# Programming Assignment 5 - hierarchical parallelism

Due: Thurs 11/30 11:59pm

**Ohio Supercomputer Center**

MVAPICH2, an open source standard MPI implementation, is installed and available on the Owens 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=16** (qsub -"capital eye" -"little ell" walltime . . . -"little ell" nodes . . . ). (Note: ppn=16 is appropriate for Owens, use ppn=12 if you are testing on Oakley.)

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 load on Owens is anticipated to increase significantly near the end of the semester. Please plan accordingly. 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 significant revisions to your source, exit qsub, edit as required, and then start a new qsub batch session to re-test. Also, do not forget we have access to the debug and batch queues as well.

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

To execute MPI programs on Owens first start an interactive qsub session, and then use *mpirun -np 16 ./a.out input_image.bmp output_image.bmp*, where a.out is your compiled program name. The "-np 12" option will specify the 16 processes required for this assignment.

**Assignment 5**

**Large-Scale Parallel Program:** Using the serial version of the sobel edge detection algorithm from lab4, design a large-scale parallel version in C using both MPI and OpenMP. Investigate the performance improvement between the serial and distributed parallel versions.

Distribute your edge detection operations over 16 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, writing to the same file is dangerous.*

Use the the default communicator, MPI_COMM_WORLD, for inter-process communication. Breakdown the image into multiple blocks across rows or columns and let each MPI process work on a given block. Communicate and synchronize your code using standard MPI blocking primitives: MPI_Send; MPI_Recv; MPI_Reduce; MPI_Allreduce; and, MPI_Barrier. Once convergence is achieved the “master” MPI process should collect all new pixel values from all “slave” processes and write to the combined output bmp image file.

Using OMP programming techniques learned in lab3, further paralleize your Sobel operator code to work on multiple pixels within each block concurrently.

**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:

- **void* read_bmp_file(FILE *bmp_file)**:
  Given a FILE* pointing to the input image file, will return a buffer pointer of type void*. The indicated return buffer will contain pixel values arranged in row-major format. (i.e. if your image is N x M pixels, the buffer will have the M pixels of the first row followed by the M pixels of the 2nd row, etc.)

  Note: read_bmp_file() assumes the image file has been opened in 'rb'. Be sure to free the buffer, returned by the function, before exiting your program.
- **void write_bmp_file(FILE *out_file, uint8_t *bmp_data)**:
  Input parameters are: FILE* pointing to output image file; buffer pointer containing the pixel values.

  Note: The output file should be opened in 'wb' mode before calling write_bmp_file. Be sure to free your pixel buffer before exiting your 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 mpicc and using the optimizer level3 (-O3) and (-fopenmp) options.
- The program should execute similar to :
  **mpirun -np 16 ./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:
  **<to be provided>**.
- Your program should output
  a) Time taken for MPI parallel execution
  b) Threshold for convergence
  Results can adhere to the format below but this example format is not a requirement:

```
mpiexec -np 12 ./lab5.out image_1.bmp out_image.bmp
*************************************************************
Image Info::
          Height=3658       Width=2962

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

## Submission and Reporting

- Submit your MPI source file following the guidelines for lab4, 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 .pdf-format report via Carmen by the due date.
  - Be sure to include in your report your name and section (12:45pm or 2:20pm) .
- Summarize timing results for provided sample images, being sure to answer following questions
  - Compare your run-time with your serial and cuda parallel versions from lab4. Did your achieve the best performance with the serial or a parallel version? Why?
  - Explain your workload distribution strategy and why you selected it.
  - Report and comment onany reduction and/or synchronization mechanisms you used.
  - Were there any surprises you encountered in this exercise?