MPCS 51087
Problem Set 3

March 2018

1 MPI Latency and Bandwidth

Due Date: Friday, May 4, 2018 by 11:59 pm

For the first portion of this problem set, you will write two small codes that test the latency and bandwidth of communication between two nodes of the Midway cluster.

1.1 Latency (15 points)

Write a basic "ping-pong" latency test measuring the time it takes for a message to be sent from MPI rank A to MPI rank B, and then back to MPI rank A. The message data package should be a single integer in size. You should have a loop in your code to perform the test many times ($> 10,000$) and report the average "round trip" ping latency time in microseconds that you measured. Once you've written your code, use it to test out the following two latency parameters on Midway:

1) The intra-node ping time. That is, you should run your code on an allocation for a single node, as in:

```
#!/bin/bash
#SBATCH --time=00:05:00
#SBATCH --partition=sandyb
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=1
#SBATCH --exclusive

module load mvapich2
mpirun -n 2 ./my_ping_test_code
```

2) The inter-node ping time. This test must be performed between two different nodes on Midway, so you should allocate two nodes with 1 MPI rank per node, as in:

```
#!/bin/bash
#SBATCH --time=00:05:00
#SBATCH --partition=sandyb
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --exclusive

module load mvapich2
mpirun -n 2 ./my_ping_test_code
```

In your PDF writeup, report the average latency time you observed with your code for both configurations.

1.2 Bandwidth (15 points)

Write a basic bandwidth test to measure the data bandwidth (in GB per second) between two MPI ranks. Experiment with different message packet sizes of 1 KB, 1 MB, and 1 GB and record the bandwidth values for both inter-node and intra-node communications on Midway. As you will be testing one-way communication, be sure to make appropriate use of MPI_Barrier() or other synchronization functions to ensure the communication has actually been completed, rather than just buffered. As these are much larger packet sizes than in the latency tests, you may need to greatly reduce the number of repetitions of the tests so that your tests run in a reasonable timeframe.

In your PDF writeup, create a table to report the 6 values you measured.

2 Julia Sets

In this section, you will be creating a code that plots Julia sets $f(z)$, which are defined by the following equation for complex values z and c:

$$f(z) = z^2 + c \tag{1}$$

More information on Julia sets is available at `https://en.wikipedia.org/wiki/Julia_set`. For the purposes of this assignment the important thing to know is that we can use an iterative algorithm to determine if any given point z in complex space is within a Julia set defined by some fixed value of c. Using this iterative test, we can create a plot of a Julia set by testing many points in a domain and plotting the results. Algorithm 1 below gives the full algorithm you'll need to plot a Julia set for a particular choice of c, on the real domain of $-1.5 < x < 1.5$ and imaginary domain of $-1.0 < y < 1.0$.

Algorithm 1 Normal Julia Set Calculation For $c = -0.7 + 0.26i$

```
1:  Input N
2:  Allocate $N \times N$ grid of integers $P$                ▷ Allocate Pixels
3:  $c_x = -0.7$
4:  $c_y = 0.26$
5:  $\Delta_x$ = 3.0 / N
6:  $\Delta_y$ = 2.0 / N
7:  for $0 < i < N$ do
8:      for $0 < j < N$ do
9:          $z_x = -1.5 + \Delta_x i$                          ▷ Real component of $z$
10:         $z_y = -1.0 + \Delta_y j$                          ▷ Imaginary component of $z$
11:         iteration = 0
12:         MAX = 1000
13:         while ($z_x^2 + z_y^2 < 4.0$) AND (iteration ≤ MAX) do
14:             tmp = $z_x^2 - z_y^2$
15:             $z_y = 2 * z_x * z_y + c_y$
16:             $z_x$ = tmp + $c_x$
17:             iteration = iteration + 1
18:         end while
19:         P[i][j] = iteration
20:     end for
21: end for
```

For reference, a plot of this particular Julia set is given in Figure 1 below so you know what to aim for.

2.1 Serial (5 points)

To begin, implement a serial version of your Julia set code. Output your data matrix to disk (in binary format), generate a plot using a plotting program of your choice, and submit your plot as part of your PDF writeup. As the pseudo-code above results in pixels with a rectangular domain, you may wish to adjust the aspect ratio with your plotter to more accurately capture the shape of the physical domain.

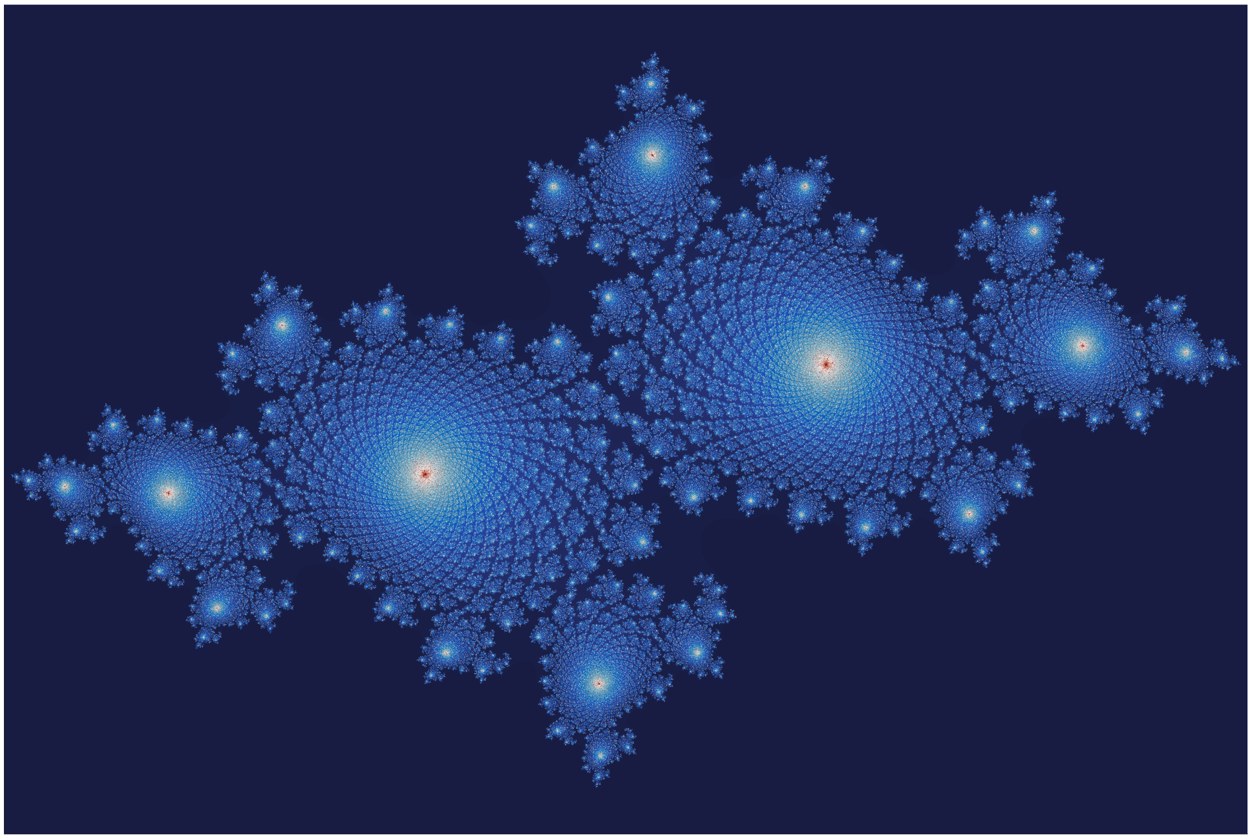

Figure 1: A Julia Set $f(z) = z^2 + c$, where c = -0.7 + 0.26i

2.2 MPI: Static Decomposition (15 points)

Parallelize your code with MPI using a static work decomposition. That is, each MPI rank should process (nearly) the same number of pixels. You are free to implement the static decomposition in any way you choose – e.g. it is not necessary to use an MPI Cartesian decomposition. Your decomposition scheme must work correctly for any number of MPI ranks. In the event that N_{pixels} is not evenly divisible by N_{ranks}, make sure that the pixels are distributed as evenly as possible across ranks.

Unlike the advection code in Problem Set 2, in Algorithm 1 each pixel is calculated independently, so there is no need for ghost cell exchanges. Also, an important constraint is that, when running on multiple ranks, no one rank may allocate space to hold all $N \times N$ pixels in its local memory at once. i.e., each MPI rank should only allocate enough space to store the pixels that it computes.

Also, your domain decomposed code must be able to generate a correct plot of the result. Similar to Problem Set 2, you should utilize MPI File I/O routines to serialize your data to disk, without requiring that a single MPI rank be able to store the full problem in memory at one point. Use binary format for writing to file. While you should generate a plot to ensure your code is working correctly, you do not need to submit the parallel-generated plot as part of your writeup.

In the following subsection, you will implement an alternative decomposition scheme. Your code should allow for either scheme to be used at runtime, as specified by the first command line argument. The static decomposition scheme should be selected with an argument of `static` while the dynamic scheme you implement in the next section should be indicated by an argument of `dynamic`.

2.3 MPI: Dynamic Load Balancing (20 points)

Parallelize your code with MPI using a dynamic load balancing scheme, also known as a "boss-worker" algorithm. A single "boss" MPI rank will keep track of what work needs to be done and handle file I/O, while all other MPI ranks serve as “worker” ranks. The worker ranks perform tasks assigned by the boss rank. In the context of the Julia Set computation, a single block of work comes in the form of a set of pixels to compute the Julia Set values for. I.e., a worker node will request a task, the boss processor will send the worker processor a message indicating which coordinates or indices to compute the Julia Set integer values for. Then, the worker will send the boss the results when it has completed its computations and request more work.

Similar to the static decomposition method, you are not allowed to allocate a matrix big enough to hold the entire global problem in memory at once on the master processor. Rather, the boss should output results to file periodically in binary format as they are received from worker processors over the course of the computation.

You should have a chunk size parameter so as to allow you to alter the number of pixels to assign for work requests. Run your code on a on 12,000 × 12,000 problem on 2 nodes of Midway using 32 MPI ranks, and generate a plot for a variety of chunk sizes to determine an efficient setting. You will use your optimized chunk size parameter for performance analyses in the next section.

2.4 Performance Analysis (20 points)

Perform a strong scaling analysis out to 2 nodes (using 1,2,4,8,16, and 32 MPI ranks) on Midway for a fixed problem size of 12,000 × 12,000 using both your static decomposition method and optimized dynamic load balancing method. When running your static decomposition, be sure to include file I/O in your timing so as to make an apples to apples comparison to your boss-worker method. Be sure to delete or overwrite your output files each run so as to minimize disk space usage on midway.

Add your strong scaling plot to your PDF writeup. Briefly compare your results qualitatively to the results you observed when experimenting with the static and dynamic OpenMP thread schedulers in your advection code from problem set 1.

2.5 Compiling, Code Cleanliness, and Documentation (10 pts)

We plan on compiling and running your code as part of grading it. You must include a makefile capable of compiling your code into a runnable executable. Also, while we are not grading to any rigid coding standard, your code should be human readable and well commented so we can understand what is going on.

2.6 What to Submit

You submission should be in the form of a zip or tar file containing the following items:

- Your source code for your latency and bandwidth tests, along with your Julia set code.
- A makefile
- Your plotting script(s)
- A single writeup (in PDF form) containing all plots and discussions asked for in the assignment