CS 4380 / CS 5351
Parallel Programming / Parallel Processing
Spring 2017

Final Project

Due: *Thursday, May 4 at 3:30 PM*

All project files including the report have to be submitted using TRACS. Please follow the instructions at http://tracsfacts.its.txstate.edu/trainingvideos/submitassignment/submitassignment.htm. Note that files are only submitted if TRACS indicates a successful submission. See below for what files to submit. The project report has to list the results of your measurements and provide the answers to the questions. The answer to each question is limited to 50 words. This project has to be done *individually for graduate students* and can be done *in pairs for undergraduates*. You have to be able to explain all of the code that you submit, including code written by your partner in case of a pair project.

Compiling hybrid parallel programs on Stampede

Since nvcc does not support MPI and mpicc does not support CUDA, the source code of programs that combine these parallelization strategies needs to be distributed over multiple files, which have to be compiled and linked in the following way to generate a combined executable:

```
module load cuda
mpicc -xhost -openmp -O3 -c fractal_hyb.cpp -o Cfractal.o
nvcc -O3 -arch=sm_35 -c fractal_hyb.cu -o CUfractal.o
mpicc -xhost -openmp -O3 Cfractal.o CUfractal.o -lcudart \
  -L$TACC_CUDA_LIB -o fractal_hyb
```

Note that the above commands will not work if copied verbatim into a makefile.

Compiling hybrid parallel programs on the lab computers

Use the following commands to generate a combined executable on the lab machines:

```
module add openmpi-x86_64
export LD_LIBRARY_PATH=/usr/local/cuda/lib64/:$LD_LIBRARY_PATH
mpicxx -march=native -O3 -fopenmp -c fractal_hyb.cpp -o Cfractal.o
nvcc -O3 -arch=sm_20 -c fractal_hyb.cu -o CUfractal.o
mpicxx -march=native -O3 -fopenmp Cfractal.o CUfractal.o -lcudart \
  -L /usr/local/cuda/lib64/ -o fractal_hyb

mpirun -n 2 ./fractal_hyb ...
```

Please run only small problem sizes with no more than two MPI processes on the lab machines.

6.1 OpenMP + CUDA Fractal [50u/35g points]

Make a copy of the provided skeleton files (and submission script) at /home1/00976/burtsche/Parallel/fractal_hyb1.* and /home1/00976/burtsche/Parallel/fractal_hyb.cu and study them carefully (prototypes, comments, etc.).

Complete the hybrid OpenMP and CUDA program by writing and inserting the missing code sections following the comments in the code. In addition to the width, there are now two command-line parameters (both integers) to specify the number of frames to compute on the CPU and the number of frames to compute on the GPU. Use 16 threads on the CPU and 512 threads per block on the GPU.

Then compile the files as outlined above. Run a small test first, check the resulting bitmap images, and make sure they are correct, especially at the transition point between the CPU and GPU computation. Once everything works, run the code on Stampede using the provided submission script. Do not modify this script.

Question 6.1a) Graphically present the runtimes in seconds in a single line chart (not a bar graph or a table) with the CPU frames along the x-axis and the runtime along the y-axis.

Question 6.1b) Explain the runtime behavior, especially the cases where the CPU or GPU gets all the frames and the general behavior of the runtime as a function of the CPU frames.

Question 6.1c) Which CPU/GPU distribution of the frames results in the highest performance?

Question 6.1d) Using four digits after the decimal point, what is the runtime of just using the GPU and what is the best hybrid runtime (in seconds)? How much faster is the best hybrid execution relative to just using the GPU?

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

6.2 MPI + OpenMP + CUDA Fractal [50u/35g points]

Copy the completed CPU source code from Part 1 into a file named fractal_hyb2.cpp and reuse the file fractal_hyb.cu without any changes. Then add MPI support to the CPU code in such a way that each node computes one n^{th} of the frames, where n is the number of MPI processes. Within a compute node, the workload is still split among the CPU cores and the GPU according to the number of frames given on the command line. For example, if 8 CPU frames and 8 GPU frames are requested with 4 MPI processes, a total of 16 frames need to be computed, 2 CPU frames and 2 GPU frames on each compute node.
Follow these steps to add MPI support:

1. Only allocate enough memory on the GPU to store the frames it needs to compute and adjust the array index, frame, and other calculations accordingly.
2. Include the MPI header file as well as the standard MPI statements.
3. 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.

4. Make process 0 print the total number of frames, CPU frames, and GPU frames. In the above example, it should print 16 frames, 8 CPU frames, and 8 GPU frames.
5. Make the code print "Hybrid2" and the number of MPI processes.
6. Use the following code to compute the workload for each node (assuming frames, cpu_frames, and gpu_frames have already been divided by comm_sz):
```
const int from_frame = my_rank * frames;
const int mid_frame = from_frame + cpu_frames;
const int to_frame = mid_frame + gpu_frames;
```
   In each compute node, the CPU cores should compute frames "from_frame" through "mid_frame" using OpenMP and the GPU should compute frames "mid_frame" through "to_frame" (including the former but excluding the latter in both cases).
7. Each MPI process (we are only running one process per compute node as we are using OpenMP to distribute the work among the cores within a node) other than process 0 is only allowed to allocate memory to hold one n^{th} of the frames, where n is the number of MPI processes. Process 0 should, additionally, allocate memory for all the frames.
8. Execute an MPI barrier right before starting the timer.
9. Gather the results from each compute node into process 0 to combine the frames into a single array before stopping the timer.
10. Free all dynamically allocated CPU memory after stopping the timer (but not the GPU memory).

Compile the files as outlined above. Then run the code using the provided submission scripts at /home1/00976/burtsche/Parallel/fractal_hyb2_*.sub. You are not allowed to modify these scripts. Develop and test your code on the lab machines using small problem sizes. Check the resulting bitmaps and make sure they are correct, especially at the transition point between MPI processes. Run your code on Stampede with the three provided submission scripts.

Question 6.2a) Graphically present the runtimes in seconds in a single line chart (not a bar graph or a table) with the CPU frames along the x-axis and the runtime along the y-axis. Use different lines styles or colors for the three different node counts.

Question 6.2b) Discuss the scaling and average speedup across node counts (in words).

Question 6.2c) Comment on the usefulness of combining the three parallelization schemes (MPI, OpenMP, and CUDA). Which one(s) boost performance the most?

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

6.3 Presentation [0u/30g points; graduate students only]

Create a three-minute presentation of your project. You will have to present your slides using the classroom computer in DERR 240 on May 4 starting at 5 PM, i.e., just before the final exam. Begin with a summary of your project. Then explain what and how you measured. Graphically present your results and provide a discussion of the key findings. Provide insight, not just data. Finally, draw conclusions. Submit your presentation slides 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, unreachable 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 has to be in PDF. All other files, including source code, have to be plain text files (e.g., *.cpp files).
- For group projects, include a comment at the beginning of the source code and the report that lists both group members. The two group members have to submit identical files on TRACS.
- 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. Be sure to submit at least once before the deadline.

April 20, 2017