CS211 HW3: Sieving Prime

Due: 11/21/2024 23:59

Before Start:

1. Click the following link to accept your assignment on GitHub Classroom:

   https://classroom.github.com/a/oT-dsfpF

   Clone the template code to your workspace on the **hpc-001** cluster (refer to the attached guide: **CS211 HPC Cluster.pdf** for instructions on how to log in). The core functions for this assignment are implemented in **sieve0.c, sieve1.c, sieve2.c, sieve3.c.** Your task is to complete the code in these files to ensure the project works correctly.

2. To debug your code, you can run the following command, which will test your code on the header node.

   - **module load gcc/gcc-5.1.0**
   - **module load mpich-3.2.1/gcc-4.8.5**
   - **make**
   - **mpirun -np {cores} ./main {function} 10000000000**

   The **{cores}** variable can be set to any of the following values: 32, 64, 96, 128, or 160, and **{function}** variable can be sieve0, sieve1, sieve2, or sieve3. If your output shows count=455052511, this indicates that your algorithm is functioning correctly.

   Please note that running code on the header node is significantly slower than submitting jobs through SLURM, so results obtained on the header node will not count towards your final grade. However, running code on the header node can be more flexible, making it useful for debugging.

3. To evaluate performance on the compute node, please use **starter.py** to compile and test your code. Each time you run **starter.py**, the script will submit all tests (sieve0 to sieve3) to SLURM. If you want to test specific cases rather than the entire algorithm, you can modify the following list variables:

   - **N_list** specifies the number of nodes to use in the test, with a maximum of 5 nodes supported on our cluster.
   - **ver_list** defines the sieving algorithms to test. For example, to test only sieve3.c, set ver_list = [3].

You can find your execution results in **.o** files under the **sh** directory.

4. In this assignment, each node is configured to run 32 processes; please do not modify this setting. Additionally, we only find prime numbers below 10^{10}, so please keep $n = 10^{10}$ unchanged for all your algorithms.

5. When running your code on the compute node, if it runs for more than 5 minutes without producing any output in the **.o** files, it may be stuck in a deadlock. In such conditions, please use **scancel -u $USER** to terminate all your submitted jobs.

6. We will use **main.c** to measure running time and performance. Please do not modify the **main.c** file when submitting your code.

7. This assignment requires multi-node parallel computation, which must be performed on the **hpc-001** cluster to access the necessary computational resources.

Submission:

Following the instructions in the questions below, complete the corresponding code or theoretical analysis and answer the questions. Collect experiment data and compare the different optimization strategies when analyzing your results. You need to submit a **PDF** report for this assignment, and your code must be submitted on GitHub Classroom (push your final code to your repository in GitHub Classroom). To better align your report with your code, please upload a copy of your report to your GitHub repository as well.

Report requirements: Do not include the problem description in your report; the report should only contain answers to the questions, the corresponding experimental data, data analysis, and the theoretical reasoning process.

Report naming format: **cs211_hw3_firstname_lastname.pdf**. Please include the link to your GitHub repository, your name, and your SID on the first page of your report.

Q1 (10 points)

Run the parallel Sieve of Eratosthenes program provided in the slides (which can be found in sieve0) and record the execution results.

Q2 (30 points)

Modify the parallel Sieve of Eratosthenes program from class (located in sieve0) so that the program does not allocate memory for even integers. Place your updated code in sieve1.

Q3 (30 points)

Modify sieve1 so that each process in the program finds its own sieving primes through local computations instead of using broadcasts. Place your updated code in sieve2.

Q4 (30 points)

Modify sieve2 so that the program makes effective use of caches. Place your updated code in sieve3.

For each version of the sieve, record the execution time and the number of primes found when $n = 10^{10}$ and the number of cores is set to 32, 64, 96, 128, and 160 (corresponding to 1, 2, 3, 4, and 5 nodes). Check if the execution time is inversely proportional to the number of cores. Additionally, compare the execution times of each version of your program and calculate the achieved speedup.