CSCI-4320/6360 - Assignment 2: Parallel C Program for a Carry-Lookahead Adder Using MPI

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590

February 1, 2019

Deadline: Friday, Feb 15th, 2019 at 11:59:59 p.m.

1 Overview

Here, you are to re-use your serial Carry Lookahead Adder and adapt it to use 32 bit blocks and extend it to work in parallel for a 1M (e.g., 1,048,576) bit CLA adder with 32 bit blocks using up to 32 MPI ranks on the class server, `mastiff.cs.rpi.edu`.

This CLA adder can be constructed from the following. *Note, the description has been updated to consider that the i-th carry at any level is really the carry-out from that bit position and that $i-1$ is the carry-in to the i-th bit, group or section, etc bit position.*

Note, to make the CLA program run in serial and parallel using 2, 4, 8, 16, and 32 MPI ranks, you can think of "slicing" the CLA adder into 32 equal size chunks. **So, you will only need to exchange messages at the ssc_l level**. That is each rank will have only one "bit" of carry information at that level.

1. Calculate g_i and p_i for all 1,048,576 i.
2. Calculate gg_j and gp_j for all 32,768 j using g_i and p_i.
3. Calculate sg_k and sp_k for all 1,024 k using gg_j and gp_j.
4. Calculate ssg_l and ssp_l for all 32 l using sg_k and sp_k.
5. Calculate ssc_l using ssg_l and ssp_l for all l and correct ssc_o. *Note, for 32 MPI ranks, Rank 0 will send carry message updates to rank 1, and rank 1 to rank 2 and so on up to rank 31. If you have fewer than 32 ranks, then each rank will perform the necessary calculations multiple ssc_l. For example at 8 ranks, there are 4 ssc_l, at 4 ranks there are 8 ssc_l and at 2 ranks there are 8 s^3c_l and the serial processor case computes all 32 ssc_l.*

6. Calculate sc_k using sg_k and sp_k for all k and replacing each sc_{k-1} when $k \; mod \; 32 = 0$ using correct $s^2c_{l-1}, l = k \; div \; 32$ as sectional carry-in for all k in that block of 32. Note, make sure you set $sc_{k-1} = s^2c_{l-1}$ when $k \; mod \; 32 = 0$ and $l = k/32$.

7. Calculate gc_j using gg_j, gp_j for all j and replacing each gc_{j-1} when $j \; mod \; 32 = 0$ using correct $sc_{k-1}, k = j/32$ as sectional carry-in for all j in that block of 32. Note, make sure you set $gc_{j-1} = sc_{k-1}$ when $j \; mod \; 32 = 0$ and $j = i/32$.

8. Calculate c_i using g_i, p_i for all i and replacing each c_{i-1} when $i \; mod \; 32 = 0$ using correct gc_{j-1}. Note, make sure you set $c_{i-1} = gc_{j-1}$ when $i \; mod \; 32 = 0$ and $j = i/32$ for the final sum step below.

9. Calculate sum_i using $a_i \oplus b_i \oplus c_{i-1}$ for all i.

More specifically, you will do the following:

1. Like before represent the 1,048,576 bit input numbers as 262,144 hex digits.

2. Have MPI Rank 0 read the input data, perform the conversion from hex to binary and **reverse** the binary input numbers such that the most significant bit is in the highest position within the binary array. **See the example "mpi-cla-io.c" for reading the input data and writing the final result that is posted on the website.**

3. Have MPI Rank 0 distribute the input binary arrays to each rank in the correct order where (for a 32 ranks configuration) MPI rank 1 has bits 32,768 through 65,535 and MPI rank 2 has bits 65536 through 98304 and so on. This should be done using `MPI_Scatter`.

4. Like before, write functions for each step in the above algorithm. You can do it using `for` loops and do not have to perform the equation substitutions by hand as we did in class.

5. Between each algorithm step perform an `MPI_Barrier` collective operation to keep all ranks in step with each other. Make the barrier operation conditional as you will want to turn it own and off as part of your performance study.

6. Use `MPI_Isend` and `MPI_Irecv` routines to exchange the nearest-neighbor carry bit information at the ssc_l level. Again, Rank 0 will send to rank 1 and rank 1 will receive from 0 and send to rank 2 and so on. Note that Rank 0 will only send and rank 32 will only receive carry information. **Note, all ranks except Rank 0, should post an `MPI_Irecv` message before the cla calculations start well in advance of any `MPI_Isend` messages being scheduled.**

7. A master "cla" routine will run through all the steps.

8. You main routine will invoke your input reading function, followed by your master "cla" function. You can check your answer by having Rank 0 read in the full data set hex input data (convert it to binary) and like before using the ripple carry adder based on computing the $c_i = g_i + p_i * c_{i-1}$ for all i and then computing the final sum, based on $sum_i = a_i \oplus b_i \oplus c_{i-1}$.

9. Have each rank send there part of the final sum_i solution to Rank 0. This should be done using `MPI_Gather`. Rank 0 will then re-reverse the sum and output the final result. (Yes, this output result will consume several pages of text).

10. For the performance study, measure the execution of serial and parallel runs using `MPI_Wtime` which returns a double. Here you'll have a `start_time` and `finish_time` and their difference is the execution. Performance testing should be done on the class server, *mastiff.cs.rpi.edu*. Exclude the reading data from STDIN or writing data to STDOUT from your timing results.

11. Next, create the following three graphs: First, plot the execution time of 2, 4, 8, 16 and 32 rank runs as function of their number of ranks. Next, plot of the speedup relative to the execution time of the serial MPI CLA adder of the 2, 4, 8, 16 and 32 rank runs and finally plot the speedup of the relative to the execution time of the serial ripple carry adder to the MPI CLA adder running in parallel on 2 thru 32 ranks. Each graph should have two plots line - one with and one without the `MPI_Barrier` operations between each step. Explain why your performance graphs turned out they way they did. This report must be written using LaTeX, MSWord other document preparation software and handed in in PDF format using `submitty` along with your code.

12. Test case will be posted on the website and run using file redirection to standard input. E.g., the command line that will be used is:

    mpirun -np X ./cla test1.txt test1-output.txt

    Note, that X is the number of ranks that will be used. We will run our tests on 4 or 8 ranks, depending on the test but again your performance tests should run on 2, 4, 8, 16 and 32 MPI ranks using *mastiff.cs.rpi.edu*.

2 HAND-IN INSTRUCTIONS

Submit both your code and PDF report to `submitty.cs.rpi.edu`.