4 tasks, each requiring 3 processors. If our cluster is comprised of quad-processors nodes and we simply ask for 12 processors, the controller might give us only 3 nodes. However, by using the --cpus-per-task=3 options, the controller knows that each task requires 3 processors on the same node, and the controller will grant an allocation of 4 nodes, one for each of the 4 tasks.

NOTE: Beginning with 22.05, srun will not inherit the --cpus-per-task value requested by salloc or sbatch. It must be requested again with the call to srun or set with the SRUN_CPUS_PER_TASK environment variable if desired for the task(s).

--deadline=<*OPT*>

remove the job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

Well…that's an important flag

… that's heavily changing behavior?!?!

And then I…

-c, --cpus-per-task=<*ncpus*>

Advise the Slurm controller that er … *cpus* number of processors per task. Without thi … y to allocate one processor per task.

For instance, consider an applic … uiring 3 processors. If our cluster is com … odes and we simply ask for 12 processors, the co … only 3 nodes. However, by using the --cpus-per-task=3 optio … the controller knows that each task requires 3 processors on the same node, and the controller will grant an allocation of 4 nodes, one for each of the 4 tasks.

NOTE: Beginning with 22.05, srun will not inherit the --cpus-per-task value requested by salloc or sbatch. It must be requested again with the call to srun or set with the SRUN_CPUS_PER_TASK environment variable if desired for the task(s).

--deadline=<*OPT*>

remove the job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

Well…that's an important flag

… oh… we do this "incorrectly" throughout the docs now too, don't we?

… that's heavily changing behavior?!?!

And then I interrupted…

-c, --cpus-per-task=<*ncpus*>

Advise the Slurm controller that er … *cpus* number of processors per task. Without thi … y to allocate one processor per task.

For instance, consider an applic … uiring 3 processors. If our cluster is com … odes and we simply ask for 12 processors, the co … only 3 nodes. However, by using the --cpus-per-task=3 optio … the controller knows that each task requires 3 processors on the same node, and the controller will grant an allocation of 4 nodes, one for each of the 4 tasks.

22.05, srun will not inherit the --cpus-per-task value … sbatch. It must be requested again with the call to srun … CPUS_PER_TASK environment variable if desired for

… job if no ending is possible before this deadline (start > (deadline - time[-min])). Default is no deadline. Valid time formats are:

HH:MM[:SS] [AM|PM]

Well…that's an important flag

… oh… we do this "incorrectly" throughout the docs now too, don't we?

… that's heavily changing behavior?!?!

WHAT? THIS ISN'T IN THE PATCH NOTES?

NERSC

DEPARTMENT OF ENERGY | Office of Science

And then I interrupted the…

-c, --cpus-per-task=<*ncpus*>

Advise the Slurm controller that er … *cpus* number of processors per task. Without this … y to allocate one processor per task.

For instance, consider an applic … uiring 3 processors. If our cluster is com … odes and we simply ask for 12 processors, the co … only 3 nodes. However, by using the --cpus-per-task=3 optio … the controller knows that each task requires 3 processors on the same node, and the controller will grant an allocation of 4 nodes, one for each of the 4 tasks.

… 22.05, srun will not i … e … sbatch. It must be … srun … CPUS_PER_TAS … for

… job if no ending is possible before … art> (deadline - time[-min])). Default is no deadline. Valid time formats are: HH:MM[:SS] [AM|PM]

Well…that's an important flag

… oh… we do this "incorrectly" throughout the docs now too, don't we?

… that's heavily changing behavior?!?!

WHAT? THIS ISN'T IN THE PATCH NOTES?

SO NO ONE KNOWS ABOUT THIS!!!!!!

And then I interrupted the meeting…

-c, --cpus-per-task=<*ncpus*>

Advise the Slurm controller that er … *cpus* number of processors p… sk. Without this … y to allocate one proces… task.

For ins… r an applic… uiring 3 proces… ster is com… odes and simply a… processors, the co… only 3 no… However, by using the --cpus-per-task=3 optio… the controller … each task requires 3 processors on the same node, and the co… nt an allocation of 4 nodes, one for each of the 4 tasks.

22.05, s… not i… e … sbatch. … srun … CPU … for

… job if no ending is possible before … art> (deadline - time[-min])). Default is no deadline. Valid time formats are: HH:MM[:SS] [AM|PM]

Well…that's an important flag

… oh… we do this "incorrectly" throughout the docs now too, don't we?

… that's heavily changing behavior?!?!

WHAT? THIS ISN'T IN THE PATCH NOTES?

SO NO ONE KNOWS ABOUT THIS!!!!!!

* Strictly speaking, may be inaccurate due to panic-induced haze.

That meeting went over by 40 minutes*

NERSC decided to do under-the-hood magic to keep -c the same.

But, other systems *clearly* won't, so I gotta do it manually to make it portable:

```
# Need to do -c and --cpu-bind here, or it will be ignored in Slurm 22.05+.
SRUN_FLAGS="--cpus-per-task=32 --cpu-bind=cores"
srun ${SRUN_FLAGS} ../run_me.sh $PROPER_AFFINITY $SLURM_JOB_NUM_NODES
```

(Which will obviously work flawlessly, because ***everyone*** reads notes in SLURM scripts.)

So, the “portable” Slurm script needs:

1) Specific SLURM flags to avoid bad binding.
   - ➢ only --gpus or –gpus-per-node

2) Hand tuned CPU-to-GPU affinity.
   - ➢ CUDA_VISIBLE_DEVICES

3) Careful use of common Slurm flags.
   - ➢ -c *must* be on the srun line

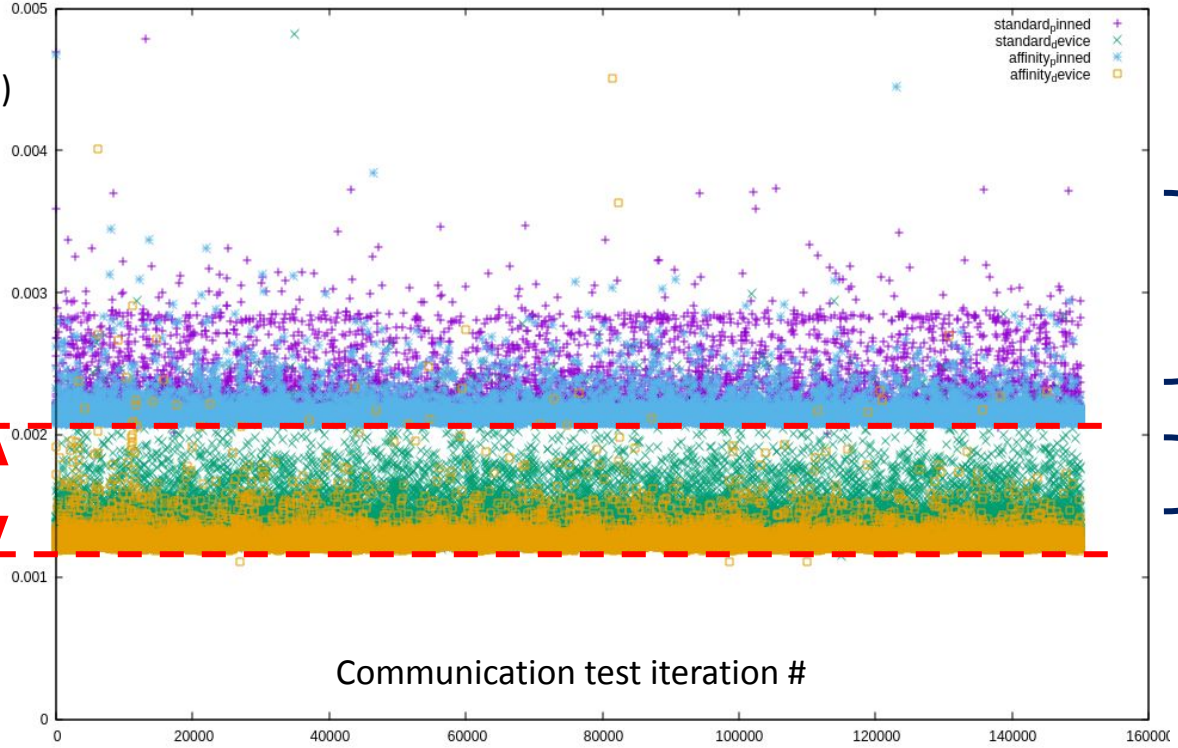
- Specialized knowledge
- System specific expertise.
- Careful control of commonly used and manipulated flags.
- Code specific tuning.
- Machine specific tuning.

★ NONE of these is guaranteed to throw an error or report any problem.

And it has a huge impact on performance*:

**Very* preliminary results

Time of the iteration (sec)



~2x improvement with CUDA Aware

Way more variability if proper affinity is not used.

So, if the code team's response speaks to you:

"I remember when running on super computers was easy."

Hi! You have a colleague in me!