Programming Assignment 5 - MPI

Due: Thurs 12/1 (during class)

Ohio Supercomputer Center

MVAPICH2, an open source standard MPI implementation, is installed and available on the Oakley cluster. Use the *module list* command to verify it is loaded, and *module load mvapich2* if necessary. To allocate a proper node and begin an interactive computing session, use: **qsub -I -l walltime=0:59:00 -l nodes=1:ppn=12** (qsub -"capital eye" -"little ell" walltime . . . -"little ell" nodes . . .).

The qsub command often obtains an interactive session within a few minutes. Note, however, that OSC is a shared resource, and your wait will very with current system load. The current load on Oakley is unusually high, so please plan accordingly and use batch submissions when you can. We are guests on the OSC cluster, so please practice good citizenship. You can compile your MPI programs on the "login nodes." Only start a qsub batch **when you are ready to run your program.** Please "exit" from qsub once your test is complete. If you need to make more than very minor changes to your source, exit qsub, edit as required, and then start a new qsub batch session to re-test.

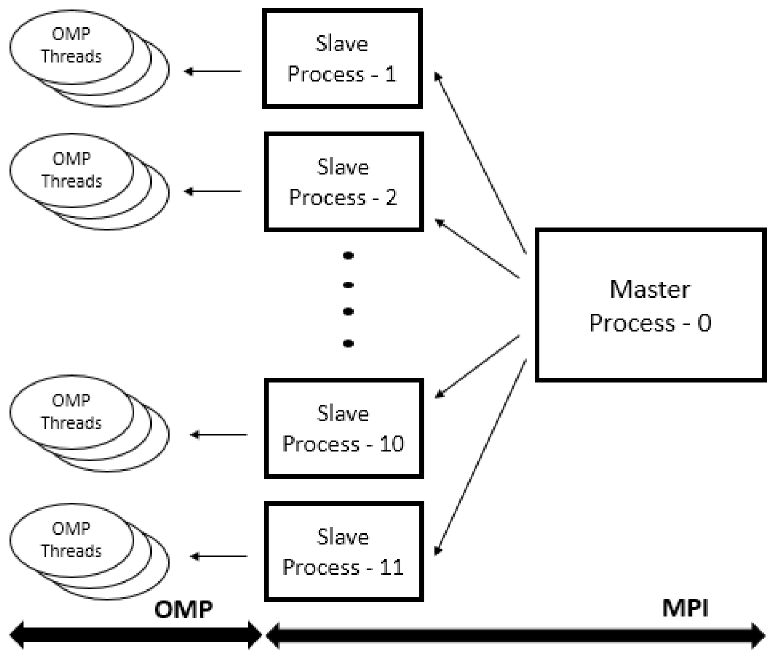
The MPI compiler is *mpicc*, which uses the same options as the gcc C compiler.

To execute MPI programs on Oakley first start an interactive qsub session, and then use *mpiexec -np 12 a.out input_image.bmp output_image.bmp*, where a.out is your compiled program name. The "-np 12" option will specify the 12 processes required for this assignment.

Assignment 5

Large-Scale Parallel Program: Using the serial version of Edge detection algorithm developed in Lab4. Design a large-scale parallel version of Edge detection algorithm using MPI and OpenMP. Investigate the performance improvement between serial and distributed parallel version.

Distribute the Edge detection operation over 12 processors using MPI and within each process use OMP to multi-thread.



Structure the body of your program as follows:

```
    master process                              slave process
read image file                             read image file
barrier                                     barrier
start timer
repeat until (black_cells < 75% pixels)     repeat until (black_cells < 75% pixels)
{                                           {
  barrier                                     barrier
  Sobel operation                             Sobel operation
  send or reduce black cell count             send or reduce black cell count
}                                           }
Gather image pixels from all Slaves         Send image pixels to master
stop timer
Write to image file
```

Note: Each MPI process can read from same file. However write to same file is dangerous.

All the MPI Process should be connected to MPI_COMM_WORLD. Breakdown the image into multiple blocks across row or column and let each MPI process work on the provided block. Instrument your code using standard MPI blocking primitives, MPI_Send, MPI_Recv, MPI_Reduce, MPI_Allreduce and MPI_Barrier. Once convergence is achieved Master MPI Process should collect all new pixel values from all slaves and write to the output bmp image file.

Further paralleize your Sobel Operator code by using OpenMP to work on multiple pixels within each block simultaneously. You can use the OMP programming techniques from Lab3.

Reading and Writing Images

Your program will work on 24-bit bmp style image format. To help you with reading and writing images, a bmp reader support library will be provided to you. You will be provided with bmp_reader.o and read_bmp.h file with the following support API calls:
Note: The API's will are implemented in C and will not belong to class, unlike lab4

- **void* read_bmp_file(FILE *bmp_file)**:
  Will take in a FILE* pointing to the input image file and will return a buffer pointer (void *). The buffer returned will contain the pixel values arranged in linear fashion running column first. i.e. if your image is N x M pixels. The buffer will have M pixels of first row followed by M pixels of 2nd row and so on. Note: You have to open the image file in 'rb' mode into the FILE* before calling this function. Ensure to free the buffer*, returned by the function before exiting the program.
- **void write_bmp_file(FILE *out_file, uint8_t *bmp_data)**:
  Will take in FILE* pointing to output image file and buffer pointer containing the pixel values arranged in linear fashion. Note: The output file should be opened in 'wb' mode into FILE* before calling write_bmp_file. Ensure to free the buffer* before exiting the program
- **uint32_t get_image_width()**;
  Will return width of the input image. Note: The image file must be loaded used read_bmp_file before calling this function
- **uint32_t get_image_height()**;
  Will return height of the input image. Note: The image file must be loaded used read_bmp_file before calling this function
- **uint32_t get_num_pixel()**;
  Will return total number of pixels in the image. Note: The image file must be loaded used read_bmp_file before calling this function

Instrumentation

- Compile your program with optimizer level3 (-O3) and (-fopenmp) option.
- The program should execute similar to :
  mpiexec -np 12 ./lab5.out <input_image.bmp> <output_image.bmp>
- Sample images, library file bmp_reader.o and read_bmp.h for lab5 will be shared in OSC path:
  /fs/project/PAS1241/lab5_lib
- This lab should be written in C using the new library file from the above shared path
- Your program should output
  a) Time taken for serial execution
  b) Time taken for MPI execution
  Results can adhere to format below but not restricted:

```
mpiexec -np 12 ./lab5.out image_1.bmp out_image.bmp
*****************************************************************
Image Info::
           Height=3658       Width=2962
Time taken for serial operation: 12.0457 sec
Threshold during convergence: 69

Time taken for MPI operation: 1.6857 sec
Threshold during convergence: 69
*****************************************************************
```

Submission and Reporting

- Submit your MPI source file following the general submission guidelines for the previous labs, with the following specifics:
  - name your program file <lastname>_<firstname>_lab5.c
  - provide a single make file called "makefile" that will name your executable "lab5.out".
- Submit your printed report during class on Tues.11/29. Include in your report your name and section (2:20pm or 12:45pm)
- Summarize timing results for provided sample images, being sure to answer following questions
  - Did your program perform better in serial or parallel version?
  - Explain your workload distribution.
  - Compare the timing results with CUDA version. Did you notice any difference?
  - Report if any reduction and synchronization mechanism used
  - Were there any surprises you encountered in this exercise?