CS 4380 / CS 5351
Parallel Programming / Parallel Processing
Fall 2020

Final Project

Due: *Thursday, December 3 at 3:00 PM*

All project files, including the report, must be submitted on TRACS. Note that files are only submitted once TRACS indicates a successful submission. See below for which files to submit. The project report must be typed, submitted in PDF, and must include the results of your measurements as well as the answers to the questions. Each answer must be preceded by a copy of the corresponding question listed below. Wrong answers will affect your grade even if the correct answers are also present. The answer to each question is limited to 50 words (including numbers). Responses that exceed 50 words per answer will not be graded. This project must be done *individually*. You must be able to explain all the code you submit.

6.1 OpenMP + CUDA Collatz [50u/30g points]

Make a copy of the provided files **`collatz_hyb_noMPI.*`** and **`collatz_hyb.cu`** located at **`/home1/00976/burtsche/Parallel/`** and study them carefully (prototypes, comments, etc.).

Complete the hybrid OpenMP and CUDA program by writing and inserting the missing code sections based on the "*todo*" comments in the code. Do not change any of the given code. Note the new command-line parameter (an integer) to specify the percentage of the workload that should be processed by the CPU. Use 1024 threads per block on the GPU and do not specify the number of threads on the CPU, i.e., do not use *num_threads()*. Base your code on the code from the earlier OpenMP and CUDA projects.

On the machines in the parallel lab, compile and link the source files as follows to produce a hybrid executable.

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64/:$LD_LIBRARY_PATH
nvcc -O3 -arch=sm_35 -c collatz_hyb.cu -o CUcollatz.o
g++ -O3 -fopenmp -c collatz_hyb_noMPI.cpp -o C1collatz.o
g++ -O3 -fopenmp C1collatz.o CUcollatz.o -lcudart -L
  /usr/local/cuda/lib64/ -o collatz_hyb_noMPI
```

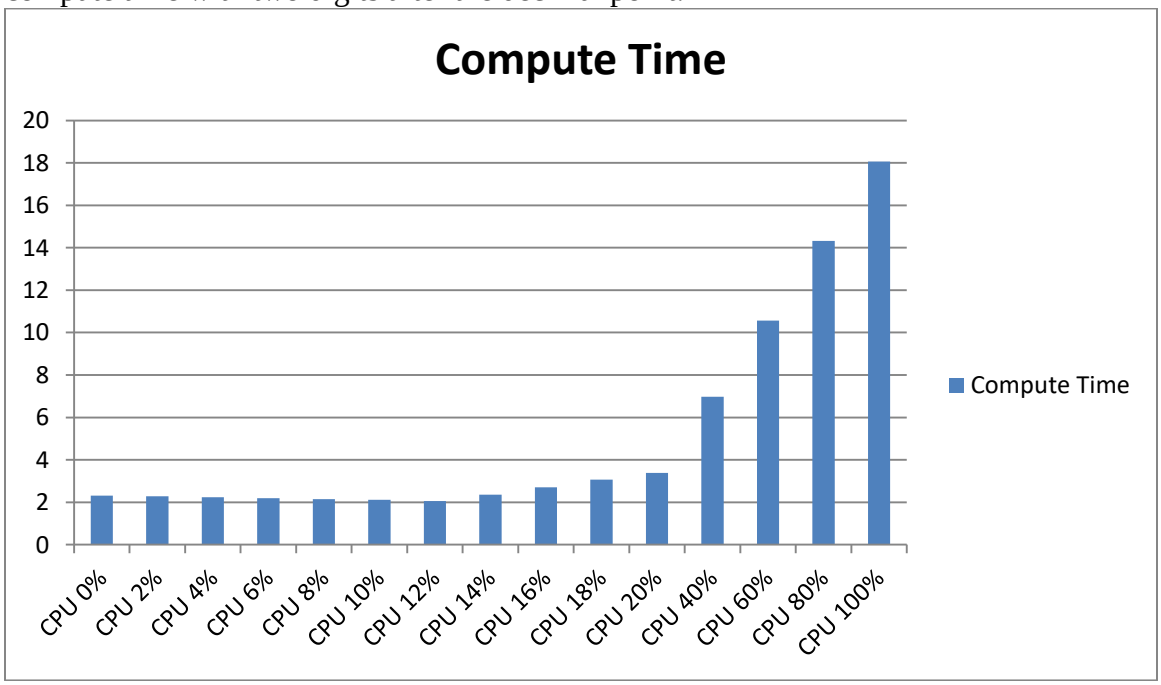
On Lonestar 5, use the following commands to compile and link the files.

```
module load cuda
nvcc -O3 -arch=sm_35 -c collatz_hyb.cu -o CUcollatz.o
icc -O3 -fopenmp -c collatz_hyb_noMPI.cpp -o C1collatz.o
icc -O3 -fopenmp C1collatz.o CUcollatz.o -lcudart -L$TACC_CUDA_LIB -o
  collatz_hyb_noMPI
```

Note that the above commands will not work if copied verbatim into a makefile. Your final code must compile and run correctly on Lonestar 5 and all measurements must be performed on Lonestar 5. Do not submit any results obtained on the lab machines.

Run a few small inputs first to make sure the results are correct, including when executing 0% and 100% of the workload on the CPU. Once everything works, run the code on Lonestar 5 using the provided submission script. Do not modify this script. If you receive a warning message like "srun: Job ... step creation temporarily disabled, retrying", please ignore the warning.

Question 6.1a) Present the compute times (in seconds) in a bar graph with the CPU workload percentage along the x-axis and the time along the y-axis. Include a label for each bar showing the compute time with two digits after the decimal point.



Question 6.1b) Explain the compute-time behavior, especially why it first drops and then increases when increasing the CPU percentage (and decreasing the GPU percentage). What happens at the optimal point where the compute time stops decreasing and starts increasing?

Question 6.1c) Which CPU percentage results in the highest performance?

CPU percentage 12 resulted in the highest performance.

Question 6.1d) How much faster is the best hybrid execution relative to just using the GPU?

The speed up from just using the GPU to best hybrid execution is 1.13

Question 6.1e) How many OpenMP threads are used in the parallel CPU code section and where is this number specified?
There are 20 OpenMP threads being used in the CPU code. This is specified in the job submission script

Question 6.1f) How many CPU sockets and cores do the GPU compute nodes of Lonestar5 have? Is hyperthreading enabled on those cores?

Lonestar5 GPU compute nodes have one CPU socket running ten cores. Hyperthreading is enabled.

Submit the completed **`.cpp`** and **`.cu`** files for this part of the project on TRACS.

6.2 OpenMP + CUDA + MPI Collatz [50u/30g points]

Copy the completed CPU source code from Part 1 into a file named **`collatz_hyb.cpp`** and reuse the **`collatz_hyb.cu`** file without making any changes to it. Then add MPI support to the CPU code in such a way that each node computes one n^{th} of the sequences using a blocked distribution in the MPI portion (do not change the CUDA or OpenMP distributions, which must remain the same as in Part 6.1), where *n* is the number of MPI processes. Within a compute node, the workload is still split among the CPU and the GPU according to the percentage given on the command line.

Follow these steps when adding the MPI support:

1. Include the MPI header file as well as the standard MPI statements.
2. Only process 0 is allowed to print normal program output, but all processes are allowed to read from the command line and print error messages.
3. Make the code print the number of MPI processes.
4. Use the following code to compute the start and stop values:

```
const long cpu_start = 1 + my_rank * bound / comm_sz;
const long gpu_stop = 1 + (my_rank + 1) * bound / comm_sz;
```

```
const long my_range = gpu_stop - cpu_start;
const long cpu_stop = cpu_start + my_range * percentage / 100;
const long gpu_start = cpu_stop;
```

Note that the CPU should execute from iteration **cpu_start** (inclusively) to **cpu_stop** (exclusively) and similarly for the GPU.

5. Execute an MPI barrier right before starting the timer.
6. Reduce the results from each compute node into process 0 before stopping the timer.

Compile, link, and run the files as follows on the lab computers.

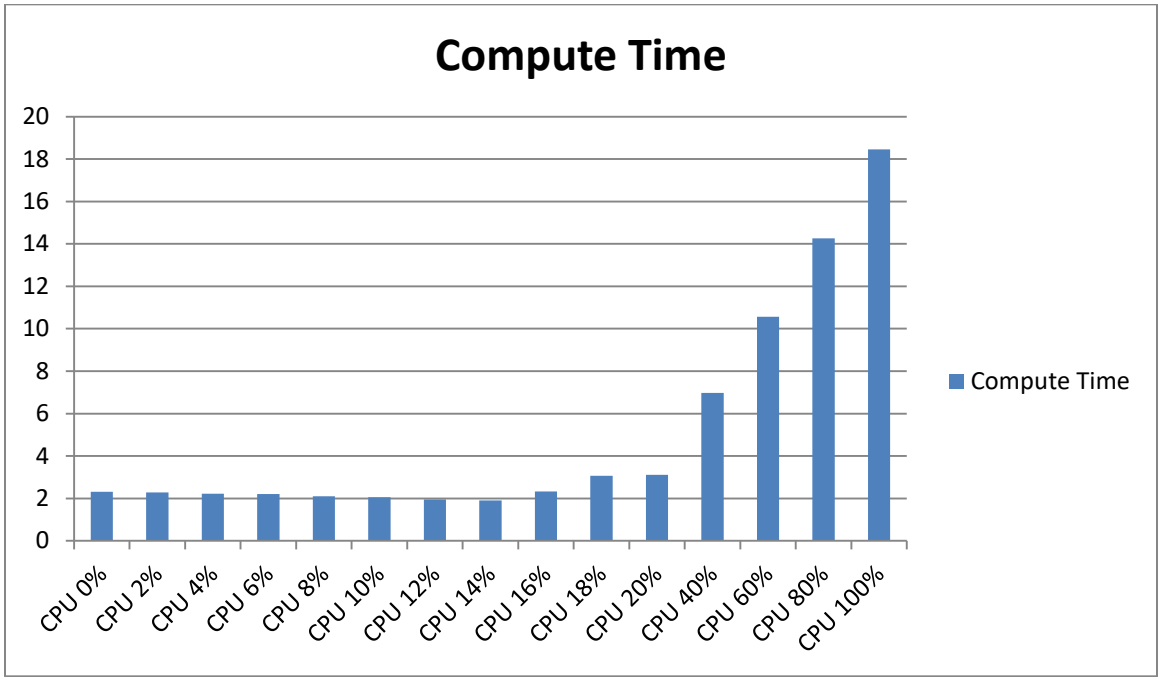
```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64/:$LD_LIBRARY_PATH
module add openmpi-x86_64
nvcc -O3 -arch=sm_35 -c collatz_hyb.cu -o CUcollatz.o
mpicxx -O3 -fopenmp -c collatz_hyb.cpp -o C2collatz.o
mpicxx -O3 -fopenmp C2collatz.o CUcollatz.o -lcudart -L /usr/lo-
  cal/cuda/lib64/ -o collatz_hyb
mpirun -n 2 ./collatz_hyb ...
```

On Lonestar 5, use the following commands to compile and link the files.

```
module load cuda
nvcc -O3 -arch=sm_35 -c collatz_hyb.cu -o CUcollatz.o
mpicxx -O3 -fopenmp -c collatz_hyb.cpp -o C2collatz.o
mpicxx -O3 -fopenmp C2collatz.o CUcollatz.o -lcudart -L$TACC_CUDA_LIB
  -o collatz_hyb
```

Run a few inputs with no more than two MPI processes and make sure the results are correct, including when executing 0% and 100% of the workload on the CPU. Once everything works, run the code on Lonestar 5 using the provided submission script at **/home1/00976/burtsche/Parallel/collatz_hyb4.sub**. Do not modify this script.

Question 6.2a) Present the compute times (in seconds) in a bar graph with the CPU workload percentage along the x-axis and the time along the y-axis. Include a label for each bar showing the compute time with two digits after the decimal point.



Question 6.2b) Which CPU percentage results in the highest performance?

CPU percentage 14% results in the highest performance

Question 6.2c) How much faster is the fastest hybrid execution running on 4 compute nodes relative to the fastest hybrid execution running on 1 compute node (from Part 1)?

The fastest hybrid execution is 1.08 running on 4 compute nodes

Question 6.2d) What is a likely problem with the blocked distribution of work across the compute nodes? Explain.

Load imbalance, especially with collatz we know that the later iterations take a lot more compute time. Threads that get blocks towards the end have a lot more to do than the threads that get the earlier blocks.

Submit the completed `.cpp` file for this part of the project on TRACS.

6.3 OpenMP + CUDA + MPI Fractal [+10u/40g points; up to 10 points of extra credit for undergraduates]

Make a copy of the files **`/home1/00976/burtsche/Parallel/fractal_hyb*`** and study them carefully. Complete the hybrid OpenMP/CUDA/MPI program by writing and inserting the missing code sections based on the "*todo*" comments in the code. Do not change any of the provided code. Note the new command-line parameter (an integer) to specify the percentage of the workload that should be processed by the CPU. Use 1024 threads per block on the GPU and do not specify the number of threads on the CPU, i.e., do not use *num_threads()*. Base your code on the code from the corresponding earlier projects. Parallelize the middle (*for-row*) loop using OpenMP and use the *double* (not the *float*) version of the CUDA code that parallelizes all three *for* loops. Compile and link the source files as shown above for the collatz code.

Run small tests first using no more than 2 MPI processes, check the resulting BMP images, and make sure they are correct, especially at the transition point between the CPU and GPU computation and between compute nodes. Once everything works, run the code on Lonestar 5 using the provided submission script. Do not modify this script.

Question 6.3a) Present the compute times (in seconds) in a bar graph with the CPU workload percentage along the x-axis and the time along the y-axis. Include a label for each bar showing the compute time with two digits after the decimal point.

Question 6.3b) Which CPU percentage results in the highest performance?

Question 6.3c) For the CPU percentage that results in the highest performance, how many frames are assigned to the CPU in each compute node and how many to the GPU?

Question 6.3d) For inputs with relatively few frames, it is better to parallelize the middle (*for-row*) loop than the outer (*for-frame*) loop. Why is that?

Submit the two completed source files on TRACS.

Code Requirements
- Make sure your code compiles.
- Make sure your code is well commented.
- Make sure your code does not produce unwanted output such as debugging messages.
- Make sure your code's runtime does not exceed the specified maximum.
- Make sure your code is correctly indented and uses a consistent coding style.
- Make sure your code does not include unused variables or code, commented out code, etc.

Code Submission
- Delete all files that you do not need anymore such as *.o and *.bmp files.
- Make sure your code complies with the above requirements before you submit it.
- Any special instructions or comments to the grader should be included in a "README" file.

- Upload all the files you need to submit onto TRACS. The report must be in PDF. All other files, including source code, must be plain text files (e.g., *.cpp, *.h, *.sub, and *.cu files).
- Upload each file separately and do not compress them.
- Do not submit any unnecessary files (e.g., provided or generated files).

You can submit your file(s) as many times as you want before the deadline. Only the last submission will be graded. The date and time stamp of the last submitted file determines the late penalty, if any. Be sure to submit at least once before the deadline.

November 19, 2020