CX 4220/CSE 6220 Introduction to High Performance Computing
Spring 2024
Programming Assignment 2
Due: March 15, 11:59 PM

1 Matrix transpose with all-to-all communication primitive

Write a C/C++ parallel program to transpose a square matrix using your own implementations of the **all-to-all** communication primitive. You will implement the two algorithms for the all-to-all primitive discussed in class, i.e., one using hypercubic permutations, and the other using arbitrary permutations.

You will then compare the runtime of transposing matrix using your all-to-all implementations and `MPI_Alltoall`. Fixing the number of processors to 8 and 16, plot the runtimes of the three approaches over different matrix sizes and describe your observations of the results. Provide a brief introduction of your code or details specific to your implementations. Contrast the performance of your implementations with the MPI provided implementation.

DO NOT PUBLISH PROJECT SOLUTIONS PUBLICLY.
THIS IS AN ACADEMIC INTEGRITY VIOLATION.

2 Code framework

2.1 Input & Output format

For an $n \times n$ input matrix, the input file to the algorithm only contains the matrix itself. The rows of the matrix are separated by new line and the elements within each row are separated by space. The output file of the program must include the transposed matrix in the same format as the input. The time taken in milliseconds (round to 6 decimal places) must be printed to the terminal using `printf`. You may assume that the number of processors always divides n, and p is always a power of 2.

The program should take 4 command line arguments – input file, output file, all-to-all algorithm choice and matrix size n. You have to structure the program such that one processor (with rank 0) reads the input file and block distribute the rows among all the processors using `MPI_Scatter`. We are doing this for convenience sake, and because we do not have a machine that supports truly parallel I/O. Note that when you time your algorithm, you should start the timer **after** the data has been block distributed and end it before you gather the transposed matrix to write to the output file. Sample input and output files are provided for reference in the Programming Assignments folder on Canvas.

All-to-all algorithm command line arguement:
`a`: Arbritary all-to-all (Implemented by you)
`h`: Hypercubic all-to-all (Implemented by you)
`m`: MPI_Alltoall

Sample command line input:
```
$ mpirun -np 8 ./transpose matrix.txt transpose.txt a 24
```

2.2 All-to-all implementation details

You will be implementing two all-to-all functions, `HPC_Alltoall_H` and `HPC_Alltoall_A` for hypercubic and arbitary permutations respectively. They both should work for `int` datatype and accept the same arguments as `MPI_Alltoall`. You are restricted to using only MPI point-to-point communication. The implementations must follow the algorithms presented in the lecture.

2.3 Deliverables

The programming assignment is to be done in groups of three. It is important that you strictly adhere to the input format described in the programming assignment. No matter how you decide to share the work amongst yourselves, each student is expected to have full knowledge of the submitted program. To submit the programming assignment, turn in the following:

1. Create a Makefile for your program, and make sure the name of your output executable is `transpose`. If you are not familiar with creating Makefiles, check the resources below.

2. Write a `README.txt` briefly describing how your program works and the machine you used for generating the results.

3. Experimentally evaluate and compare the performance of your program with varying input matrix sizes. For $p = 8$ and $p = 16$, plot the run-time of matrix transpose using both of your all-to-all implementations and `MPI_Alltoall` as a function of the problem size.

   Try to come up with some important observations on your program. We are not asking that specific questions be answered but you are expected to think on your own, experiment and show the conclusions along with the evidence that led you to reach the conclusions. Any irregular or unexpected results should also be noted and explained why it is unavoidable. Include your plots and observations in a PDF file with name `report.pdf`.

   Make sure to list names of all your teammates at the very beginning of your report. Provide a brief description of your implementation along with space and run-time analysis, and include a table at the end containing the contributions of each team member.

4. Submit the following:

   (a) All the source files and the Makefile in "Programming Assignment 2 Code" on Gradescope. Do not upload the executable file in your submission.

   (b) The report in "Programming Assignment 2 Report" on Gradescope.

   Only one student needs to make the submission for the whole group. If students of the same group makes two different submissions, there would be a minor penalty applied.

3 Grading (100 pts total)

The program will be graded on correctness and usage of the required parallel programming features. You should also use good programming style and comment your program so that it is understandable. Grading consists of the following components:

1. Correctness (50 pts)

   - Automated tests on Gradescope that will test your program functionality for various inputs including all corner cases.

2. Code and Performance (30 pts)

   - We will go through your code to make sure appropriate parallel programming practices discussed in the class are being followed along with the right MPI functions being called.
   - Specifically, we will make sure that (a) all-to-all functions are utilized efficiently to transpose the input matrix and (b) both all-to-all implementations follows the structure of the algorithms for the hypercubic and arbitary permutations in the slides.
   - We will also check the implementations for:
     - Proper load distribution
     - Efficient data transfer among processes
     - Appropriate uses of only point-to-point communications for the all-to-all implementations

     Note: Points will be deducted for ignoring performance protocols; **Serialization in the program will lead to a zero on the whole assignment. This includes unnecessary serialization of point-to-point communications.**

3. Report (20 pts)

   - The two plots (8 pts)
   - Theoretical analysis for space and time complexities for both all-to-all implementation and the matrix transpose (4 pts)
   - Empirical analysis, observations, and conclusion (8 pts).

4 Resources

1. What is a Makefile and how does it work?: https://opensource.com/article/18/8/what-how-makefile
2. PACE ICE cluster guide: Link here.