Homework 5: due by 11:59pm on Friday, April 19, 2019 (automatic extension until 11:59pm on Sunday, April 21, 2019)

Instructors: Mackale Joyner, Zoran Budimlić.

All homeworks should be submitted through the Autograder, and also committed in the svn repository at `https://svn.rice.edu/r/comp322/turnin/S19/`*your-netid*`/hw_5` that we will create for you. In case of problems committing your files, please contact the teaching staff at `comp322-staff@mailman.rice.edu` before the deadline to get help resolving for your issues.

Your solution to the written assignment should be submitted as a PDF file named `hw_5_written.pdf` in the hw_5 directory. This is important — you will be penalized 10 points if you place the file in some other folder or with some other name. The PDF file can be created however you choose. If you scan handwritten text, make sure that you use a proper scanner (not a digital camera) to create the PDF file. Your solution to the programming assignment should be submitted in the appropriate location in the hw_5 directory.

This homework has an automatic extension until April 21, 2019. You can use slip days beyond then if needed. The slip day policy for COMP 322 is similar to that of COMP 321. All students will be given 3 slip days to use throughout the semester. When you use a slip day, you will receive up to 24 additional hours to complete the assignment. You may use these slip days in any way you see fit (3 days on one assignment, 1 day each on 3 assignments, etc.). Slip days will be automatically tracked through the Autograder.

Other than slip days, no extensions will be given unless there are exceptional circumstances (such as severe sickness, not because you have too much other work). Such extensions must be requested and approved by the instructor (via e-mail, phone, or in person) before the due date for the assignment. Last minute requests are likely to be denied.

If you see an ambiguity or inconsistency in a question, please seek a clarification on Piazza (remember not to share homework solutions in public posts) or from the teaching staff. If it is not resolved through those channels, you should state the ambiguity/inconsistency that you see, as well as any assumptions that you make to resolve it.

Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates, the teaching assistants and the professor, but you should never misrepresent someone else's work as your own. If you use any material from external sources, you must provide proper attribution.

1 Written Assignment: Locality with Places and Distributions (25 points)

The use of the HJlib `place` construct (Lecture 34) is motivated by improving locality in a computer system's memory hierarchy. We will use a very simple model of locality in this problem by focusing our attention on remote reads. A remote read is a read access on variable V performed by task T0 executing in place P0, such that the value in V read by T0 was written by another task T1 executing in place P1 $\neq$ P0. All other reads are local reads. By this definition, the read of A[0] in line 8 in the example code below is a local read and the read of A[1] in line 9 is a remote read, assuming this HJlib program is run with 2 places, each place with one worker thread.

```
1.        finish {
2.          place p0 = place(0); place p1 = place(1);
3.          double[] A = new double[2];
4.          finish {
5.            async at(p0) { A[0] = ... ; } async at(p1) { A[1] = ... ; }
6.          }
7.          async at(p0) {
8.            ... = A[0]; // Local read
9.            ... = A[1]; // Remote read
10.         }
11.       }
```

Consider the following variant of the one-dimensional iterative averaging example studied in past lectures. We are only concerned with local vs. remote reads in this example, and not with the overheads of creating `async` tasks.

```
1.      dist d = dist.factory.block([1:N]); // generate block distribution (Lecture 31)
2.      for (point [iter] : [0:M-1]) {
3.        finish for(int j=1; j<=N; j++)
4.          async at(d[j]) {
5.            myNew[j] = (myVal[j-1] + myVal[j+1]) / 2.0;
6.          } //finish-for-async-at
7.        double[] temp = myNew; myNew = myVal; myVal = temp;
8.      } // for
```

1. **(10 points)** Write an exact (not big-O) formula for the total number of remote reads in this code as a symbolic function of the array size parameter, N, the number of iterations, M, and the number of places P (assuming that the HJlib program was executed using P places, and 1 worker thread per place). Explain your answer.

2. **(10 points)** Repeat part 1 above if line 1 was changed to "`dist d = dist.factory.cyclic([1:N]);`". Explain your answer.

3. **(5 points)** What conclusions can you draw about the relative impact of block vs. cyclic distributions on the number of remote reads in this example?

2 Written Assignment: Message Passing Interface (25 points)

Consider the MPI code fragment shown below when executed with two processes:

1. **(10 points)** What value will be output by the print statement in process 0? Explain your answer. (Your answer should indicate which MPI calls on which ranks matched with each other.)
2. **(15 points)** How will the output change if the Irecv() call is replaced by Recv() (and the Wait() call eliminated)? Explain your answer. (Your answer should indicate which MPI calls on which ranks matched with each other.)

```
1.        int rank, size, next, prev;
2.        int n1[] = new int[1]; int n2[] = new int[1];
3.        int tag1 = 201, tag2 = 202;
4.        Request request; Status status;

6.        size = MPI.COMM_WORLD.Size();
7.        rank = MPI.COMM_WORLD.Rank();
8.        next = (rank + 1) % size;
9.        prev = (rank + size - 1) %size;
10.       n1[0] = rank*10 + 1; n2[0] = rank*10 + 2;

11.       if ( rank == 0 ) {
12.         request= MPI.COMM_WORLD.Irecv(n1,0,1,MPI_INT,prev,tag1);
13.         MPI.COMM_WORLD.Send(n2,0,1,MPI_INT,next,tag2);
14.         status = MPI.COMM_WORLD.Wait(request);
15.         System.out.println("Output = " + n1[0]);
16.       } else { // rank == 1
17.         MPI.COMM_WORLD.Recv(n1,0,1,MPI_INT,prev,tag2);
18.         n2[0] = n1[0];
19.         MPI.COMM_WORLD.Send(n2,0,1,MPI_INT,next, tag1);
20.       }
```

3 Programming Assignment (50 points total)

3.1 Preliminaries

We highly recommend that you install the VirtualBox virtual machine on your computer and the COMP322-MPI-S19.ova virtual machine image on your laptop as we did in Lab 10, so that you can test and debug your solution locally as you are developing, without the need to run it on NOTS. This will also significantly reduce the traffic on NOTS and autograder.

In the Actors lab, we used HJlib Actors to approximate π. In the Message Passing Interface (MPI) lab, we will gain experience with distributed computing using MPI. The goal of this programming assignment is to exploit inter-process parallelism (by communicating using MPI) while computing π.

3.2 Pi Computation using Bailey-Borwein-Plouffe formula

The assignment involves computing π to a specified precision using MPI. The following formula can be used to compute π:

$$\pi = \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left(\frac{1}{16} \right)^n$$

The `PiNumTermsSerial.java` file contains a simple sequential algorithm for computing π using Java's `BigDecimal` data type, that runs for a fixed number of iterations. The `PiNumTermsMpi.java` file contains a parallel version of `PiNumTermsSerial.java` using MPI for inter-process communication and parallelism. The provided code includes implementations of the `send*Message()` and `recv*Message()` variants of communicating Java objects between the processes.

In contrast, the `PiPrecisionSerial.java` file contains a more realistic sequential algorithm that uses a `while` loop to compute more and more terms of the series until a desired precision is reached. **Your first task is to convert the sequential program in `PiPrecisionSerial.java` (for computing π to a desired precision) to a distributed parallel program in `PiPrecisionMpi.java` by using MPI for communication between the processes.** We have already provided a version of `PiPrecisionMpi.java` with `TODO` comments as helpful hints. Your task will include filling in code to address the comments in the `TODO` segments. Please do not modify the provided `main()` method in `PiPrecisionMpi`. The output produced by the `main()` method will be used to verify the correctness of your solutions.

The next task is to evaluate the performance of the serial and parallel versions, `PiPrecisionSerial` and `PiPrecisionMpi`, on a NOTS compute node. You are required to measure the speedup of your solution (`PiPrecisionMpi`) over the sequential implementation (`PiPrecisionSerial`) on NOTS while using P processes for the following four values and using $precisionDigits = 25,000$:

1. $P = 1$,
2. $P = 2$,
3. $P = 4$ and
4. $P = 8$.

Note: *As in Lab 10, you will only be able to run your code on NOTS. It likely will not run locally.* Local execution is not supported as this homework depends on compiled third-party binaries and a complex development environment that is only available on NOTS. However, you will still be able to compile locally as long as you import the project dependencies from the provided `pom.xml` file.

3.3 Hints

- Passing individual terms to the helper processes from the main process will incur a lot of inter-process communication overhead. You may obtain performance benefits by sending ranges of terms (like chunking) to compute to the helper processes. These ranges can then be distributed among local worker actors to compute the sum of the corresponding terms.
- The `edu.rice.comp322.mpi.MpiUtils` class provides helpful methods to `send()` and `recv()` complex Java objects between the MPI processes. You may try and reduce the overhead in how `BigDecimal` instances are communicated across the processes. If so, you will need to create `send*Message()` and `recv*Message()` methods of your own while developing your solution in `PiPrecisionMpi`. You should, however, be able to reuse the existing `MpiUtils.ResultMessage` class.
- You will notice that there are no unit tests included in the project. The teaching staff will rely on running their reference solution and the sequential `PiPrecisionSerial` and comparing the console output (obtained from `System.out.println()` statements) to verify the correctness of the value of PI computed by your program.

3.4 Grading Rubric

- [**Solution correctness and efficacy (25 points)**] You will be graded on the correctness of your solution while extracting inter-process parallelism using MPI. Your coverage of the various TODOs and issues mentioned in the provided hints should help achieve the maximum points. To get full credit, there should be no assertion failures on each test, and the test should not time out.

- [**Performance evaluation (10 points)**] You should include performance numbers for the four values of P mentioned earlier in the document using $precisionDigits = 25,000$. For your reference purposes, our solution achieved over 3× and over 5× speedup over the sequential `PiPrecisionSerial` on 4 and 8 MPI processes, respectively. You can get these performance numbers using the provided SLURM script under src/main/resources.
- [**Homework report (15 points)**] You should submit a brief report summarizing the design and implementation of your solution, and explain why you believe that the implementation is correct, including why it is free of data races, deadlocks, and livelocks.

Please place the report file named `hw_5_report.pdf` in the top-level `hw_5` directory. Remember to commit all your source code into the subversion repository during your assignment submission.