

Building and  
running GPU  
applications on  
Perlmutter



On break:  
Starting  
10am PT

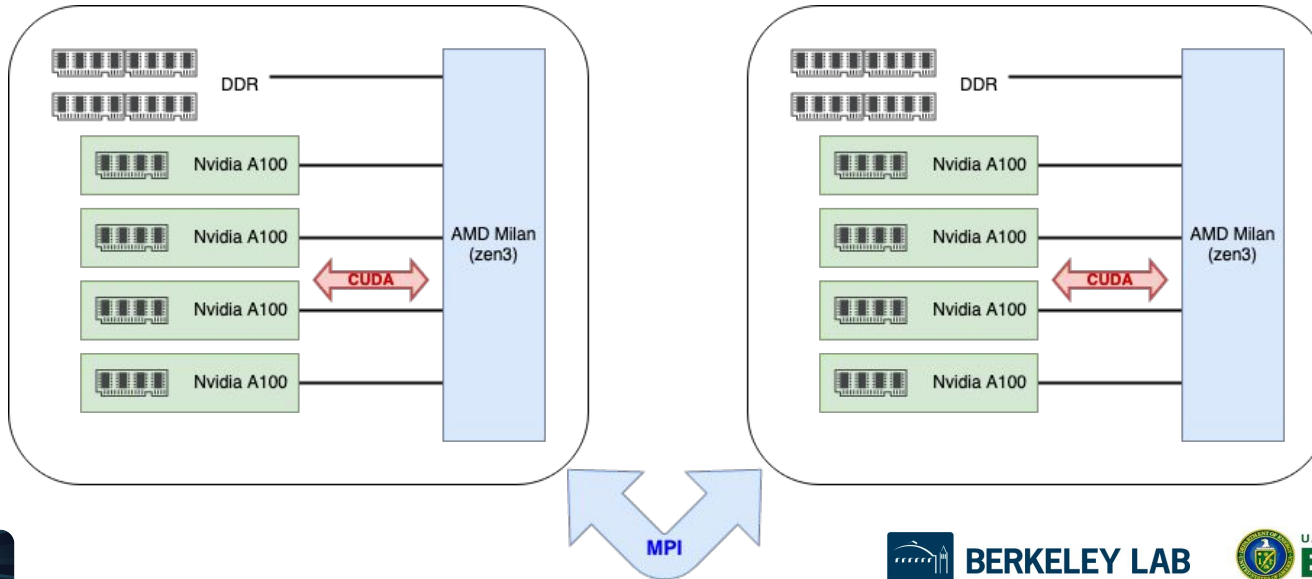


# Today

- 9:30am PT (now): Session 1 - Building and running an application on Perlmutter with MPI + GPUs (CUDA)
- 10:30am PT: 30 minute Break
- 11:00am PT: Session 2 - Additional Scenarios:
  - BLAS/LAPACK/FFTW etc with GPUs
  - Other compilers (not NVidia)
  - CUDA-aware MPI
  - Not CUDA (OpenMP offload, OpenACC)
  - cmake
  - Spack

# Goal for this session:

- Build and run a simple application with:
  - MPI to communicate between tasks
  - CUDA to offload computation to GPUs within a task

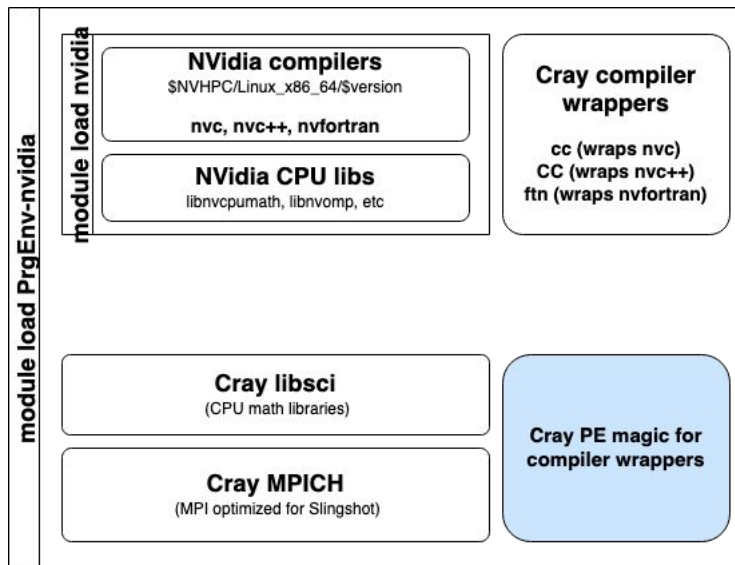


# Compiling

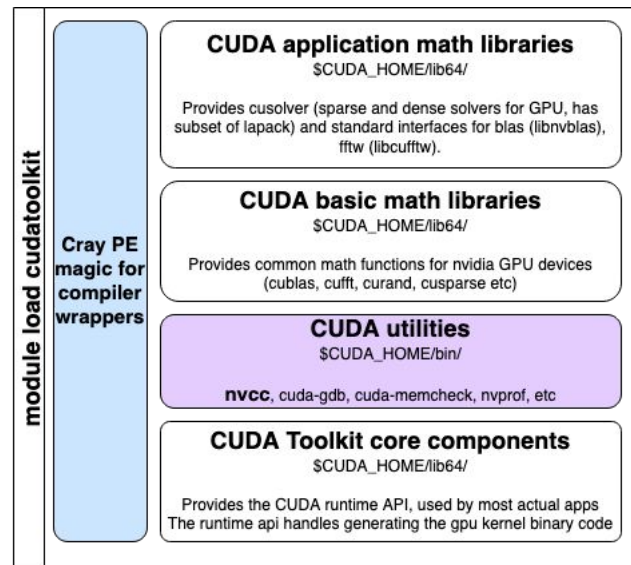
- \*.c, \*.cpp, \*.f90 => CPU source code
  - may include MPI
  - may use directives for GPU
  - **compile with regular compilers (Cray wrappers)**
    - **CC** for C++
    - **cc** for C
    - **ftn** for Fortran
- \*.cu => CUDA kernels
  - **compile with nvcc**
- (Note: With PrgEnv-nvidia, CUDA can be incorporated into same source files as CPU code, add "-cuda" or "-gpu" flag at compile time)

# CUDA and the Perlmutter PrgEnv

**PrgEnv-nvidia** gets you Nvidia compilers plus  
Cray compiler wrappers with MPI and other  
library support  
(loaded by default)



**cuda toolkit** gets you `nvcc` (CUDA) compiler  
plus GPU libraries and tools (Nvidia CUDA  
toolkit) - **needed** for GPU code!  
(you must module load)



# What to load?

For most applications (including today's examples) we recommend the **PrgEnv-nvidia** stack (loaded by default)

To build GPU applications, you will need to load a **cuda toolkit** module

- Choose the CUDA version matching what your application needs

For OpenMP/OpenACC offloading or for CUDA-aware MPI, you also need: **module load craype-accel-nvidia80**

# Try it out!

- Login to Perlmutter
- `git clone https://github.com/NERSC/Perlmutter\_Training\_Jan2022.git`
- Navigate to CUDA/Ex-3/
- **`module load cudatoolkit`**
- Run "make" and look at the output
- (Zoom: everyone raise your hand, and lower it when you have completed this. Jump to breakout room if you have questions about accessing Perlmutter. We'll reconvene when everyone has completed the exercise, or after 10 minutes. For bonus points, try Ex-1 and Ex-2 too)

# What just happened?

You should have seen:

```
CC -gpu=cc80 vecAdd.cu -o vec_add
```

- We are using the C++ Cray compiler wrapper
- All the code is in the CUDA source file
- The Cray wrapper is calling the Nvidia C++ compiler (nvc++) and passing a **-gpu** flag to target A100 GPUs



# More exercises

Work through these at your leisure:

- Ex-1/ is a simple GPU kernel that you can build with only nvcc
- Ex-2/ has C++ and CUDA code in separate files
  - Compile the .cu file with nvcc
  - Compile the .cpp file with (any) C++ compiler
  - Try "module load gcc" to use the GNU compiler instead
- The README.md one level up has info about each

# Running the GPU application

## The Basics:

- Don't run on the login nodes, submit a batch job!
- When submitting a job, you must specify a GPU-enabled account name
  - Same as your "normal" (CPU, Cori) repo name but with `_g` suffix, eg for today:
    - `#SBATCH -A ntrain3_g`

# Necessary SBATCH options (1)

```
#!/bin/bash
#SBATCH -q regular          # "regular" QOS for most jobs
#SBATCH -t 5                # max wallclock time (5 minutes)
#SBATCH -n 8                # number of MPI tasks
#SBATCH -c 32               # reserve 32 cpus per task
#SBATCH --ntasks-per-node=4 # 8 tasks / 4 per node = 2 nodes
#SBATCH --gpus-per-task=1   # 1 GPU per task
#SBATCH -A ntrain3_g         # project/repo
#SBATCH -C gpu
#SBATCH -reservation=perlmutter_day1 # for today only
```

Each phase 1 (GPU) node has 64 cores x 2 hyperthreads, so 128 CPUs => 32 cpus is 1/4th of a node

# Necessary SBATCH options (2)

```
#!/bin/bash
#SBATCH -q regular           # regular QOS
#SBATCH -t 5                 # 5 minutes)
#SBATCH -n 8                 # 8 MPI tasks
#SBATCH -c 32                # reserve 32 cpus per task
#SBATCH --ntasks-per-node=4  # 8 tasks / 4 per node = 2 nodes
#SBATCH --gpus-per-task=1    # reserve 1 (of four) GPUs per task
#SBATCH -A ntrain3_g         # GPU version of your project/repo
#SBATCH -C gpu               # Specify a constraint of "run only on
                             # gpu nodes" and use the _g version of
                             # your repo for GPU hours.
#SBATCH -reservation=perlmutter_day
```

# Running the GPU code

```
#!/bin/bash  
#SBATCH -q regular  
# ... (SBATCH directives as per previous slides)  
srun -n4 ./vec_add
```

## **--gpus-per-task vs -G**

- With **#SBATCH -G** you can specify the total number of GPUs for a job (eg with 2 nodes, you can use "-G 8")
- Handy shorthand for when you are using few or 1 nodes

# Did it work?

If you see errors: make sure you have all the SBATCH directives specified! Eg, when `--gpus-per-task` is not set:

```
sleak@nid001408:~/.../Ex-3> srun -n4 ./vec_add
srun: error: nid001408: tasks 0-1: Floating point exception
srun: launch/slurm: _step_signal: Terminating StepId=944530.1
srun: error: nid001409: tasks 2-3: Floating point exception
```

# Try it out!

- On Perlmutter, in your clone of
- `git clone https://github.com/NERSC/Perlmutter\_Training\_Jan2022.git`
- Navigate to `CUDA/Ex-3/`
- **`make`**
- **`sbatch batch.sh`**
- Bonus points: modify `batch.sh` and run across 2 nodes
- (Zoom: everyone raise your hand, and lower it when you have completed this. Jump to breakout room if you have questions about accessing Perlmutter. We'll reconvene when everyone has completed the exercise, or after 10 minutes. For more bonus points: try Ex-4/ too)

# Affinity and binding (1)

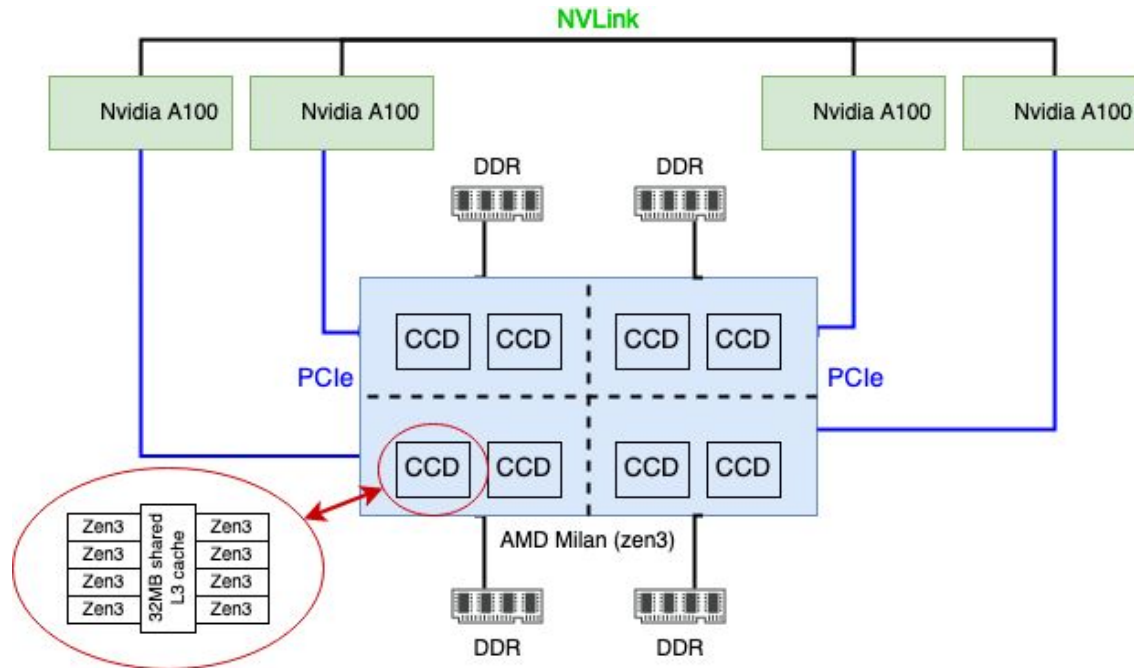
Experienced Cori users will be familiar with the ideas of ***affinity*** and ***binding***:

- Different CPU cores have *affinity* (closeness) to certain memory and caches
- *Binding* a thread or process to certain cores ensures the thread stays on a core close to its data
  - OMP\_PLACES=cores
  - srun --cpu-bind=cores



# Affinity and binding (2)

Perlmutter GPU nodes are configured as "NPS4" => 4 NUMA nodes per socket. Each GPU is "closest" to certain cores



# Affinity and binding (3)

- Set GPU binding with:

```
srunk -n8 --cpu-bind=cores --gpu-bind=closest ./vec_add
```

bind each task to a subset  
of the CPU cores..

.. and to the closest GPU

<https://docs.nersc.gov/jobs/affinity/#gpus>

# Try it out!

- On Perlmutter, in your clone of
- `git clone https://github.com/NERSC/Perlmutter\_Training\_Jan2022.git`
- Navigate to CUDA/Ex-5/
- **make**
- Look at **script\_reg.sh** and **script\_close.sh**.  
Submit each and compare outputs
- Next item on agenda is a break - you can (optionally) continue the exercise into the break, we will reconvene at 11am PT

# Building and running GPU applications on Perlmutter - Part 2



# Recap

We built and ran a C++ application with MPI+CUDA

- Cray compiler wrappers for CPU/MPI code
- nvcc for CUDA code

Two software stacks:

- PrgEnv-nvidia for CPU + MPI
- cudatoolkit for GPU

#SBATCH directives

GPU Affinity

# This session:

"My application isn't that straightforward"

What to do for other common scenarios

# Scenarios

- BLAS/LAPACK/FFTW etc with GPUs
- Other compilers (not NVidia)
- CUDA-aware MPI
- Not CUDA (OpenMP offload, OpenACC)
- cmake
- Spack

# GPU-accelerated math libraries in CUDA

- GPU-accelerated implementations of - or alternatives to - common math libraries are available, or obtainable
- BLAS => cuBLAS (module load cudatoolkit)
- LAPACK => cuSOLVER (module load cudatoolkit)
- Note: cuSOLVER is not the same API as LAPACK
  - BUT: -nvlamath option (NVidia compiler) provides a compatible interface
- FFTW => cuFFT and cuFFTW (FFTW interface to cuFFT)
- cuSPARSE



# Other GPU-accelerated math libraries

- MAGMA  $\leq$  BLAS and a subset of LAPACK
- SLATE  $\leq$  ScaLAPACK

<https://docs.nersc.gov/performance/readiness/#blaslapackscalapack>

# Nvidia HPC SDK Training, Jan 12-13

- A hands-on training provided by Nvidia next week
- **Nvidia HPC SDK compiler**
  - **Default and recommended compiler for Perlmutter GPU**
- Topics include:
  - GPU architecture and HPC SW developer considerations
  - Standard Language Acceleration and Libraries
  - OpenACC
  - OpenMP offload
  - CUDA
  - Profiling tools
- Registration and more info at:  
<https://www.nersc.gov/users/training/events/nvidia-hpcsdk-training-jan2022/>

# Other compilers (non-NVidia)

- **PrgEnv-nvidia is currently the best-supported toolchain for GPU-based applications on Perlmutter**
  - We recommend using PrgEnv-nvidia in most cases, and PrgEnv-nvidia is the default.
- **However** different compilers have different strengths and weaknesses, and for some applications PrgEnv-nvidia will hit difficulties.
  - PrgEnv-gnu is usually a good alternative in these cases

# Compiler Limitations: GNU

- Must choose correct GCC version for a given CUDA version
  - eg cudatoolkit/21.9\_11.4 (default) supports gcc/11.2.0 (default) but previous cudatoolkit versions eg 21.3\_11.2 only support gcc/9.3.0
- OpenMP/OpenACC offloading not yet supported
- CUDA code **must** be in separate files from main source code

# Compiler Limitations: LLVM

## Coming Soon!

- PrgEnv-llvm is a NERSC-supported PrgEnv based on the LLVM compiler
- For C/C++ only (no Fortran)
- Support for SYCL and OpenMP offload
- Not available on Perlmutter yet, but expected soon

# Compiler Limitations: Cray

- PrgEnv-cray offloading does not yet support our A100 GPUs
  - Must `module load craype-accel-host` instead
  - Not tested by NERSC, very limited support

# Compiler Limitations: AOCC

- No offloading support yet
- No testing by NERSC yet, very limited support

# Recommendation

**If PrgEnv-nvidia is not viable for your application, use PrgEnv-gnu**

Support for other toolchains will improve, but for now only PrgEnv-nvidia and PrgEnv-gnu are supported



# Errors you might see

- `srun: error: nid001408: tasks 0-1: Floating point exception => did you specify --gpus-per-task (or -G) in your sbatch allocation?`
- `slurmstepd: error: Bind request 3 (0x8) does not specify any devices within the allocation. Binding to the first device in the allocation instead. => Did you request all of the GPUs on the node? (eg -N2 -G4 gets you only 2 GPUs per node)`
- `vec_add: error while loading shared libraries: libnvcpumath.so: cannot open shared object file: No such file or directory => Did you "make clean" after swapping PrgEnvs?`

# Try it out!

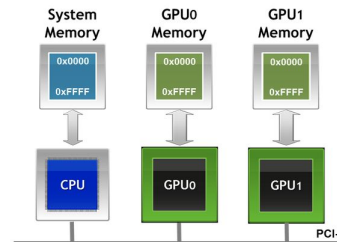
- On Perlmutter, in your clone of
- `git clone https://github.com/NERSC/Perlmutter\_Training\_Jan2022.git`
- Repeat Ex-4, Ex-5 with PrgEnv-gnu
  - Note: Ex-3 has CUDA and C++ code in same source file, only supported by PrgEnv-nvidia
- (Zoom: everyone raise your hand, and lower it when you have completed this. Jump to breakout room if you have questions about accessing Perlmutter. We'll reconvene when everyone has completed the exercise, or after 10 minutes. For more bonus points: try Ex-4/ too)

# CUDA-aware MPI (1)

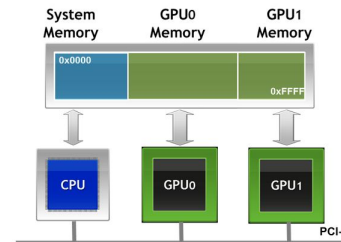
NVIDIA UVA presents GPU device memory as part of the same address space as CPU main memory

- Allows a CUDA-aware MPI implementation (eg Cray-MPICH) to send and receive messages directly from/to GPU memory - no copy-to-main-memory needed

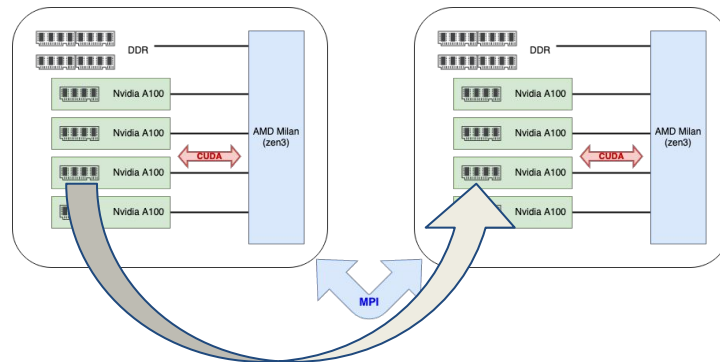
*No UVA: Multiple Memory Spaces*



*UVA: Single Address Space*



(from <https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/>)



## CUDA-aware MPI (2)

If your executable uses CUDA-aware MPI, `ldd` should show `libmpi_gtl_cuda.so.0`, eg:

```
libmpi_gtl_cuda.so.0 =>  
/opt/cray/pe/lib64/libmpi_gtl_cuda.so.0
```

At runtime (in batch script) set:

```
export MPICH_GPU_SUPPORT_ENABLED=1
```

# Try it out!

- On Perlmutter, in your clone of
- `git clone https://github.com/NERSC/Perlmutter\_Training\_Jan2022.git`
- Look at the example in CUDA-aware-MPI. Build and run it, and use `ldd` to verify the presence of `libmpi_gtl_cuda`
  - Note: Remember to switch back to `PrgEnv-nvidia`!
- (Zoom: everyone raise your hand, and lower it when you have completed this. Jump to breakout room if you have questions about accessing Perlmutter. We'll reconvene when everyone has completed the exercise, or after 10 minutes. This example comes from <https://docs.nersc.gov/development/programming-models/mpi/#cuda-aware-mpi>)

# Try it out

```
rgayatri@perlmutter:login34:/pscratch/sd/r/rgayatri/Perlmutter_Training_Jan2022/CUDA/CUDA-aware-MPI> ldd bcast_from_device
```

```
[snip]
```

```
libz.so.1 => /lib64/libz.so.1 (0x00007f7621b2e000)
```

```
libdl.so.2 => /lib64/libdl.so.2 (0x00007f762192a000)
```

```
libmpi_nvidia.so.12 => /opt/cray/pe/lib64/libmpi_nvidia.so.12 (0x00007f761f1cc000)
```

```
libmpi_gtl_cuda.so.0 => /opt/cray/pe/lib64/libmpi_gtl_cuda.so.0  
(0x00007f761efbb000)
```

```
libxpmem.so.0 => /opt/cray/xpmem/default/lib64/libxpmem.so.0 (0x00007f761edb8000)
```

```
libcudanhpc.so => /opt/nvidia/hpc_sdk/Linux_x86_64/21.9/compiler/lib/libcudanhpc.so  
(0x00007f761ebb4000)
```

```
linux-vdso.so.1 (0x00007ffd375ab000)
```

# OpenMP Offload

```
#pragma omp target teams distribute parallel for \  
map(to: a[:n], b[:n]) map(from: c[:n])
```

CXXFLAGS += -mp=gpu -gpu=cc80 -Minfo

(-Minfo is optional, prints useful info during compile)

# OpenACC

```
#pragma acc parallel loop gang vector \  
    copyin(a[:n]) copyout(c[:n])
```

```
CXXFLAGS += -acc -Minfo=accel
```



# Try it out!

- On Perlmutter, in your clone of
- `git clone https://github.com/NERSC/Perlmutter\_Training\_Jan2022.git`
- Look at the example in OpenMP-OpenACC
- Build and run it
  - Note: Remember to switch back to PrgEnv-nvidia!
- (Zoom: everyone raise your hand, and lower it when you have completed this. Jump to breakout room if you have questions about accessing Perlmutter. We'll reconvene when everyone has completed the exercise, or after 10 minutes)

# Using cmake on Perlmutter

- cmake modules available on perlmutter
  - Latest cmake version
- Current issues with linking math libraries (cufft and cusolver)
  - `export`  
`CMAKE_PREFIX_PATH=/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/math_libs/11.4:$CMAKE_PREFIX_PATH`
- Known issues
  - <https://docs.nersc.gov/current/#new-issues>

# Spack on Perlmutter

Spack 0.17.0 will be available for Perlmutter soon

Configured to work with NERSC E4S deployment