MPI: Practical 3 Solutions

1 Gaussian quadrature

1.1 Reduction operator

This should be straight-forward. See `quadrature_reduce.cpp`.

1.1.1 Speed

On apollo, I got the following times:

Processes	Time (s)
1	2.836
2	1.546
4	0.862
8	0.534
16	0.416

These are fairly good speed-ups, as the calculation of individual terms (and local sums) is completely independent. Speed-up at higher core-counts is less due to the overhead of launching the processes.

1.1.2 Accuracy

This test is fairly robust to numbers of processors. However:

- On 13 processors (using 16 s.f.):

  ```
  Grand total is 2456621.125977768 and integral is 0.886226957423437
  ```

- On 19 processors:

  ```
  Grand total is 2456621.125977737 and integral is 0.8862269574234262
  ```

This is due to the order in which the terms are summed (finite-precision arithmetic is not associative). If this is a for your code, you will need to think more carefully about how the terms are summed. The use of Kahan summation may be appropriate, but will not solve all problems. Again, this is not a bug in MPI, but in the algorithm.

1.1.3 Modifications

- You will need to change the `0` in the `MPI_Reduce`call to `nproc-1`, and then output `if (rank == nproc-1)`
- To make the result availble on all processors, use `MPI_Allreduce` or `MPI_Reduce` followed by an `MPI_Bcast`
- Use of `MPI_FLOAT`, etc. instead of `MPI_DOUBLE` will give nonsense results. MPI does **not** have type checking; you must make sure you get the correct one.

2 Gather

See `quadrature_gather.cpp`.

3 Send/Receive

See `quadrature_sendrecv.cpp`.

4 Extra: Tree-based sum

See `quadrature_parallel_tree.cpp`. It may help to draw the reduction for, say, 8 processors, and then for 10 processors, so that you correctly identify the case where processes do not expect to receive data from a higher-ranked process.