RUNNING CONCURRENT TASKS FROM A JOB AND USING GNU PARALLEL

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

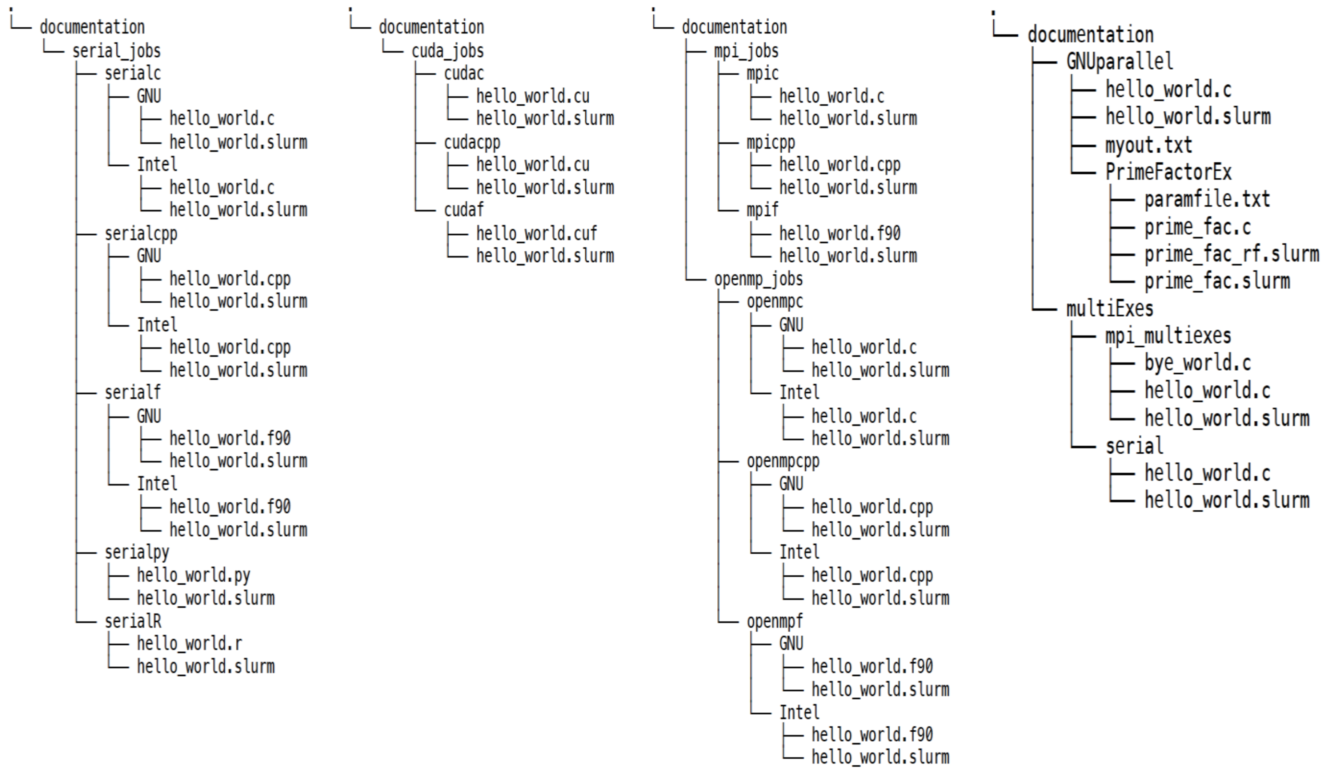
If you want to get a copy of all the programs and scripts used in the examples shown in this document, you can clone the GitHub repository for the examples using the following command:

```
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):



```
.
└── documentation
    └── serial_jobs
        ├── serialc
        │   ├── GNU
        │   │   ├── hello_world.c
        │   │   └── hello_world.slurm
        │   └── Intel
        │       ├── hello_world.c
        │       └── hello_world.slurm
        ├── serialcpp
        │   ├── GNU
        │   │   ├── hello_world.cpp
        │   │   └── hello_world.slurm
        │   └── Intel
        │       ├── hello_world.cpp
        │       └── hello_world.slurm
        ├── serialf
        │   ├── GNU
        │   │   ├── hello_world.f90
        │   │   └── hello_world.slurm
        │   └── Intel
        │       ├── hello_world.f90
        │       └── hello_world.slurm
        ├── serialpy
        │   ├── hello_world.py
        │   └── hello_world.slurm
        └── serialR
            ├── hello_world.r
            └── hello_world.slurm
```

```
.
└── documentation
    └── cuda_jobs
        ├── cudac
        │   ├── hello_world.cu
        │   └── hello_world.slurm
        ├── cudacpp
        │   ├── hello_world.cu
        │   └── hello_world.slurm
        └── cudaf
            ├── hello_world.cuf
            └── hello_world.slurm
```

```
.
└── documentation
    ├── mpi_jobs
    │   ├── mpic
    │   │   ├── hello_world.c
    │   │   └── hello_world.slurm
    │   ├── mpicpp
    │   │   ├── hello_world.cpp
    │   │   └── hello_world.slurm
    │   └── mpif
    │       ├── hello_world.f90
    │       └── hello_world.slurm
    └── openmp_jobs
        ├── openmpc
        │   ├── GNU
        │   │   ├── hello_world.c
        │   │   └── hello_world.slurm
        │   └── Intel
        │       ├── hello_world.c
        │       └── hello_world.slurm
        ├── openmpcpp
        │   ├── GNU
        │   │   ├── hello_world.cpp
        │   │   └── hello_world.slurm
        │   └── Intel
        │       ├── hello_world.cpp
        │       └── hello_world.slurm
        └── openmpf
            ├── GNU
            │   ├── hello_world.f90
            │   └── hello_world.slurm
            └── Intel
                ├── hello_world.f90
                └── hello_world.slurm
```

```
.
└── documentation
    ├── GNUparallel
    │   ├── hello_world.c
    │   ├── hello_world.slurm
    │   ├── myout.txt
    │   └── PrimeFactorEx
    │       ├── paramfile.txt
    │       ├── prime_fac.c
    │       ├── prime_fac_rf.slurm
    │       └── prime_fac.slurm
    └── multiExes
        ├── mpi_multiexes
        │   ├── bye_world.c
        │   ├── hello_world.c
        │   └── hello_world.slurm
        └── serial
            ├── hello_world.c
            └── hello_world.slurm
```

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. RUNNING MULTIPLE COPIES OF ONE OR MORE EXECUTABLES FROM THE SAME SLURM JOB

Depending upon the memory and CPU needs of your application, you may be able to bundle multiple copies of the same application or different applications in the same Slurm job and also have them start concurrently. They can be started concurrently on the same node or on different nodes. This can help you utilize the compute nodes more efficiently and reduce the overall time-to-results. However, please be aware that if your application is memory-intensive, you may want to consider the amount of memory that is required per application process before scheduling multiple concurrent and independent applications on the same node. Oversaturating a node can cause a job to crash due to memory starvation.

A sample C code is shown in Listing 31. This code prints "`Hello World!!: #!`" to standard output where # is replaced by the iteration number, and the iteration number depends upon the argument passed to the executable at run-time.

```
# include <stdio.h>
# include<stdlib.h>
int main(int argc, char *argv[]){
    int n = atoi( argv[1]);
    for(int i=1;i<=n;i++){
     printf("Hello World!!: %d \n",i);
    }
    return 0;
}
```

Listing 31: Sample C program – hello_world.c
(documentation/multiExes/serial/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 -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 -std=c99 -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 multiple copies of an executable concurrently and interactively on a compute node with different input parameters: Multiple copies of the executable **`hello_world`** can be run interactively and concurrently on a compute node with different input parameters using the following set of commands, and the output will be displayed on the terminal:

```
[login001]$ srun -p compute1 -n 5 -t 00:05:00 --pty bash
[c001]$  ./hello_world  1&  ./hello_world  2&  ./hello_world  3&
./hello_world 4& ./hello_world 5
Hello World!!: 1
Hello World!!: 2
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 1
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 5
```

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
```

Bundling Multiple Copies of Executables of a Serial Program in a Slurm Job-Script: A sample Slurm batch job-script to run the executable named **`hello_world`** is shown in Listing 31. This batch script corresponds to the serial program `hello_world.c`. This script should be run from a login node.

```
#!/bin/bash
#SBATCH -J hello_world_cm
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:02:00
#SBATCH -N 1
#SBATCH -n 5

./hello_world 1 &
./hello_world 2 &
./hello_world 3 &
./hello_world 4 &
./hello_world 5  # executable corresponds to code in Listing 31
```

Listing 32: Batch Job Script for C codes – hello_world.slurm
(documentation/multiExes/serial/hello_world.slurm)

The job-script shown in Listing 32 can be submitted as follows:

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

The output from the Slurm job shown in Listing 32 can be checked by opening the output file as follows:

```
[login001]$ cat hello_world.txt
Hello World!!: 1
Hello World!!: 2
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 1
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 5
```

Running Multiple Executables of Parallel Programs Interactively and Concurrently: A sample C code is shown in Listing 33. This code prints "`Hello world from processor #, rank # out of # processors`", where # varies depending upon the number of MPI processes running the executable.

```
#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 33: Sample C+MPI code – hello_world.c
(documentation/multiExes/mpi_multiexes/hello_world.c)

Another sample C code is shown in Listing 34. This code prints "`Bye world from processor #, rank # out of # processors`", where # varies depending upon the number of MPI processes engaged in running the executable.

For compiling the C examples shown in Listings 33 and 34 using Intel OneAPI, you can run the following commands:

```
[login001]$ ml intel/oneapi/2021.2.0
[login001]$ mpicc -o hello_world hello_world.c
[login001]$ mpicc -o bye_world bye_world.c
```

```
#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("Bye 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 34: Sample "Bye World" C+MPI code - bye_world.c
(documentation/multiExes/mpi_multiexes/bye_world.c)

The executables `hello_world` and `bye_world` can be run concurrently in the same interactive session using the set of commands below and the output will be displayed on the terminal.

```
[login001]$ srun -p compute1 -n 24 -N 2 -t 00:05:00 --pty bash
[c001]$ mpirun -np 12 ./hello_world & mpirun -np 12 ./bye_world
Hello world from processor c002, rank 4 out of 12 processors
Hello world from processor c002, rank 7 out of 12 processors
Hello world from processor c002, rank 6 out of 12 processors
Hello world from processor c002, rank 8 out of 12 processors
Bye world from processor c002, rank 7 out of 12 processors
Hello world from processor c002, rank 3 out of 12 processors
Bye world from processor c002, rank 9 out of 12 processors
Bye world from processor c002, rank 11 out of 12 processors
Hello world from processor c002, rank 1 out of 12 processors
Bye world from processor c002, rank 4 out of 12 processors
Hello world from processor c002, rank 2 out of 12 processors
Hello world from processor c002, rank 11 out of 12 processors
Hello world from processor c002, rank 0 out of 12 processors
Bye world from processor c002, rank 5 out of 12 processors
```

```
Hello world from processor c002, rank 5 out of 12 processors
Bye world from processor c002, rank 8 out of 12 processors
Bye world from processor c002, rank 2 out of 12 processors
Bye world from processor c002, rank 10 out of 12 processors
Bye world from processor c002, rank 1 out of 12 processors
Hello world from processor c002, rank 10 out of 12 processors
Bye world from processor c002, rank 6 out of 12 processors
Hello world from processor c002, rank 9 out of 12 processors
Bye world from processor c002, rank 0 out of 12 processors
Bye world from processor c002, rank 3 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
```

Bundling Multiple Executables of Parallel Programs in a Job-Script for Concurrent Execution: A sample Slurm batch job-script to run the executables named `hello_world` and `bye_world` is shown in Listing 35. This batch script corresponds to the parallel programs `hello_world.c` (Listing 33) and `bye_world.c`(Listing 34). This script should be run from a login node.

```
#!/bin/bash
#
#SBATCH -J hello_world_mpicm
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:10:00
#SBATCH -N 2
#SBATCH -n 24

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

Listing 35: Batch Job Script for C+MPI codes – hello_world.slurm (documentation/multiExes/mpi_multiexes/hello_world.slurm)

The job-script shown in Listing 35 can be submitted as follows:

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

The output from the Slurm batch-job shown in Listing 35 can be checked by opening the output file as follows:

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

4. RUNNING AN EXECUTABLE WITH MULTIPLE DIFFERENT PARAMETERS SIMULTANEOUSLY USING GNU PARALLEL

GNU Parallel is a tool that is used for running parameter-sweep applications. It helps in running multiple copies of an executable simultaneously with different parameters read from an input file as a part of the same Slurm job. GNU Parallel is installed on Arc and can be used after loading the `parallel/20210722` module. For detailed information on this tool, please refer to the GNU Parallel documentation page.

The general syntax of the command for using the GNU Parallel tool is as follows:

```
parallel --joblog logfilename.txt ./exefilename {1} ::::
paramfile.txt
```

In the command above, the option "`--joblog`" creates a log file that can be used to resume the execution of the job in case it is interrupted. For resuming a job after an interruption, the "`--resume-failed`" option should be used along with the previously saved log file as shown in the following command:

```
parallel --resume-failed --joblog logfilename.txt ./exefilename
{1} :::: paramfile.txt
```

A sample C code that will serve as our parameter-sweep application is shown in Listing 36. This code prints "`Hello World!!: #`" multiple times to the standard output such that # is replaced by the iteration number, and the total number of iterations is determined by the argument passed at run-time.

```
# include <stdio.h>
# include<stdlib.h>
int main(int argc, char *argv[]){
     int n = atoi( argv[1]);
     for(int i=1;i<=n;i++){
       printf("Hello World!!: %d \n",i);
     }
    return 0;
}
```

Listing 36: Sample C code - hello_world.c
(documentation/GNUparallel/hello_world.c)

If you would like to compile the example in listing 36 using the GNU C compiler, you can run the following commands:

```
[login001]$ ml gcc/11.2.0
[login001]$ gcc -o hello_world hello_world.c
```

If you would like to compile the example in Listing 36 using Intel OneAPI, you can run the following commands:

```
[login001]$ ml intel/oneapi/2021.2.0
[login001]$ icc -std=c99 -o hello_world hello_world.c
```

The contents of the sample input file that provide arguments to the executable in the Slurm batch job script are shown in Listing 37.

```
[login001]$ cat myout.txt
1
2
3
4
5
```

Listing 37: Text file containing arguments for the executables - myout.txt
(documentation/GNUparallel/myout.txt)

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 20 -t 00:05:00 --pty bash
```

```
[c001]$ ml gcc/11.2.0
[c001]$ ml parallel/20210722
[c001]$ parallel --joblog logfilename.txt ./hello_world {1} ::::
myout.txt
Hello World!!: 1
Hello World!!: 1
Hello World!!: 2
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 5
```

If you would like to put the output in separate files for each instance, use the following Parallel option:

```
[c001]$ parallel --results outdir ./hello_world {1} ::::
myout.txt
```

The output files can be found in outdir/1/ directory after running the aforementioned command:

```
[c001]$ cd outdir/1
[c001]$ ls
1  2  3  4  5  seq  stderr  stdout
[c001]$ cd 2
[c001]$ ls
seq  stderr  stdout
[c001]$ cat stdout
Hello World!!: 1
Hello World!!: 2
```

Logs from the parallel command above can be checked by opening the log file as follows:

```
[c001]$ cat logfilename.txt
Seq  Host Starttime JobRuntime     Send Receive   Exitval
Signal    Command
1    :    1626380539.453      0.003     0     18    0     0
./hello_world 1
2    :    1626380539.457      0.009     0     36    0     0
./hello_world 2
```

```
3    :    1626380539.462      0.009     0    54   0    0
./hello_world 3
4    :    1626380539.466      0.008     0    72   0    0
./hello_world 4
5    :    1626380539.471      0.006     0    90   0    0
./hello_world 5
```

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 GNU Parallel Job in Batch-Mode: A sample Slurm batch job-script to run the executable named `hello_world` with GNU Parallel is shown in Listing 38. This batch script corresponds to the serial program `hello_world.c.` This script should be run from a login node.

```
#!/bin/sh
#SBATCH -J hello_world_gnuc
#SBATCH -o hello_world.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 20


ml gcc/11.2.0
ml parallel/20210722


parallel --joblog mylog.txt ./hello_world {1} :::: myout.txt
#executable name hello_world is obtained after compiling the
program
```

Listing 38: Batch Job Script for C code – hello_world.slurm
(documentation/GNUparallel/hello_world.slurm)

The `parallel` command shown in Listing 38 will run the `hello_world` executable with different parameters read from the input file named `myout.txt`. This command will also create a log file named `mylog.txt` that can be used to resume execution in case it is interrupted. This program will print "`Hello World!!: #`", where # varies depending upon input from the `myout.txt`.

The job-script shown in Listing 38 can be submitted as follows :

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

The output from the Slurm batch-job shown in Listing 37 can be checked by displaying the contents of the output file as follows:

```
[login001]$ cat hello_world.txt
Hello World!!: 1
Hello World!!: 1
Hello World!!: 2
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 1
Hello World!!: 2
Hello World!!: 3
Hello World!!: 4
Hello World!!: 5
```

Logs from the Slurm batch-job shown in Listing 38 can be checked by displaying the contents of the log file as follows:

```
[login001]$ cat mylog.txt
Seq  Host Starttime JobRuntime     Send Receive   Exitval
Signal    Command
1    :    1626294133.948      0.003     0     18   0     0
./hello_world 1
2    :    1626294133.952      0.009     0     36   0     0
./hello_world 2
3    :    1626294133.957      0.007     0     54   0     0
./hello_world 3
4    :    1626294133.960      0.007     0     72   0     0
./hello_world 4
5    :    1626294133.964      0.007     0     90   0     0
./hello_world 5
```

5. USING GNU PARALLEL TO FIND THE PRIME FACTORS OF GIVEN NUMBERS

Another example of a parameter-sweep application is shown in the sample C code in Listing 39. This code prints "`The prime factors of # are:*`" multiple times to standard output such that # is replaced by the input numbers, and * is replaced by the prime factors of that number.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(int argc, char** argv){
   int n,i;
   sscanf(argv[1],"%d",&n);
   printf(" The prime factors of %d are:", n);
   while (n%2 == 0){
       n = n/2;
   }
   for(i = 3; i <= (int)sqrt(n); i = i+2){
                while (n%i == 0){
                   printf("%d ", i);
                   n = n/i;
                }
   }
   if (n > 2)
       printf ("%d", n);
   printf("\n");
   return 0;
}
```

Listing 39: Sample parameter-sweep application – prime_fac.c (documentation/GNUparallel/PrimeFactorEx/prime_fac.c)

If you would like to compile the example in listing 39 using the GNU C compiler, you can run the following commands:

```
[login001]$ ml gnu8/8.3.0
[login001]$ gcc -o prime_fac prime_fac.c -lm
```

Note: The math library (libm.so) must be linked in by using **`-lm`** when building the executable.

If you would like to compile the example in listing 39 using Intel OneAPI, you can run the following commands:

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

The contents of the sample input file to provide arguments to the executable **`prime_fac`** is shown in Listing 40.

```
[login001]$ cat paramfile.txt
1234567896
1234567891
1234567899
1234567895
```

```
1234567847
1234567941
1234568927
1234578957
1234689630
1234578960
1234678975
1235678933
1245678998
1345678920
1234567892
1234567844
1234567970
1234568951
1234578981
1234678969
1235678949
1234578993
1234567897
1234567898
1234567891
1234567966
1234568918
1234578983
1234678956
1235678921
1245678955
1345678937
1234567895
1234567895
1234567895
1234567988
1234568936
1234578932
1234678946
1235678913
1245678924
1234567896
1234567892
1234567811
1234567990
1234568979
1234578986
1678977111
1234567896
1234567891
```

```
1234567899
1234567895
1234567847
1234567941
1234568927
1234578957
1234689630
1234578960
1234678975
1235678933
1245678998
1345678920
1234567892
1234567844
1234567970
1234568951
1234578981
1234678969
1235678949
1234578993
1234567897
1234567898
1234567891
1234567966
1234568918
1234578983
1234678956
1235678921
1245678955
1345678937
1234567895
1234567895
1234567895
1234567988
1234568936
1234578932
1234678946
1235678913
1245678924
1456789841
```
Listing 40: Text file containing different arguments for the executable `prime_fac` – paramfile.txt (documentation/GNUparallel/PrimeFactorEx/paramfile.txt)

The executable `prime_fac` can be run using GNU Parallel either in a batch mode using a Slurm 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 set of commands below and the output will be displayed on the terminal. Since **`-n 20`** is specified in the **`srun`** command, 20 cores will be used for running this job. If the execution is interrupted due to any failures or the user exits or interrupts the execution, the option "`--joblog`" is provided to create a log file that can be used to resume the execution of the job.

```
[login001]$ srun -p compute1 -n 20 -t 00:05:00 --pty bash
[c001]$ ml gcc/11.2.0
[c001]$ ml parallel/20210722
[c001]$ parallel --joblog logfilename.txt ./prime_fac {1} ::::
paramfile.txt
```

The output of the interactive job is shown below:

```
 The prime factors of 1234567896 are:3 83 619763
 The prime factors of 1234567891 are:1234567891
 The prime factors of 1234567899 are:3 3 3 3 109 139831
 The prime factors of 1234567895 are:5 11 23 975943
 The prime factors of 1234567847 are:47 251 104651
 The prime factors of 1234567941 are:3 23 47 199 1913
 The prime factors of 1234568927 are:491 1123 2239
 The prime factors of 1234578957 are:3 139 2960621
 The prime factors of 1234689630 are:3 5 37 1112333
 The prime factors of 1234578960 are:3 3 5 19 90247
 The prime factors of 1234678975 are:5 5 17 409 7103
 The prime factors of 1235678933 are:23 23 2335877
 The prime factors of 1245678998 are:622839499
```

Logs from the parallel command can be checked by opening the log file as follows:

```
[c001]$ cat logfilename.txt
Seq     Host    Starttime       JobRuntime      Send    Receive
Exitval Signal  Command
1       :       1626460968.422       0.000      0       49
0       0       ./prime_fac 1234567896
2       :       1626460968.426       0.003      0       48
0       0       ./prime_fac 1234567891
3       :       1626460968.430       0.003      0       56      0
0       ./prime_fac 1234567899
4       :       1626460968.433       0.006      0       52
0       0       ./prime_fac 1234567895
5       :       1626460968.437       0.006      0       51
0       0       ./prime_fac 1234567847
6       :       1626460968.439       0.009      0       54
0       0       ./prime_fac 1234567941
```

```
7       :       1626460968.443      0.008      0       51
0       0       ./prime_fac 1234568927
8       :       1626460968.448      0.003      0       51
0       0       ./prime_fac 1234578957
9       :       1626460968.451      0.006      0       52      0
0       ./prime_fac 1234689630
10      :       1626460968.455      0.009      0       52
0       0       ./prime_fac 1234578960
11      :       1626460968.458      0.006      0       53
0       0       ./prime_fac 1234678975
12      :       1626460968.461      0.006      0       51
0       0       ./prime_fac 1235678933
13      :       1626460968.464      0.006      0       47
0       0       ./prime_fac 1245678998
```

For resuming a job after interruption, the **`--resume-failed`** option should be used along with the previously saved log file as shown in the following command:

```
[c001]$ parallel --resume-failed --joblog logfilename.txt \
./prime_fac {1} :::: paramfile.txt
```

A snippet from the output on the command prompt is shown here:

```
 The prime factors of 1345678920 are:3 3 3 3 5 67 6199
 The prime factors of 1234567892 are:41 7527853
 The prime factors of 1235678913 are:3 3 7 7 1009 2777
 The prime factors of 1456789841 are:13 907 123551
 The prime factors of 1245678924 are:3 7 181 81931
```

Logs from the job run with the **`--resume-failed`** command can be checked as follows:

```
[c001]$ cat logfilename.txt
```

A snippet of the log file is shown below:

```
Seq Host   Starttime JobRuntime    Send   Receive Exitval
Signal   Command
1   :   1626460968.422        0.000    0   49  0    0
./prime_fac 1234567896
2   :   1626460968.426        0.003    0   48  0    0
./prime_fac 1234567891
3   :   1626460968.430        0.003    0   56  0    0
./prime_fac 1234567899
4   :   1626460968.433        0.006    0   52  0    0
./prime_fac 1234567895
5   :   1626460968.437        0.006    0   51  0    0
./prime_fac 1234567847
```

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 job-script to run the executable named `prime_fac` is shown in Listing 41. This batch script corresponds to the serial program `prime_fac.c`. This script should be run from a login node.

```
#!/bin/sh
#SBATCH -J prime_fac
#SBATCH -o prime_fac.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 20

ml gcc/11.2.0
ml parallel/20210722

parallel --joblog logfilename.txt ./prime_fac {1} ::::
paramfile.txt #executable named prime_fac is obtained by
compiling the program
```

Listing 41: Batch Job Script for C code – prime_fac.slurm (documentation/GNUparallel/PrimeFactorEx/prime_fac.slurm)

The job-script shown in Listing 41 can be submitted as follows:

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

Assuming that the job-id corresponding to the submission of the job-script shown in Listing 41 is 8088547, and this job is to be interrupted using scancel, then the following command can be used:

```
[login001]$ scancel 8088547
```

The output from the Slurm batch-job shown in Listing 41 can be checked by displaying the contents of the output file as follows:

```
[login001]$ cat prime_fac.txt
 The prime factors of 1234567896 are:3 83 619763
 The prime factors of 1234567891 are:1234567891
 The prime factors of 1234567899 are:3 3 3 3 109 139831
 The prime factors of 1234567895 are:5 11 23 975943
 The prime factors of 1234567847 are:47 251 104651
 The prime factors of 1234567941 are:3 23 47 199 1913
 The prime factors of 1234568927 are:491 1123 2239
 The prime factors of 1234578957 are:3 139 2960621
 The prime factors of 1234689630 are:3 5 37 1112333
 The prime factors of 1234578960 are:3 3 5 19 90247
 The prime factors of 1234678975 are:5 5 17 409 7103
 The prime factors of 1235678933 are:23 23 2335877
 The prime factors of 1245678998 are:622839499



slurmstepd: error: *** JOB $SLURM_JOBID ON c001 CANCELLED AT
2021-07-16T13:10:17 ***
```

If the above execution is interrupted due to any failure or the user exits or interrupts the execution, the option "`--joblog`" helps in creating a log file that can be used to resume the execution of the job.

Logs from the parallel command can be checked by opening the log file as follows:

```
[login001]$ cat logfilename.txt
Seq     Host    Starttime       JobRuntime      Send    Receive
Exitval Signal  Command
1       :       1626460968.422       0.000      0       49
0       0       ./prime_fac 1234567896
2       :       1626460968.426       0.003      0       48
0       0       ./prime_fac 1234567891
3       :       1626460968.430       0.003      0       56      0
0       ./prime_fac 1234567899
4       :       1626460968.433       0.006      0       52
0       0       ./prime_fac 1234567895
5       :       1626460968.437       0.006      0       51
0       0       ./prime_fac 1234567847
6       :       1626460968.439       0.009      0       54
0       0       ./prime_fac 1234567941
7       :       1626460968.443       0.008      0       51
0       0       ./prime_fac 1234568927
8       :       1626460968.448       0.003      0       51
0       0       ./prime_fac 1234578957
9       :       1626460968.451       0.006      0       52      0
0       ./prime_fac 1234689630
10      :       1626460968.455       0.009      0       52
0       0       ./prime_fac 1234578960
11      :       1626460968.458       0.006      0       53
0       0       ./prime_fac 1234678975
12      :       1626460968.461       0.006      0       51
0       0       ./prime_fac 1235678933
13      :       1626460968.464       0.006      0       47
0       0       ./prime_fac 1245678998
```

For resuming a job after interruption, the `--resume-failed` option should be used along with the previously saved log file.

A sample Slurm job-script to run the executable named `prime_fac` is shown in Listing 41. This batch script corresponds to the serial program `prime_fac.c.` This script should be run from a login node.

```
#!/bin/sh
#SBATCH -J prime_fac_rf
#SBATCH -o prime_fac_rf.txt
#SBATCH -p compute1
#SBATCH -t 00:05:00
#SBATCH -N 1
#SBATCH -n 20

ml gcc/11.2.0
ml parallel/20210722

parallel --resume-failed --joblog logfilename_rf.txt
./prime_fac {1} :::: paramfile.txt #executable named prime_fac
is obtained by compiling the program
```

Listing 42: Batch Job Script for C code – prime_fac_rf.slurm
(documentation/GNUparallel/PrimeFactorEx/prime_fac_rf.slurm)

The job-script shown in Listing 42 can be submitted as follows:

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

The output from the Slurm batch-job shown in Listing 42 can be checked by displaying the contents of the output file as follows:

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

A snippet from the output file is shown here:

```
 The prime factors of 1345678920 are:3 3 3 3 5 67 6199
 The prime factors of 1234567892 are:41 7527853
 The prime factors of 1234567844 are:101 1277 2393
 The prime factors of 1234567970 are:5 73 1691189
 The prime factors of 1234568951 are:7 11 17 943139
```

Logs from the GNU Parallel command using `--resume-failed` can be checked by opening the log file as follows:

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

A snippet from the log file is shown below:

```
Seq Host   Starttime JobRuntime    Send   Receive Exitval
Signal   Command
1   :   1626460968.422        0.000    0   49 0    0
./prime_fac 1234567896
2   :   1626460968.426        0.003    0   48 0    0
./prime_fac 1234567891
3   :   1626460968.430        0.003    0   56 0    0
./prime_fac 1234567899
4   :   1626460968.433        0.006    0   52 0    0
./prime_fac 1234567895
5   :   1626460968.437        0.006    0   51 0    0
./prime_fac 1234567847
```