RUNNING PARALLEL PROGRAMS – MPI, OpenMP, CUDA

Table of Contents

1. OVERVIEW

This section of the user-guide covers the steps to run sample serial and parallel code in C, C++, Fortran, Python, and R using both interactive and batch job submission modes. Where relevant, separate steps for using Intel and GNU compilers are covered.

Please note that **all code should be run only on the compute nodes** either interactively or in batch mode. Note that when a batch job is submitted on the login node, the batch job script specifies that the commands are to be run on the compute nodes. **Please DO NOT run any code on the login nodes**. Acceptable use of login nodes is as follows: installing code, file transfer, file editing, and job submission and monitoring.

Note: The examples provided in this document include the loading of specific Linux environment modules. These modules adjust your shell configuration, such as the $PATH environment variable, to accommodate particular software packages. Further details and information regarding the management of Linux modules, such as loading, unloading, swapping, and other operations, can be found here: [ModuleEnvironments < ARC < UTSA Research Support Group](#).

2. CLONE THE GITHUB REPOSITORY

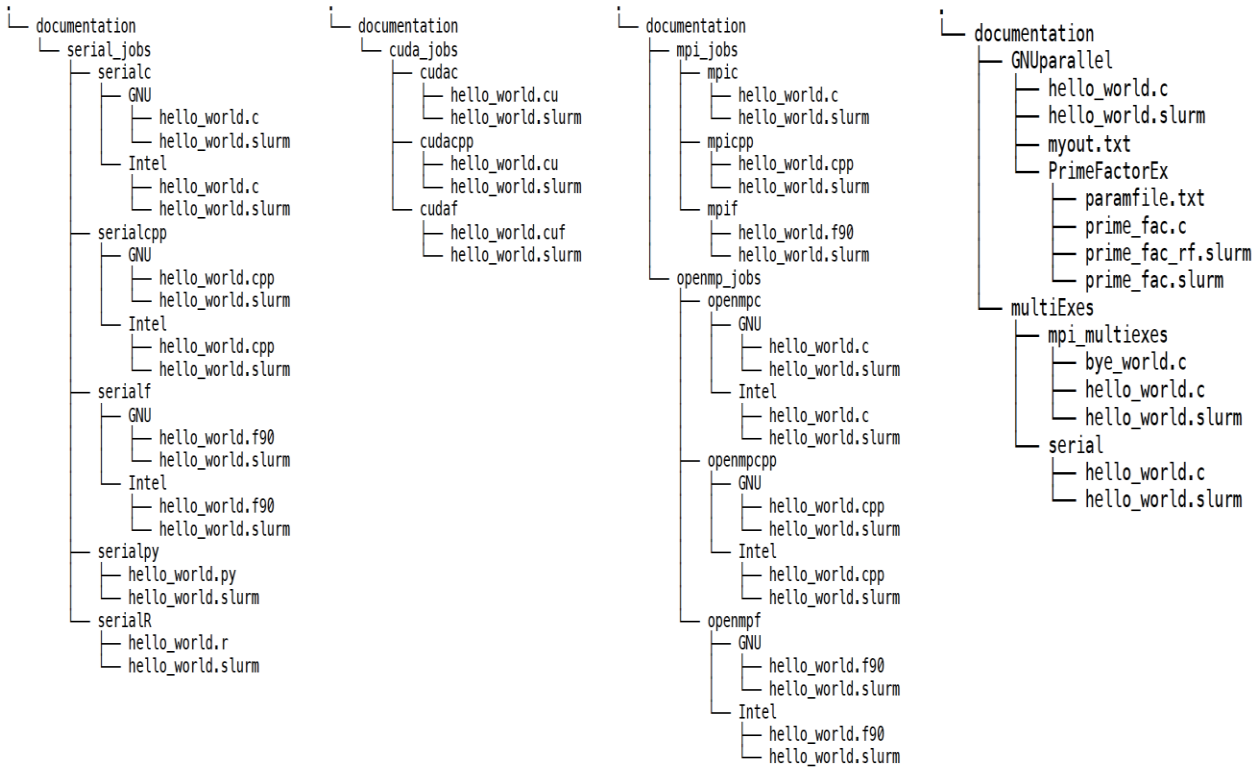
If you want to get a copy of all the programs and scripts used in the examples shown in this document, you can clone the GitHub repository for the examples using the following command:

```
git clone https://github.com/ritua2/documentation
```

If you cloned the GitHub repository, you can switch to the documentation directory with the following command to find the scripts and programs referred to throughout the document:

```
[login001]$ cd documentation
```

The **documentation** directory structure is shown below (split across four columns):



When you switch to the subdirectories within the **`documentation`** folder, the Slurm job script files are available with the *.slurm extension.

If you do not want to clone the aforementioned GitHub repository, you should be able to copy the code shown in the listings into files with appropriate names and file extensions.

3. COMPILING AND RUNNING A SAMPLE C+MPI CODE IN PARALLEL MODE

A sample C + MPI program is shown in Listing 11. This program will print "`Hello world from processor #, rank # out of # processors`" to standard output. The "#" signs in the aforementioned quoted text will be replaced with the processor name, rank, and total number of MPI processes participating in the computation.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
     // Initialize the MPI environment
     MPI_Init(NULL, NULL);
     // Get the number of processes
     int world_size;
     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
     // Get the rank of the process
     int world_rank;
     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
     // Get the name of the processor
     char processor_name[MPI_MAX_PROCESSOR_NAME];
     int name_len;
     MPI_Get_processor_name(processor_name, &name_len);
     // Print off a hello world message
     printf("Hello world from processor %s, rank %d out of %d
     processors\n",
     processor_name, world_rank, world_size);
     // Finalize the MPI environment.
     MPI_Finalize();
     return 0;}
```

Listing 11: Sample C+MPI code – hello_world.c (documentation/mpi_jobs/mpic/hello_world.c)

If you would like to compile the C + MPI example using the MVAPICH2 MPI library, you can run the following commands:

```
[login001]$ ml mvapich2
[login001]$ mpicc -o hello_world hello_world.c
```

The executable **`hello_world`** can be run either in a batch mode using a Slurm batch job-script or interactively on a compute node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a compute node using the following set of commands and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 12 -N 2 -t 00:05:00 --pty bash
[c001]$ mpirun -np 12 ./hello_world
Hello world from processor c002, rank 10 out of 12 processors
Hello world from processor c002, rank 8 out of 12 processors
Hello world from processor c002, rank 6 out of 12 processors
Hello world from processor c002, rank 9 out of 12 processors
Hello world from processor c002, rank 11 out of 12 processors
Hello world from processor c002, rank 7 out of 12 processors
Hello world from processor c001, rank 4 out of 12 processors
Hello world from processor c001, rank 1 out of 12 processors
Hello world from processor c001, rank 3 out of 12 processors
Hello world from processor c001, rank 5 out of 12 processors
Hello world from processor c001, rank 2 out of 12 processors
Hello world from processor c001, rank 0 out of 12 processors
```

If you are currently on a compute node and would like to switch back to the login node then enter the exit command as follows:

```
[c001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 12. This batch script corresponds to the program named `hello_world.c`. This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_mpic
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:10:00
#SBATCH -N 2
#SBATCH -n 12

mpirun -np 12 ./hello_world #the executable name is obtained
after compiling the program
```

Listing 12: Batch Job Script for C+MPI code – hello_world.slurm
(documentation/mpi_jobs/mpic/hello_world.slurm)

The job-script shown in Listing 12 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 12 can be checked by opening the output file as follows:

```
[login001]$ cat  hello_world.txt
Hello world from processor c002, rank 10 out of 12 processors
Hello world from processor c002, rank 8 out of 12 processors
Hello world from processor c002, rank 6 out of 12 processors
Hello world from processor c002, rank 9 out of 12 processors
Hello world from processor c002, rank 11 out of 12 processors
Hello world from processor c002, rank 7 out of 12 processors
Hello world from processor c001, rank 4 out of 12 processors
Hello world from processor c001, rank 1 out of 12 processors
Hello world from processor c001, rank 3 out of 12 processors
Hello world from processor c001, rank 5 out of 12 processors
Hello world from processor c001, rank 2 out of 12 processors
Hello world from processor c001, rank 0 out of 12 processors
```

4. COMPILING AND RUNNING SAMPLE MPI PROGRAM IN C++

A sample C++ MPI program is shown in Listing 13. This program will print "`Hello world from processor #, rank # out of # processors`" to standard output. The "#" signs in the aforementioned quoted text will be replaced with the processor name, rank, and total number of MPI processes participating in the computation.

If you would like to compile the C++ example using the MVAPICH2 MPI library, you can run the following commands:

```
[login001]$ ml mvapich2
[login001]$ mpicxx -o hello_world hello_world.cpp
```

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a compute node.

```
#include <mpi.h>
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
     // Initialize the MPI environment
     MPI_Init(NULL, NULL);
     // Get the number of processes
     int world_size;
     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
     // Get the rank of the process
     int world_rank;
     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
     // Get the name of the processor
     char processor_name[MPI_MAX_PROCESSOR_NAME];
     int name_len;
     MPI_Get_processor_name(processor_name, &name_len);
     // Print off a hello world message
     cout<<"Hello World from processor "<<processor_name<<",
rank "<<world_rank<<"out of "<<world_size<<"processors\n";
    // Finalize the MPI environment.
     MPI_Finalize();
     return 0;}
```

Listing 13: Sample MPI program with C++ - hello_world.cpp (documentation/mpi_jobs/mpicpp/hello_world.cpp)

Running the Executable in Interactive-Mode: The executable can be run interactively on a compute node using the following set of commands and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 12 -N 2 -t 00:05:00 --pty bash
[c001]$ mpirun -np 12 ./hello_world
Hello World from processor c001, rank 0out of 12processors
Hello World from processor c001, rank 1out of 12processors
Hello World from processor c001, rank 2out of 12processors
Hello World from processor c001, rank 5out of 12processors
Hello World from processor c001, rank 3out of 12processors
Hello World from processor c001, rank 4out of 12processors
Hello World from processor c002, rank 6out of 12processors
Hello World from processor c002, rank Hello World from processor
c002, rank 8out of 12processors
7out of 12processors
Hello World from processor c002, rank 11out of 12processors
Hello World from processor c002, rank 10out of 12processors
Hello World from processor c002, rank 9out of 12processors
```

Note: It is common to see the output printed in a non-deterministic manner - in the example above the process rank 7 and rank 8 overlap each other in the writing step.

If you are currently on a compute node and would like to switch back to the login node then enter the `exit` command as follows:

```
[c001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 14. This batch script corresponds to the program named `hello_world.cpp`. This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_mpicpp
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00                # Time limit hrs:min:sec. It
is an estimation about how long it will take to complete the
job. 72:00:00 is the maximum
#SBATCH -N 2
#SBATCH -n 12


mpirun -np 12 ./hello_world #the executable name is obtained
after compiling the program
```

Listing 14: Batch Job Script for MPI with C++ code - hello_world.slurm (documentation/mpi_jobs/mpicpp/hello_worldslurm)

The job-script shown in Listing 14 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 14 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
Hello World from processor c001, rank 0out of 12processors
Hello World from processor c001, rank 1out of 12processors
Hello World from processor c001, rank 2out of 12processors
Hello World from processor c001, rank 5out of 12processors
Hello World from processor c001, rank 3out of 12processors
Hello World from processor c001, rank 4out of 12processors
Hello World from processor c002, rank 6out of 12processors
Hello World from processor c002, rank Hello World from processor
c002, rank 8out of 12processors
7out of 12processors
Hello World from processor c002, rank 11out of 12processors
Hello World from processor c002, rank 10out of 12processors
Hello World from processor c002, rank 9out of 12processors
```

Note: It is common to see the output printed in a non-deterministic manner - in the example above the process rank 7 and rank 8 overlap each other in the writing step.

5. COMPILING AND RUNNING A SAMPLE FORTRAN+MPI PROGRAM

A sample Fortran+MPI program is shown in Listing 15. This program will print "`Hello world`" to the output file as many times as there are MPI processes.

```
program hello
include 'mpif.h'
call MPI_INIT (ierr)
print *, "Hello world"
call MPI_FINALIZE (ierr)
end program hello
```

Listing 15: Sample Fortran+MPI code - hello_world.f90 (documentation/mpi_jobs/mpif/hello_world.f90)

If you would like to compile the Fortran example using the Intel OneAPI library, you can run the following commands:

```
[login001]$ ml intel/oneapi/2021.2.0
[login001]$ mpiifort -o hello_world hello_world.f90
```

The executable **`hello_world`** can be run either in a batch mode using a Slurm batch job-script or interactively on a compute node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a compute node using the following set of commands and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 12 -N 2 -t 00:05:00 --pty bash
[c001]$ mpirun -np 12 ./hello_world
 Hello world
 Hello world
 Hello world
 Hello world
 Hello world
 Hello world
 Hello world
```

```
 Hello world
 Hello world
 Hello world
 Hello world
 Hello world
```

If you are currently on a compute node and would like to switch back to the login node then enter the `exit` command as follows:

```
[c001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 16. This batch scirpt corresponds to the program named `hello_world.f90`. This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_mpif
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH --time=00:05:00
#SBATCH --nodes=2
#SBATCH --ntasks=12

mpirun -np 12 ./hello_world #the executable name is obtained
after compiling the program
```

Listing 16: Batch Job Script for Fortran+ MPI - hello_world.slurm (documentation/mpi_jobs/mpif/hello_world.slurm)

The job-script shown in Listing 16 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 16 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
 Hello world
 Hello world
 Hello world
```

```
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
```

6. COMPILING AND RUNNING A SAMPLE C+OPENMP PROGRAM

A sample C+OpenMP program is shown in Listing 17. This code will print "`Hello World... from thread =`" # where # is the thread number to standard output.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[]){
     // Beginning of parallel region
     #pragma omp parallel
     {
     printf("Hello World... from thread = %d\n",
     omp_get_thread_num());
     }
     // Ending of parallel region
     return 0;}
```

Listing 17: Sample C+OpenMP program – hello_world.c (documentation/openmp_jobs/openmpc/GNU/hello_world.c)

If you would like to compile the C example using the GNU C compiler, you can run the following commands:

```
[login001]$ ml gnu8/8.3.0
[login001]$ gcc -fopenmp -o hello_world hello_world.c
```

If you would like to compile the C example using the Intel OneAPI, you can run the following commands:

```
[login001]$ ml intel/oneapi/2021.2.0
[login001]$ icc -qopenmp -o hello_world hello_world.c
```

Note: Some compiler options are the same for both Intel and GNU (e.g. "-o"), while others are different (e.g. "-qopenmp" vs "-fopenmp").

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a compute node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a compute node using the following set of commands and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 4 -t 00:05:00 --pty bash
[c001]$ export OMP_NUM_THREADS=4
[c001]$ ./hello_world
Hello World... from thread = 2
Hello World... from thread = 0
Hello World... from thread = 1
Hello World... from thread = 3
```

If you are currently on a compute node and would like to switch back to the login node then enter the `exit` command as follows:

```
[c001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 18. This batch script corresponds to the parallel program named `hello_world.c`. This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_openmpc
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 4
```

```
export OMP_NUM_THREADS=4
./hello_world #the executable name is obtained after compiling
the program
```

Listing 18: Batch Job Script for C+OpenMP - hello_world.slurm (documentation/openmp_jobs/openmpc/GNU/hello_world.slurm)

The job-script shown in Listing 18 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 18 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
Hello World... from thread = 2
Hello World... from thread = 0
Hello World... from thread = 1
Hello World... from thread = 3
```

7. COMPILING AND RUNNING A SAMPLE OPENMP PROGRAM IN C++

A sample C++ program with OpenMP is shown in Listing 19. This code will print "`Hello World... from thread =`" # where # is the thread number to standard output.

```
#include <omp.h>
#include <iostream>
#include <stdlib.h>
using namespace std;
int main(int argc, char* argv[]){
     // Beginning of parallel region
     #pragma omp parallel
     {
          cout<<"Hello World... from thread ="<<
          omp_get_thread_num();
          cout<<"\n";}
     // Ending of parallel region
     return 0;}
```

Listing 19: Sample OpenMP program in C++ - hello_world.cpp (documentation/openmp_jobs/openmpcpp/GNU/hello_world.cpp)

If you would like to compile the C++ example using the GNU CPP compiler, you can run the following commands:

```
[login001]$ ml gnu8/8.3.0
[login001]$ g++ -fopenmp -o hello_world hello_world.cpp
```

If you would like to compile the C++ example using the Intel OneAPI, you can run the following commands:

```
[login001]$ ml intel/oneapi/2021.2.0
[login001]$ icpc -qopenmp -o hello_world hello_world.cpp
```

Note: Some compiler options are the same for both Intel and GNU (e.g. "-o"), while others are different (e.g. "-qopenmp" vs "-fopenmp").

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a compute node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a compute node using the following set of commands and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 4 -t 00:05:00 --pty bash
[c001]$ export OMP_NUM_THREADS=4
[c001]$ ./hello_world
```

```
Hello World... from thread =Hello World... from thread =Hello
World... from thread =Hello World... from thread =103
```

If you are currently on a compute node and would like to switch back to the login node then enter the `exit` command as follows:

```
[c001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in listing 20. This batch script corresponds to the parallel program `hello_world.cpp`. This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_openmpcpp
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 4
export OMP_NUM_THREADS=4

./hello_world #the executable name is obtained after compiling
the program
```

Listing 20: Batch Job Script for OpenMP in C++ - hello_world.slurm (documentation/openmp_jobs/openmpcpp/GNU/hello_world.slurm)

The job-script shown in Listing 20 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 20 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
Hello World... from thread =Hello World... from thread =Hello
World... from thread =Hello World... from thread =103
```

8. COMPILING AND RUNNING A SAMPLE FORTRAN + OPENMP PROGRAM

A sample Fortran program is shown in Listing 21. This code will print "`Hello from process: #`" to standard output, where # represents different thread numbers.

```
PROGRAM Parallel_Hello_World
USE OMP_LIB
!$OMP PARALLEL

print *,"Hello from process: ", OMP_GET_THREAD_NUM()
!$OMP END PARALLEL
END
```

Listing 21: Sample Fortran+OpenMP program – hello_world.f90 (documentation/openmp_jobs/openmpf/GNU/hello_world.f90)

If you would like to compile the Fortran example using the GNU Fortran compiler, you can run the following commands:

```
[login001]$ ml gnu8/8.3.0
[login001]$ gfortran -fopenmp -o hello_world hello_world.f90
```

If you would like to compile the Fortran example using the Intel OneAPI, you can run the following commands:

```
[login001]$ ml intel/oneapi/2021.2.0
[login001]$ ifort -qopenmp -o hello_world hello_world.f90
```

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a compute node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a compute node using the following set of commands and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 4 -t 00:05:00 --pty bash
[c001]$ export OMP_NUM_THREADS=4
[c001]$ ./hello_world
 Hello from process:           2
 Hello from process:           0
 Hello from process:           1
```

```
 Hello from process:            3
```

If you are currently on a compute node and would like to switch back to the login node then enter the `exit` command as follows:

```
[c001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 22. This batch script corresponds to the parallel program `hello_world.f90`. This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_openmpf
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 4

export OMP_NUM_THREADS=4
./hello_world #the executable name is obtained after compiling
the program
```

Listing 22: Batch Job Script for Fortran+OpenMP - hello_world.slurm (documentation/openmp_jobs/openmpf/GNU/hello_world.slurm)

The job-script shown in Listing 22 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 22 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
 Hello from process:            2
 Hello from process:            0
 Hello from process:            1
 Hello from process:            3
```

9. COMPILING AND RUNNING A SAMPLE C+CUDA PROGRAM

A sample C program is shown in Listing 23. This code will print "`device count = #`" to standard output, where # is replaced by the actual number of GPUs on the node.

```
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#include <unistd.h>

int main(){

      int count, err;
      cudaGetDeviceCount(&count);
      printf("device count = %d\n", count);
      if(err = cudaSetDevice(count-1)){
          printf("cudaSetDevice error, %d\n", err);
      }
      return 0;
}
```

Listing 23: Sample C+Cuda program - hello_world.cu
(documentation/cuda_jobs/cudac/hello_world.cu)

If you would like to compile the C example using the Nvidia CUDA compiler, you can run the following commands:

```
[login001]$ ml cuda/toolkit/11.3
[login001]$ nvcc -o hello_world hello_world.cu
```

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a GPU node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a GPU node using the following set of commands:

Single GPU

```
[login001]$ srun -p gpu1v100 -n 1 -t 00:05:00 --pty bash
[gpu001]$ ./hello_world
```

```
device count = 1
```

Multiple GPU

```
[login001]$ srun -p gpu2v100 -n 1 -t 00:05:00 --pty bash
[gpu031]$ ./hello_world
device count = 2
```

If you are currently on a GPU node and would like to switch back to the login node then enter the exit command as follows:

```
[gpu001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 24 (single GPU) and Listing 25 (multiple GPU). These batch scripts correspond to the C+CUDA program `hello_world.cu`. These scripts should be run from a login node.

```
#!/bin/bash
#SBATCH -J hello_world_cudac
#SBATCH -o hello_world.txt
#SBATCH -p gpu1v100
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 1

./hello_world #the executable obtained after compiling the
program
```

Listing 24: Batch Job Script for C+CUDA on single GPU– hello_world.slurm (documentation/cuda_jobs/cudac/hello_world.slurm)

```
#!/bin/bash
#SBATCH -J hello_world_cudac
#SBATCH -o hello_world.txt
#SBATCH -p gpu2v100
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 1

./hello_world #the executable obtained after compiling the
program
```

Listing 25: Batch Job Script for C+CUDA on multiple GPU – hello_world2.slurm (documentation/cuda_jobs/cudac/hello_world2.slurm)

The job-script shown in Listing 24 and 25 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
[login001]$ sbatch hello_world2.slurm
```

The output from the Slurm batch-jobs shown in Listing 24 and 25 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
```

Single GPU
`device count = 1`

Multiple GPU
`device count = 2`

10. COMPILING AND RUNNING A SAMPLE CUDA PROGRAM IN C++

A sample C++ program is shown in Listing 26. This code will print "`device count = #`" to standard output, where # is replaced by the actual number of GPUs on the node.

```
#include <iostream>
#include <cuda.h>
#include <stdlib.h>
#include <unistd.h>
using namespace std;

int main(){

 int count, err;
 cudaGetDeviceCount(&count);
 cout<<"device count = "<<count<<"\n";
 if(err = cudaSetDevice(count-1)){
     cout<<"cudaSetDevice error, "<<err<<"\n";
 }
 return 0;
}
```

Listing 26: Sample CUDA program in C++ - hello_world.cu (documentation/cuda_jobs/cudacpp/hello_world.cu)

If you would like to compile the C++ example using the Nvidia CUDA compiler, you can run the following commands:

```
[login001]$ ml cuda/toolkit/11.3
[login001]$ nvcc -o hello_world hello_world.cu
```

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a GPU node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a GPU node using the set of commands below.

Single GPU

```
[login001]$ srun -p gpu1v100 -n 1 -t 00:05:00 --pty bash
[gpu001]$ ./hello_world
device count = 1
```

Multiple GPU

```
[login001]$ srun -p gpu2v100 -n 1 -t 00:05:00 --pty bash
[gpu031]$ ./hello_world
```

```
device count = 2
```

If you are currently on a GPU node and would like to switch back to the login node then enter the `exit` command as follows:

```
[gpu001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 27 (single GPU) and Listing 28 (multiple GPU). This batch script corresponds to the CPP+CUDA program `hello_world.cu`. These scripts should be run from a login node.

```
#!/bin/sh
#SBATCH -J hello_world_cudacpp
#SBATCH -o hello_world.txt
#SBATCH -p gpu1v100
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --gres=gpu:v100:0

./hello_world #the executable name is obtained after compiling
the program
```

Listing 27: Batch Job Script for CUDA in C++ for single GPU - hello_world.slurm (documentation/cuda_jobs/cudacpp/hello_world.slurm)

```
#!/bin/sh
#SBATCH -J hello_world_cudacpp
#SBATCH -o hello_world.txt
#SBATCH -p gpu2v100
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --gres=gpu:v100:0

./hello_world #the executable name is obtained after compiling
the program
```

Listing 28: Batch Job Script for CUDA in C++ for multiple GPU - hello_world2.slurm (documentation/cuda_jobs/cudacpp/hello_world2.slurm)

The job-script shown in Listing 27 and 28 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
[login001]$ sbatch hello_world2.slurm
```

The output from the Slurm batch-job shown in Listing 27 and 28 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
```

Single GPU

```
device count = 1
```

Multiple GPU

```
device count = 2
```

11. COMPILING AND RUNNING A SAMPLE FORTRAN+CUDA PROGRAM

A sample Fortran code is shown in Listing 29. This code will check whether a data transfer is successful or not between Host CPU and device GPU.

```
program cpydata
      use cudafor
      implicit none
      integer, parameter :: n = 256
      real :: a(n), b(n)
      real, device::a_d(n), b_d(n)
      a = 1.0
      a_d = a
      b_d = a_d
      b = b_d
      if (all(a==b)) then
        write(*,*) 'Test  Passed'
      else
        write(*,*) 'Test  Failed'
      end if
end program cpydata
```

Listing 29: Sample Fortran+CUDA code – hello_world.cuf

(documentation/cuda_jobs/cudaf/hello_world.cuf)

For compiling the Fortran example using the CUDA Fortran compiler, you can run the following commands:

```
[login001]$ ml cuda/toolkit/11.3
[login001]$ ml cuda/sdk/nvhpc/21.5
[login001]$ nvfortran -o hello_world hello_world.cuf
```

The executable `hello_world` can be run either in a batch mode using a Slurm batch job-script or interactively on a GPU node.

Running the Executable in Interactive-Mode: The executable can be run interactively on a GPU node using the following set of commands:

```
[login001]$ srun -p gpu1v100 -n 1 -t 00:05:00 --pty bash
[gpu001]$ ./hello_world
 Test  Passed
```

If you are currently on the GPU node, switch back to login node before running the job-script in batch mode using the following command:

```
[gpu001]$ exit
```

Running the Executable in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` is shown in Listing 30. This batch script corresponds to the Fortran+CUDA program `hello_world.cuf`. This script should be run from a login node.

```
#!/bin/sh
#SBATCH -J hello_world_cudaf
#SBATCH -o hello_world.txt
#SBATCH -p gpu2v100
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --gres=gpu:v100:0

./hello_world #the executable name is obtained after compiling
the program
```

Listing 30: Batch Job Script for fortran+cuda - hello_world.slurm (documentation/cuda_jobs/cudaf/hello_world.slurm)

The job-script shown in Listing 30 can be submitted as follows:

```
[login001]$ sbatch hello_world.slurm
```

The output from the Slurm batch-job shown in Listing 29 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
 Test Passed
```