

Programming and Building HPC Applications for Running on One Nvidia GPU

(Part 2 of the Cabeus Training)

Mar. 13, 2024

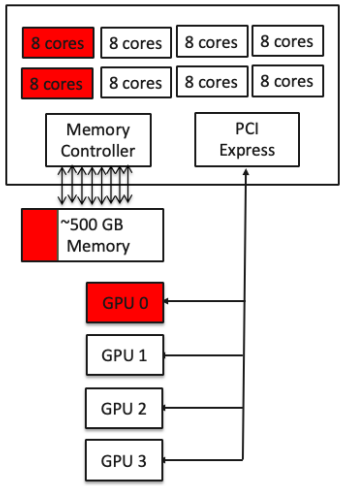
NASA Advanced Supercomputing (NAS) Division

Topics

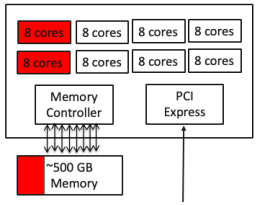
- Part 1: Overview on the New Cabeus Cluster
  - Cabeus Hardware Resources
  - PBS Jobs Sharing GPU Nodes
  - SBU Charging

- Part 2: Programming and Building HPC Applications for Running on One Nvidia GPU
  - Programming
    - Methods Recommended by Nvidia
    - CPU Offloading to GPU
    - Explicit and Implicit Data Movement
  - Building
    - Compute Capability
    - CUDA Toolkit and Driver

A CPU Processor vs A Nvidia GPU Card



CPU (Central Processing Unit) Processor (example: Milan 7763)

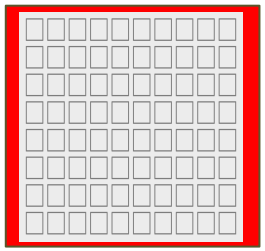


- A CPU, like a CEO, where the OS runs, is always needed in a node
- Larger size but lower bandwidth memory
  - ~500 GB DDR4 memory per socket
  - ~200 GB/s memory bandwidth
- Fewer but more powerful generalized cores
  - Higher clock speed, e.g. 2.45 GHz
  - 64 cores
  - Can perform/switch between multiple tasks (out-of-order, speculative) quickly
- Less parallelism

Nvidia GPU (graphics processing unit) (example: Nvidia A100)

GPU

Is more like

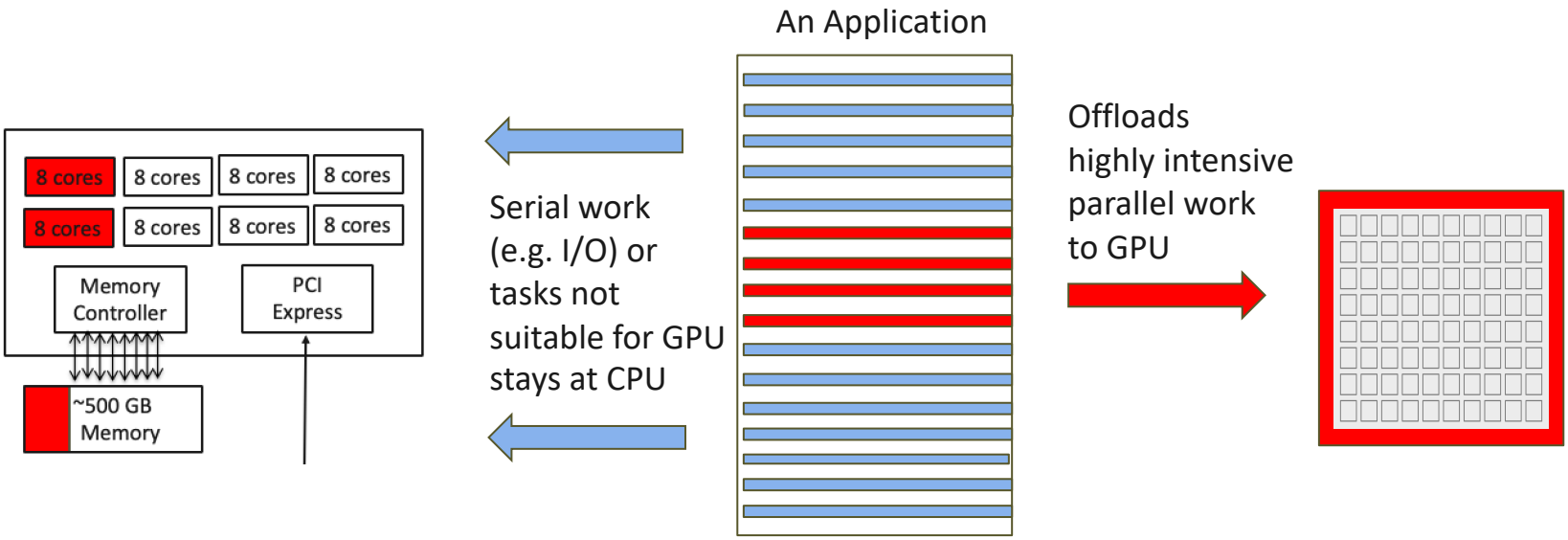


(still over-simplified)

- A GPU, with many “specialized engineers”, functions as a coprocessor in a node
- Smaller size but higher bandwidth memory
  - 80 GB HBM2e memory per GPU card
  - ~2000 GB/s memory bandwidth
- Many less powerful but specialized cores per GPU card
  - Lower clock speed, e.g., base clock 1.275 GHz
  - 3,456 double precision CUDA cores (CUDA = Compute Unified Device Architecture); A CUDA core handles general purpose computing similar to a CPU core, however, it mostly does number crunching and much less on out-of-order or speculative operation; with so many of them, they can process a lot more data for the same task in parallel
  - 432 Tensor cores; Tensor core specialized for matrix multiply and mixed precision computing required for AI; note: availability of Tensor cores started with V100
  - Perform simple and repetitive tasks much faster
- High parallelism

CPU Offloads Parallel Work to GPU

A typical application offloads compute-intensive parallelized work, mostly in the form of parallel loops, from CPU to GPU (aka the CUDA code) and keeps the serial work or tasks not suitable for the GPU on the CPU



Programming Methods Recommended by Nvidia



- Language extension CUDA model (when aiming for best possible performance)
  - CUDA C/C++: The CPU host launches a kernel on the GPU with the triple angle bracket syntax <<<. .. >>>. E.g.

    CPU code CPU launches GPU CUDA code

    add(N, x, y); add<<<nb, nt>>>(N, x, y); where nb, nt: # of blocks, # of threads/block running in parallel
  - CUDA Fortran: call saxpy<<<nb,nt>>>(x,y,a) where saxpy is a single precision function for y = a* x + y
- Compiler directive OpenACC model (easier for porting an existing code)

  To offload to GPU: #pragma acc kernels (C/C++); !$acc kernels (Fortran); need Nvidia compiler flag **–acc=gpu**

  or #pragma acc parallel (C/C++); !$acc parallel (Fortran); need Nvidia compiler flag **–acc=gpu**
- Compiler directive OpenMP model (not as well supported as OpenACC by Nvidia)

  To offload to GPU: #pragma omp target (C/C++); !$omp target (Fortran); need Nvidia compiler flag **–mp=gpu**
- Standard language parallel programming model (standard code can run on GPUs or multi-core CPU that supports this)
  - ISO C++17 standard library parallel algorithms and ISO Fortran 2018 do concurrent loop construct automatically offload to GPU (Volta and newer) with use of Nvidia compiler flag **–stdpar=gpu**; **relies on CUDA Unified Memory**
  - For running on many cores CPU platforms, use –stdpar=multicore

Sample C++ code with a parallel sort algorithm

```
std:sort (std:execution::par,
          employees.begin(), employees.end(),
          CompareByLastName() );
```

Sample Fortran do concurrent Loop construct

```
DO CONCURRENT (i=1:n)
  y(i) = y(i) + a* x(i)
END DO
```

References: http://www.nas.nasa.gov/hecc/support/kb/entry/647 and many references therein

Past HECC webinar: https://www.nas.nasa.gov/hecc/assets/pdf/training/OpenACC_OpenMP_04-25-18.pdf

Nvidia Compilers and Math Libraries

https://developer.nvidia.com/hpc-sdk

Available after loading one of these modules: nvhpc/23.11, nvhpc/23.7, nvhpc/23.5, e.g.,

a GPU node, cfe, pfe % module load nvhpc/23.11

your $PATH and $LD_LIBRARY_PATH are modified to include locations of the compilers, libraries

- Compilers

  Under /nasa/nvidia/hpc_sdk/toss4/Linux_x86_64/23.xx/compilers (where xx = 11, for example)

  - nvcc: Nvidia CUDA C and C++ compiler driver (for programs written in CUDA C/C++)

    nvcc splits code into host code (to be compiled by a host compiler such as gcc, g++) and device code (by nvcc)

    https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html

  - Nvidia HPC Compilers, rebranded from PGI compiler, for Nvidia GPUs (and also Intel, AMD, OpenPower, Arm CPUs)
    - nvc (was pgcc, C11 compiler) : supports OpenACC, OpenMP
    - nvc++ (was pgc++) : supports OpenACC, OpenMP, C++17 language standard
    - nvfortran (was pgfortran) : supports OpenACC, OpenMP, CUDA Fortran, ISO Fortran 2018 'DO CONCURRENT'

    To learn more options: man nvfortran, nvfortran –h, nvfortran –gpu –h (replace nvfortran with nvc or nvc++)

- Math libraries: GPU-accelerated cuBLAS, cuSPARSE, cuFFT, cuTENSOR, cuSOLVER, cuRAND

  Under /nasa/nvidia/hpc_sdk/toss4/Linux_x86_64/23.xx/math_libs/12.3/targets/x86_64-linux/lib

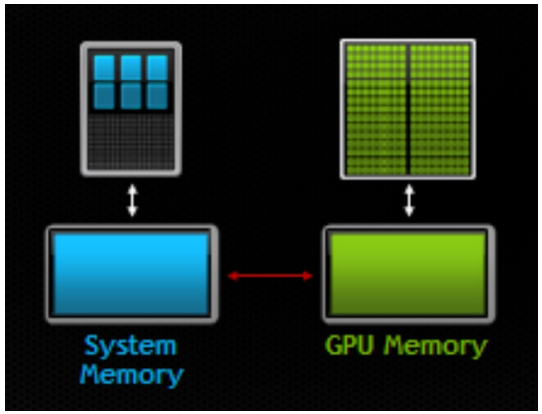
  - Callable from CUDA, OpenACC, and OpenMP programs written in Fortran, C, C++
  - Some examples available at /nasa/nvidia/hpc_sdk/toss4/Linux_x86_64/23.xx/examples/CUDA-Libraries

Note: Under /nasa/nvidia/…../examples directory, there are also CUDA-Fortran, OpenACC, OpenMP, stdpar examples

Explicit Data Movement

- The CPU and GPU have separate physical memory.

Image from: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/



- Traditionally, data shared between CPU and GPU need to be **explicitly** transferred. This is **difficult** for program development.
- Typical workflow:
  1. Declare and allocate CPU host memory (e.g. malloc) and GPU device memory (e.g. cudaMalloc)
  2. Initialize host data
  3. Copy data from host to device (e.g. cudaMemcpy; #pragma acc data copyin)
  4. Run the kernels on the GPUs
  5. Copy results from device to host (e.g., cudaMemcpy; #pragma acc data copyout)
  6. Free allocated host (e.g., free) and device memory (e.g., cudaFree)

Sample CUDA Fortran Code with Explicit Data Movement



- Device Code in test.f90

```
module sample_device_add
contains
  attributes(global) subroutine add(a, b)
    implicit none
    integer, device :: a(:), b(:)
    integer :: i

    i = threadIdx%x
    a(i) = a(i) + b(i)
  end subroutine add
 end module sample_device_add
```

```
% module load nvhpc/23.11
% nvfortran –cuda test.f90
```

- Host Code in test.f90

```
Program sample_host_add
use cudafor
use sample_device_add
implicit none
integer :: n = 256
integer, allocatable :: a(:), b(:)
integer, allocatable, device :: a_d(:), b_d(:)

allocate( a(n), b(n), a_d(n), b_d(n) )
a = 1
b = 3
a_d = a
b_d = b
call add<<<1,n>>>(a_d, b_d)
a = a_d
write (6,*) 'a = ', a
deallocate(a, b, a_d, b_d)
end program sample_host_add
```

cudafor module contains CUDA Fortran definitions

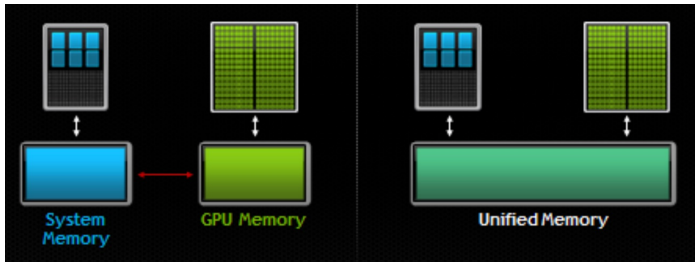
#User-defined module shown at left

1. Allocate
2. Initiate
3. Copy host -> device
4. Run kernels
5. Copy device -> host
6. Deallocate

Implicit Data Movement

- Unified Memory (logical view of memory; not physical)

Image from: https://developer.nvidia.com/blog/unified-memory-in-cuda-6/



- Development in CUDA device driver and Nvidia GPU hardware enabled unified memory to allow **implicit** (fine-grain, on-demand, automated) data movement (handled by Nvidia CUDA driver) and **ease** of programming; but **maybe slower**
  - CUDA >= 6, Kepler GPUs and later: a new API cudaMallocManaged() for creating a pool of managed memory shared between CPUs and GPUs
  - CUDA >=8, Pascal GPUs and later: more features, e.g., page fault, on-demand migration, memory oversubscription (i.e., use more memory than the size of GPU memory), concurrent access by both CPU and GPU, etc.
  - Depending on data access pattern, explicit bulk data transfer may perform better than using unified memory

https://developer.nvidia.com/blog/unified-memory-in-cuda-6/

https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda

- Nvidia compiler/linker support for Unified/Managed Memory when using directive based or standard language methods
  - ❖ -gpu=managed (only dynamic allocated data, i.e., heap, is allocated in CUDA managed memory); -stdpar implies -gpu=managed when compiled on systems with CUDA Managed Memory capability only, such as Volta, Ampere
  - ❖ -gpu=unified (starting nvhpc/23.11) all host data, i.e., heap, stack, global, are placed in a unified single address space); -stdpar implies –gpu=unified on systems with full CUDA Unified Memory capability, such as Grace Hopper

Sample Fortran 2018 DO CONCURRENT Code



- This code looks much cleaner and easier to write
- Standard language code can be compiled and run on GPU or Multicore CPU

```
Program sample_fortran2018_add
implicit none
integer :: i,  n = 256
integer, allocatable :: a(:), b(:)

allocate(a(n), b(n))
a = 1
b = 3
DO CONCURRENT (i = 1: n)
a(i) = a(i) + b(i)
END DO

write (6,*) 'a = ', a

deallocate(a, b)
end program sample_fortran2008_add
```

```
% module load nvhpc/23.11

# For running on GPU
% nvfortran –stdpar=gpu test.f90

# For running on multicore CPU
% nvfortran –stdpar=multicore test.f90
```

Compute Capability of Different GPUs

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities

- The compute capability of a device is represented by a number which identifies the features supported by the GPU hardware and is used by applications at runtime to determine which hardware features and/or instructions are available on the present GPU. Examples of features or specifications: has tensor cores? maximum # of blocks, threads, cache, …
- Compute Capability of V100 and A100

  % module load nvhpc/23.11; nvaccelinfo | grep Target
  - V100: cc70 (compute capability 7.0)
  - A100: cc80 (compute capability 8.0)

  Note: For a Nvidia Hopper GPU, it will be cc90
- Compilation examples with or without –gpu=ccXY to target certain compute capability X.Y

  Note: You can build a GPU application on a GPU compute node, cfe[01,02] or pfe
  - Compile a C++ GPU code with the standard language parallel programming model, target Milan CPU and A100 GPU

    nvc++ -stdpar=gpu –gpu=cc80 -tp=znver3 program.cpp
  - Compile an OpenACC Fortran code to offload OpenACC region to GPUs and generate optimization messages

    nvfortran -acc=gpu -fast -Minfo=all program.f90

    When –gpu is not included, on A100 nodes, it defaults to cc80; on V100 nodes, defaults to cc70, on cfe or pfe, defaults to –gpu=cc35 –gpu=cc50 –gpu=cc60 –gpu=cc61 –gpu=cc70 –gpu=cc75 –gpu=cc80 –gpu=cc86 –gpu=cc89 –gpu=cc90

    Tip: add –dryrun to see what the compiler does, such as what ccXY values are included
  - Compile a CUDA Fortran code

    nvfortran -cuda program.xx (if xx = cuf or CUF, -cuda can be omitted)
- Compatibility
  - An executable built with –gpu=cc70 can run on both V100 and A100
  - An executable built with just –gpu=cc80 can run on A100, but NOT on V100
  - An executable built with –gpu=cc70 –gpu=cc80 can run on either V100 or A100

CUDA Toolkit and CUDA Driver

- CUDA Toolkit
  - Includes libraries (such as cudart CUDA runtime, cuBLAS, cuFFT, etc) and tools (such as cuda-gdb, nvprof) for building, debugging and profiling applications (see https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html)
  - Can have multiple versions installed on a non-system path, e.g., /nasa has these Nvidia HPC Software Development Kits
    - nvhpc/23.11 : 12.3, 11.8 ; nvhpc/23.7 : 12.2 ; nvhpc/23.5 : 12.1, 11.8, 11.0
    - With each nvhpc/23.xx, during compilation, the CUDA Toolkit version chosen depends on whether the host has a CUDA driver; if yes, use a CUDA Toolkit version with closest match with the CUDA driver; if no, then definition of DEFCUDAVERSION in /nasa/nvidia/hpc_sdk/…/23.xx/compilers/bin/localrc is used
    - Recommendation: add –dryrun during compilation to check which version of the CUDA Toolkit is used
  - With nvc, nvc++ and nvfortran, can explicitly choose a version using –gpu=cudaX.Y (e.g., 12.3) if version X.Y is available
- CUDA Driver
  - includes user-mode driver (libcuda.so) and kernel-mode driver (nvidia.ko) for running applications
  - Only one version can be installed by a sysadm (/usr/lib/modules/…../nvidia); current version (nvidia-smi or nvaccelinfo)
    - mil_a100 : 535.104.12 (installed in late 2023)
    - Older NAS GPU nodes : 535.104.12 (updated from 530.30.02 in Feb. 2024)
- Compatibility

  https://docs.nvidia.com/deploy/cuda-compatibility and CUDA_Toolkit_Release_Notes.pdf
  - Each CUDA release has a Toolkit version and a corresponding driver version. E.g., Toolkit 12.3 update 2, driver 545.23.08
  - CUDA driver is backward compatible – applications built with older CUDA Toolkit will work on newer driver releases
  - Applications built with a new CUDA Toolkit require a minimum older CUDA driver release
    - Applications built with CUDA Toolkit 11.x require CUDA driver >= 450.80.02
    - Applications built with CUDA Toolkit 12.x require CUDA driver >= 525.60.13

  If you bring an application built elsewhere (such as latest development version of a container from Nvidia), check which version of CUDA Toolkit was used to build it and whether the CUDA driver on NAS GPUs is compatible with it.

Other Topics Not Covered

- Multi-GPU/Multi-Node
  - HPE MPT library

    mpi-hpe/mpt.2.28_25Apr23_rhel87
  - OpenMPI
    - Various nvhpc releases provide more than 1 version of OpenMPI

      For example, nvhpc/23.11 has 3.1.5 and 4.1.5
    - Version built by NAS: module use /nasa/modulefiles/testing

      module load openmpi/4.1.6-toss4-gnu
  - NCCL (Nvidia Collective Communications Library)

    For example, /nasa/nvidia/hpc_sdk/toss4/Linux_x86_64/23.11/comm_libs/12.3/nccl/lib

    https://developer.nvidia.com/nccl
- AI/ML
  - Machine learning At NAS:

    https://www.nas.nasa.gov/hecc/support/kb/machine-learning-at-nas-183
- Profiling

  https://developer.nvidia.com/performance-analysis-tools

Annual NASA GPU Hackathon



Consider joining this hackathon to learn more faster

http://www.nas.nasa.gov/hackathon

Sept 10, 17-19, 2024

- Optimizing existing GPU applications
- Converting an existing CPU-only application to run on GPUs
- Writing a new application from scratch for the GPUs (??)

Questions?

Recording and slides of Part 1 and Part 2 will be available in a few days at
http://nas.nasa.gov/hecc/support/past_webinars.html