James Hahn
CS1645 High Performance Computing
Dr. Jaber Hasbestan
Homework #2

Custom Collective

In the custom collective, the algorithm from homework 1 section 3 was implemented. This is a binary tree collective communication function.

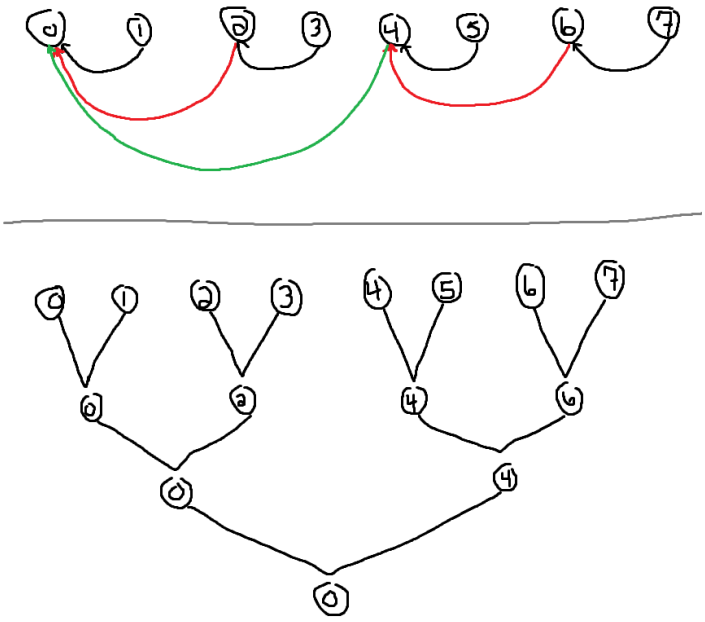

Figure 1
Demonstrating the binary tree collective and its general flow.

Essentially, let's assume the program is being ran with 8 processors. Each processor has a "parent" node. In the above case, for the first iteration of summations, processor 1 sends to processor 0, 3 sends to 2, 5 sends to 4, and 7 sends to 6. Then, in the next iteration, processor 2 sends to processor 0, and then 6 to 4. Finally, processor 4 sends its summation to processor 0. In the end, processor 0 possesses all summations from all processors because they either sent their 'chunk' summation of the integral directly or indirectly to processor 0.

So, since the integral is calculated on the interval from 0 to 1, with 8 processors, each processor computes the integral on some interval of length 1 / 8 = .125 . So, processor 0 would compute the integral on the interval [0, .125], processor 1 computes [.125, .250], and so on until processor 7 computes [.875, 1]; each interval is referred to as a chunk. Additionally, the user passes in a value **n** to determine the how granular each chunk will be. For example, if there are 8 chunks, but the user specifies **n = 32**, then each chunk will contain 32 / 8 = 4 parts. So, each processor

will further break down their interval. For example, processor 0 will compute the integral over intervals [0, .03125], [.03125, .0625], [.0625, .09375], and [.09375, .125], further increasing the accuracy of our results.

MPI Native Collective

MPI has very useful functionality. It provides the MPI_Reduce function. With this function, each processor will perform its own work and then send its work to one processor, such as the root, and MPI will perform some operation on all the data form each processor, such as a summation, average, or more. In the case of this project, each processor computed the integral over its respective chunk, then called MPI_Reduce, sent its summation to processor 0 (root) with the MPI_SUM (sum all the data, or sum all the summations) operation. In the end, processor 0 now contains the sum of all summations of the integral chunks, stored in an integral variable. Then, all processor 0 has to do is output the integral value to the user.

Results

All results were computed for different values of n, resulting in higher accuracy in the computation of pi. A higher number of processors increases the speed of computation, but the higher value of n increases the accuracy of the integral. Times were given for both the MPI native collective and the custom collective.

All times were gathered by using Unix's bash time command and the sys time was pulled. Note that as the number of processors increases, the time increases. It is important to note that although the sys time increases, the time per processor decreases, decreasing the overall running time of the program

1 processor

n	**MPI Native Collective time (seconds)**	**Custom collective time (seconds)**
8	0.050	0.053
16	0.050	0.054
32	0.049	0.056
64	0.052	0.057
128	0.050	0.058

2 processors

n	**MPI Native Collective time (seconds)**	**Custom collective time (seconds)**
8	0.072	0.073
16	0.071	0.074
32	0.074	0.079
64	0.075	0.080
128	0.076	0.082

4 processors

n	**MPI Native Collective time (seconds)**	**Custom collective time (seconds)**
8	0.115	0.120
16	0.115	0.123
32	0.117	0.125
64	0.125	0.139
128	0.128	0.142

8 processors

n	**MPI Native Collective time (seconds)**	**Custom collective time (seconds)**
8	0.222	0.242
16	0.235	0.246
32	0.246	0.253
64	0.249	0.256
128	0.255	0.261

Additionally, as n increases, the accuracy of the integral increases. This is shown in the below table. The values converge toward pi with a decimal precision of 9 digits.

n	**Integral value**
8	3.13898849
16	3.14094161
32	3.14142989
64	3.14155196
128	3.14158248