CPSC 424/524 Spring 2020
Assignment 4

Due Date: Tuesday, April 14, 2020 by 9:00 a.m.

Triangular Matrix Multiplication

The Problem. Given a lower triangular *N*×*N* double precision matrix A ($a_{ij} = 0$, for $j > i$) and an upper triangular *N*×*N* matrix B ($b_{ij} = 0$, for $j < i$), develop an MPI program to compute the product of A and B.

Mathematically, you are computing an *N*×*N* matrix C whose entries are defined by:

$$c_{ij} = \sum_{k=1}^{\min(i,j)} a_{ik} b_{kj} \qquad (1)$$

Task 1: Serial Program (5 points)

I have provided source files (`serial.c & matmul.c`) for a serial program that computes *C* using equation (1). These files are in the directory `/home/cpsc424_ahs3/assignments/assignment4`. You may copy and use all files in that directory. (However, please do not copy the `.dat` files, since they are rather large; just use them where they are.) *You are free to modify or replace the C program, if you wish. If you do so, you may use any data structures you like, but you may store and use* ***only the potentially nonzero elements of the matrices****. That means you may not store entries of the strict upper triangle of A or the strict lower triangle of B. If your modifications require significantly more space or time than the original, you may be docked some points for inefficiency.*

Run the serial program on randomly-generated triangular matrices using *N* = 1000; 2000; 4000; and 8000. The `serial.c` program contains code (a seed setting) to reproducibly create randomly-generated triangular matrices. Please use that code for this purpose throughout this assignment to facilitate checking the numerical results. Along with the source files, I have provided binary result files containing the correct C matrices, and `serial.c` computes the Frobenius norm of the error in the computed C, which should be zero (or very nearly so). All your codes should illustrate performance and correctness by printing a table similar to the one printed by `serial.c`. (To save disk space, please do not copy the binary result files; simply read them where they are, as done in `serial.c`.) For timing, you may use my `timing()` functions from earlier assignments; see the `Makefile`. Alternatively, you may use standard MPI timing routines `MPI_WTime()`. Below is a sample output table from `serial.c`. (Your timings may not match mine.) See the `Makefile` and the file `build-run.sh` for a list of relevant `#SBATCH` settings for Slurm, required module files, and how to build the serial program.

```
Matrix multiplication times:
   N      TIME (secs)     F-norm of Error
 -----   -------------   -----------------
   1000        0.1650     0.0000000000
   2000        1.6968     0.0000000000
   4000       17.9664     0.0000000000
   8000      144.1152     0.0000000000
```

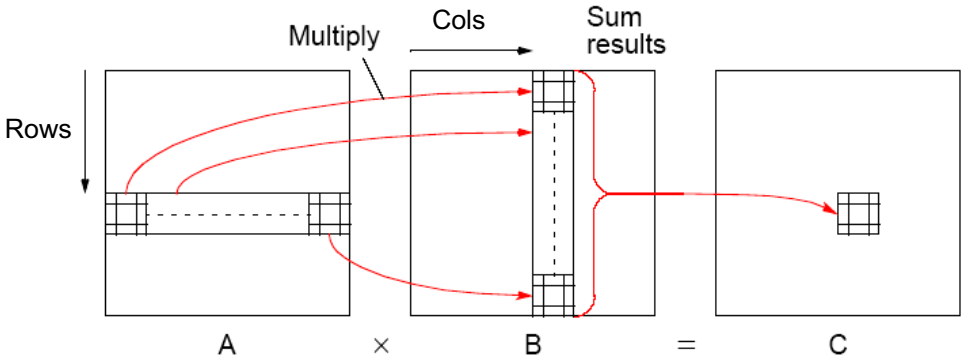
NOTES:

(a) Each job run for this assignment should need no more than ~30 minutes. Please limit your walltime requests so that everyone has fair access to the class partition. Please debug using *N* = 1000 before trying larger cases. My own experience suggests that you may see variation in timings, so make several runs and report average timings.

(b) As you develop parallel codes, ensure that your programs produce the correct answers by having the master compute and print error norms as in `serial.c`. (However, DO NOT include this work in your reported timings.)

Tasks 2-4: Parallel Programs

Parallelization Approach: Here's the approach you'll use for this assignment: Divide the input matrices into blocks of rows (for A) or columns (for B). Then a computational task would be to compute the product of a block row of A and a block column of B to form a rectangular block of C. Pictorially (shown for full, not triangular, matrices A and B), this looks like:



To implement this using p MPI processes (including the master, which should do real work in addition to managing the entire computation in this case), you will use the following "ring-pass" approach:

1. The master (rank 0) process creates space for the full matrices and initializes A and B as in `serial.c`.
2. The master then partitions A and C into p block rows each, and it partitions B into p block columns.
3. Next, the master permanently assigns each MPI process (including the master) one block row each of A and C, and it assigns each MPI process one block column of B as its initial assignment. The master should use MPI collective operations to distribute the assigned block rows or columns of A and B to each MPI process. (Each MPI process will need to allocate memory space for its blocks of A, B, and C.)
4. The computation then proceeds iteratively as follows:
    a. Each MPI process does all the computation it can with the data it has in hand.
    b. MPI processes then pass their block columns of B to the next higher-ranked MPI process. (MPI process p-1 passes to MPI process 0.)
    c. Repeat until done with the calculation.
5. The master then uses MPI collective operations to assemble the full C matrix by collecting all the block rows of C from the other MPI processes.
6. Finally, the master computes the error norm and prints a table similar to the one printed by `serial.c`.

Notes: You will use various MPI functions in this assignment, but you may not use `MPI_Sendrecv()`. Except for Task 5, you may assume that N is a multiple of p. For timings, allocate entire nodes.

Task 2: Blocking MPI Parallel Program (40 points for CPSC 424; 30 points for CPSC 524)

Part A (25 points for CPSC 424; 15 points for CPSC 524): Implement an MPI program that computes the product of two triangular matrices using equation (1) and the parallelization approach described above. **For Task 2, you must use MPI's blocking communication operations (including collectives).** Your program should use exactly p block rows for A, and p block columns for B, where p is the total number of MPI processes (including the master). **For Task 2, each block row or block column should contain N/p consecutive rows or columns.** Insert timing calls to help understand and critique the performance of your program. (Specifically, try to (a) assess load balance by comparing the times for each MPI process; and (b) distinguish between time spent computing and time spent communicating for some of the processes.) Run your program on randomly-generated triangular matrices using N = 1000; 2000; 4000; and 8000, using p = 1, 2, 4, and 8 on one node for each value of N. For each run print a summary timing/error table similar to that in the serial program. (Do not report the breakdown of the computation vs. communication times for this part of Task 2, but you will need to do that in the additional test cases below.) For this part of Task 2, in your report, please discuss the raw performance, scalability, and load balance you observe. Include per-process timing data sufficient to support your discussion of load balance.

Part B (15 points): For $p = 4$ and 8 *only*, using $N = 8000$ *only*, make additional runs using 2 and 4 nodes with appropriate numbers of MPI processes per node. (You will have to tell Slurm and/or `mpiexec` how to distribute processes on available cores. In this case, please use a *round-robin assignment across the sockets*. To learn how to do this, see the man pages for `sbatch` and `mpiexec`.) Provide the summary table for each run, and, in addition, provide tables that show the timing breakdowns for computation and communication in your program when run on 1, 2, and 4 nodes. (You should already have results for 1 node from Part A.) Discuss the raw performance and load balance you observe, including specific comments based on the timings for computation vs. communication. Based on your observations in both parts of Task 2, suggest (**but do not implement**) ways to improve the program (including raw performance, scalability, load balance, and timing splits between computation and communication). In formulating these suggestions, you may consider use of any MPI operations except `MPI_Sendrecv()`, and you may suggest variations in the number and sizes of row- and column-blocks, or modifications to the communication pattern, if you wish. **You MUST describe why and how you expect each of your suggestions to improve program performance, scalability, and/or load balance.**

Task 3: Non-Blocking MPI Parallel Program (40 points for CPSC 424; 35 points for CPSC 524)

Part A (25 points for CPSC 424; 20 points for CPSC 524): One way to improve performance might be to overlap computation and communication. Modify your Task-2 program to try to implement overlapping by using non-blocking MPI communication operations where possible. You should continue to use collective operations as in Task 2. (Hint: Think about the steps in the program that might be overlapped. You may need to introduce additional communication buffers to implement certain types of overlap.) Once you have a working program, repeat the runs and discussions/analyses from Part A of Task 2.

Part B (15 points): Repeat the runs and discussions/analyses from Part B of Task 2 using your modified program. Compare the results to your Task-2 results, and try to explain what you observe. Be sure to address differences in raw performance, scalability, and load balance, and to discuss the observed breakdowns between computation and communication times. (Note that you may or may not see large differences, depending on your code.)

Task 4: Load Balance (15 points)

If the observed load balance in your Task-3 program was not as good you'd have liked, then suggest and implement one ***simple*** approach that might improve the load balance, while retaining similar levels of raw performance and scalability. For this task, you may modify the parallelization approach described above, if you feel that doing so will achieve better results. (Hint: Follow the KISS principle. Don't expect to achieve perfect load balance. Just come up with, describe, justify, and implement one ***simple*** strategy that tries to make a significant improvement. If you believe that your load balance in Task 3 was good enough, you need not make new runs—in that case, all you need to do here for full credit is to explain how your Task-3 program achieved such good load balance.)

If you do create a modified program for this task, then run it on the test cases from Part B of Task 3 (and also on the corresponding test cases on 1 node), and produce similar tabular outputs. Briefly discuss/explain the performance differences you see, including raw performance, scalability, and load balance. **Note: You are only expected to try out one reasonable strategy, which may or may not actually produce improvements.** If your strategy didn't work, you need not implement a different one, but you must try to explain why it didn't do what you had hoped.

Task 5: Generalization (15 points). Required for CPSC 524; Extra credit for CPSC 424

Modify your Task-4 program (or your Task-3 program if you didn't modify it for Task 4) so that it works for arbitrary N and p (if you wish, with the presumption that $p \leq 32 << N$). **For this task only, N is not presumed to be a multiple of p.** Demonstrate that your code works by running it for $N = 7633$ and $p = 7$ on 4 nodes, with a round-robin MPI process distribution by socket. (Note that I have provided a correct answer for $N = 7633$.)

Procedures for Programming Assignments

As usual, we will use the Canvas website to submit solutions to programming assignments.

Remember: While you may discuss the assignment with me, a ULA, or your classmates, the source code you turn in must be yours alone and should not represent collaborations or the ideas of others!

What should you include in your solution package?

1. **All source code files, Makefiles, and scripts that you developed, used, or modified.** All source code files should contain proper attributions and suitable comments to explain your code. For each task or major subtasks (e.g., parts A or B of Tasks 2 and 3), please create a build-run script similar to the ones I have provided. Each script should load the proper module files, build the code(s) using a suitable makefile, and run the required tests. The script outputs should document what was run and what the timing and error-check results were.
2. **A report in PDF format** containing:
    a. Your name, the assignment number, and course name/number.
    b. Information on building and running the code:
        i. A brief description of the software/development environment used. For example, you should list the module files you've loaded.
        ii. Description of the steps/commands used to compile, link, and run the submitted code. Best is to reference the makefiles and build-run scripts in the files you submit.
        iii. Outputs from executing your program (probably also included in the files you submit).
    c. Any other information required for the assignment, including any questions you were asked to answer.

How should you submit your solution?

1. On the cluster, create a directory named "**NetID_ps4_cpsc424**". (For me, that would be "**ahs3_ps4_cpsc424**". Put into it all the files you need to submit.
2. Create a compressed tar file of your directory by running the following in its parent directory:

```
tar -cvzf NetID_ps4_cpsc424.tar.gz NetID_ps4_cpsc424
```

3. To submit your solution, click on the "Assignments" button on the Canvas website and select this assignment from the list. Then click on the "Submit Assignment" button and upload your solution file **NetID_ps4_cpsc424.tar.gz**. You may add additional text comments to your submission, but your report should be included as an attachment. You can use scp or rsync or various GUI tools to move files back and forth to Grace.

Due Date and Late Policy

Due Date: **Tuesday, April 14, 2020 by 9:00 a.m.**

Late Policy: On time submission: Full credit

Up to 24 hours late: 90% credit

Up to 72 hours late: 75% credit

Up to 1 week late: 50% credit

More than 1 week late: 35% credit

General Statement on Collaboration

Unless instructed otherwise, all submitted assignments must be your own individual work. Neither copied work nor work done in groups will be permitted or accepted without prior permission.

However….

You may discuss questions about assignments and course material with anyone. You may always consult with the instructor or ULAs on any course-related matter. You may seek general advice and debugging assistance from fellow students, the instructor, ULAs, Piazza conversations, and Internet sites.

However, except when instructed otherwise, the work you submit (e.g., source code, reports, etc.) must be entirely your own. If you do benefit substantially from outside assistance, then you must acknowledge the source and nature of the assistance. (In rare instances, your grade for the work may be adjusted appropriately.) It is acceptable and encouraged to use outside resources to *inform* your approach to a problem, but *plagiarism is unacceptable* in any form and under any circumstances. If discovered, plagiarism will lead to adverse consequences for all involved.

DO NOT UNDER ANY CIRCUMSTANCES COPY ANOTHER PERSON'S CODE—to do so is a clear violation of ethical/academic standards that, when discovered, will be referred to the appropriate university authorities for disciplinary action. Modifying code to conceal copying only compounds the offense.