CX 4220/CSE 6220 Introduction to High Performance Computing
Spring 2024
Programming Assignment 3
Due: April 19, 11:59 PM

1 Sparse Square Matrix Multiplication

Your friend, George P. Brudell, has to write a program to multiply very large square matrices for his special problems class. He also noticed that their matrices have many zeros in them, making them quite sparse. He decided to store his matrices as an array of (row, col, value) tuples instead of the standard 2D format. But now he's stuck since he does not know how to perform efficient parallel sparse matrix multiplication using this representation of his. He asked you for help since he heard you are taking the HPC class this semester.

Your job is to figure out an efficient parallel algorithm to perform sparse matrix multiplication where the matrices are stored as arrays of (row, col, value) tuples. Give the parallel runtime for your algorithm as a function of the total number of non-zero elements in the matrix.

Then write a C/C++ parallel program to perform sparse matrix multiplication using George's representation and show strong scaling of your implementation for different sized matrices and different sparsity parameters. Compare your experimental observation with the theoretical analysis of your algorithm and explain your findings.

DO NOT PUBLISH PROJECT SOLUTIONS PUBLICLY.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

2 Code framework

2.1 Input & Output format

There is no input file in this project. You'll have to provide an output file if 'print results' parameter is set to 1 in the command line. In that output file, you'll have to print the two matrices you generate and their multiplication result in standard 2D format (not George's sparse format). Sample output files are provided for reference in the Programming Assignments folder on Canvas.

Sample command line argument:
`n`: Dimension of the square matrices is $n \times n$
`s`: Sparsity parameter $\in (0, 1]$
`pf`: Print flag (results) $\in \{0, 1\}$
`out`: Output file name

Sample command line input:

```
$ mpirun -np 8 ./spmat 10000 0.001 0 spmat_out
```

2.2 Implementation details

Let's assume you are supposed to perform the matrix multiplication of $A \times B = C$ in a p processor parallel computer.

- Write a sparse matrix generator that generates (n/p) rows of matrix A and B using a random number of generator and that assigns 0 to the (i, j)-th element of the matrix with Probability $(1 - s)$ [s is the sparsity parameter given in the command line].
- The runtime of the sparse matrix multiplication algorithm must depend upon the number of non-zero elements in the matrix and must be faster than the dense matrix multiplication algorithms presented in class. Please strictly adhere to George's data structure and do not represent the matrix in any other form, except for the output matrix.
- Each element of the matrix will be a 64 bit unsigned integer. To make things simple, you are allowed to store the output of the matrix multiplication in dense 2D format (using $n \times n$ entries), but both the inputs must be stored and processed in George's sparse format.
- Transpose the sparse B matrix such that each processor now stores n/p columns locally (in the same sparse format) using `MPI_Alltoallv` collective operation.
- Your program must embed a ring topology for all the p processors using the MPI's embedding functions, and use the ring topology for the algorithm.
- Rotate the B matrix using that ring topology such that all rows of A matrix can perform the dot products with all the columns of B matrix to compute the corresponding elements of the resulting C matrix.
- For simplicity, we are storing the C matrix in dense 2D array format in $1D$ block distributed fashion. That means, the first processor stores first n/p rows (n^2/p elements) of C, the second processor stores next n/p rows, $\cdots$.
- **(Bonus points)** Come up with your own data structure and algorithm for sparse matrix multiplication. Explain your data structure, algorithm, and it's performance in the report. You are free to implement your algorithm however way you find it useful, as long as it uses no extra library (other than standard C++ STL library) and written purely in MPI.

- When the print flag 1 in command line, you are supposed to print matrix A, matrix B and then matrix C, (all in dense 2D array format) in the filename given in the command with an empty line after one another.

2.3 Deliverables

The programming assignment is to be done in groups of three. It is important that you strictly adhere to the command line arguments and the output 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 `spmat`. 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 $p = 16$, plot the run-time of sparse matrix multiplication using three different n (> 1000) parameters by keeping $e = 0.01$ fixed. Next, plot the runtime of your program by fixing $n \geq 10000$ and changing $e = 0.1, 0.01, 0.001$ using $p = 2, 4, 8, 16$ processors.

4. Theoretically analyze the runtime of your parallel algorithm and discuss if the experimental results supports your analysis or not. Provide explanations if your experimental performance doesn't match your theoretical analysis.

5. For bonus points, your own data structure and implementation must be $\geq 2\times$ faster than George's original algorithm. Any solution that deliberately writes a slow implementation of the original algorithm will be negatively penalized.

   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.

6. Submit the following:

   (a) All the source files and the Makefile in "Programming Assignment 3 Code" on Gradescope. Do not upload the executable file in your submission.

(b) The report in "Programming Assignment 3 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 + 30 bonus pts)

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 (40 pts)
   - Automated tests on Gradescope that will test your program functionality for various inputs including all corner cases.
2. Code and Performance (40 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) you are generating the matrices properly in George's sparse format based on the command line arguments, (b) your implementations follows the structure of the algorithm mentioned and format mentioned previously.
   - We will also check the implementations for:
     - No serialization in your implementation
     - Efficient data transfer among processes
     - Appropriate use of MPI collectives and/or point-to-point operations
     - Proper use of ring topology

     Note: Points will be deducted for ignoring performance protocols; **Serialization in the program will lead to a zero on the whole assignment. This includes unnescessary serialization of point-to-point communications.**
3. Report (20 pts)
   - Explain the time and space complexities of your implementation (4 pts)
   - The mentioned plots (8 pts)
   - Empirical analysis, observations, and conclusion (8 pts)
4. **Bonus (30 pts)**
   - Perform the same experiments of your own algorithm and data structure for sparse matrix multiplication and describe it in the report. Explain why your new algorithm is faster than George's original plan.

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.