KTH-id: olieri
Github: https://github.com/raaavioli/MPI-Assignments-DD2356

Exercise 1 - MPI Hello World

1. *How do you compile it, which compiler and flags have you used if any?*
   Local: mpicc hello-mpi.c -o hello-mpi.out
   Beskow: cc hello-mpi.c -o hello-mpi.out

   No particular flags.

2. *How do you run the MPI code on Beskow?*
   Allocation: salloc --nodes=1 -A edu21.DD2356 -t 00:05:00 -C Haswell
   Run: srun -n 8 ./hello-mpi.out

3. *How do you change the number of MPI processes?*
   srun -n X ./hello-mpi.out, where X determines the number of MPI processes.

4. *Which functions do you use for retrieving the rank of an MPI process and the total number of processes?*

Number of processes:	MPI_Comm_size(MPI_COMM_WORLD, &size)
Rank:	MPI_Comm_rank(MPI_COMM_WORLD, &rank)

5. *What are the names of the most used MPI implementations?*
   LAM-MPI and MPICH. [1]

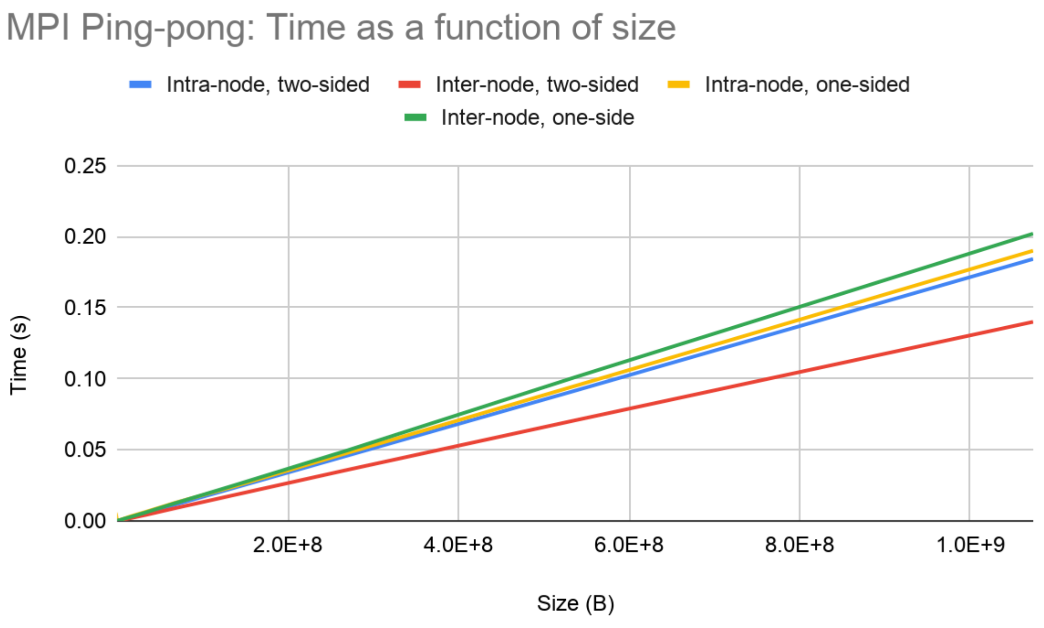
Exercise 2 - 1D Domain Decomposition with Blocking Communication

1. *Test the results by checking the correct values are on the ghost cells and the derivative of sin(x) on the edges of the domain is correct (the derivative of sin(x) is cos(x)).*
   - Show code and execute program.

2. *Why are `MPI_Send` and `MPI_Recv` called "blocking "communication?*
   Because they block communication and will not return until the provided buffers are properly sent or received. This means particularly that any buffer sent or received can safely be used immediately after MPI_Send or MPI_Recv invocations. Moreover, MPI will deadlock if receives and sends are not done in a proper order. Let’s say p0 sends something to p1, meanwhile p1 waits to receive a message from p2, but p2 is not sending any message. In this scenario, both p0 and p1 will deadlock, and neither of them will be able to continue execution and depend on p2.

Exercise 3 - Measure Network Bandwidth and Latency on Beskow with Ping-Pong

1. *Run the ping-pong benchmark for inter- and intra-node communication.*



2. *Using best fit (using Matlab, Python, or similar), calculate the bandwidth and latency for 1) and 2).*
   Notice: No matter which values I sampled, the latency regression would behave very strangely. For some measurements, the larger times caused latency to become negative, and for some measurements, the smaller times caused negative latency. Seems like this regression does not say anything. For example, for the intra-node data, if any of the data points for size > 32MB was included, latency would be negative, but for *inter* the result would be negative if including any result where size < 256B.

	Bandwidth (Gb/s)	Latency (µs)
Intra,two	6.100733501	-0.01542181422
Inter,two	7.976541156	0.03649780112

Calculated latency and bandwidth from pingpong benchmark. Measured values for size < 256B was excluded. Both having non realistic latencies due to statistical variation. Expected would be 0.7µs for intra and 1.6µs for inter-node communication on Beskow.

3. *What is the difference between one-sided communication and point-to-point (two-sided) communication?*
   One sided communication implies that only one side initiates the transaction of data. The target side does not explicitly have to receive the data. In one sided communication, the target side shares a block of memory (a window), where any other rank in the same group can write or read. In two sided communication on the other hand, both the origin and the target have to act to complete a transaction. In two-sided communication, we are also implicitly talking about a connection between only two ranks at a time.

4. *Calculate the latency and bandwidth using the one-sided ping-pong test. Compare and discuss the difference of performance between MPI one-sided and point-to-point communication.*

   The results show that two-sided inter-node communication is the fastest. I did not expect this as I thought intra-node communication would be faster. Moreover, I thought one-sided communication would be faster than two-sided communication. However, as stated in the lectures, one-sided communication MPI implementations can vary in quality, which may cause the two-sided rates to dominate.

   When calculating latencies, I could not sample any fixed set of sizes so that the latencies of all benchmarks turned positive. In the example below, message sizes >= 256B was used, causing intra-two sided, intra-one sided and inter-one sided communication to have negative intersection points for latency.

   The error is due to statistical variation between the tests, and to gain an accurate value for the latency and bandwidth, one would have to perform more tests and calculate the average, making sure that the standard deviation is about 1-2% of the execution time.

	Bandwidth (Gb/s)	Latency (μs)
Intra,two	6.100733501	-0.01542181422
Inter,two	7.976541156	0.03649780112
Intra,one	5.19622928	0.01010811119
Inter,one	5.603423492	-0.00974010295

5. *Read [Modeling MPI Communication Performance on SMP Nodes: Is it Time to Retire the Ping Pong Test (Links to an external site.)] paper and answer the following questions:*

1. *Why is the postal model not the best performance model for communication?*
   Because it does not take the interface between the nodes into account. That is, the model assumes any two processes' execution are independent of each other. The communication is in fact bound by the network interface rates for inter-node communication, and context switches and internal overheads for intra-node communication, which is not considered with the ping pong test. Also, implementations of MPI handle messages of different length differently, which may add some overhead depending on the message length.

2. *Which performance model is proposed?*

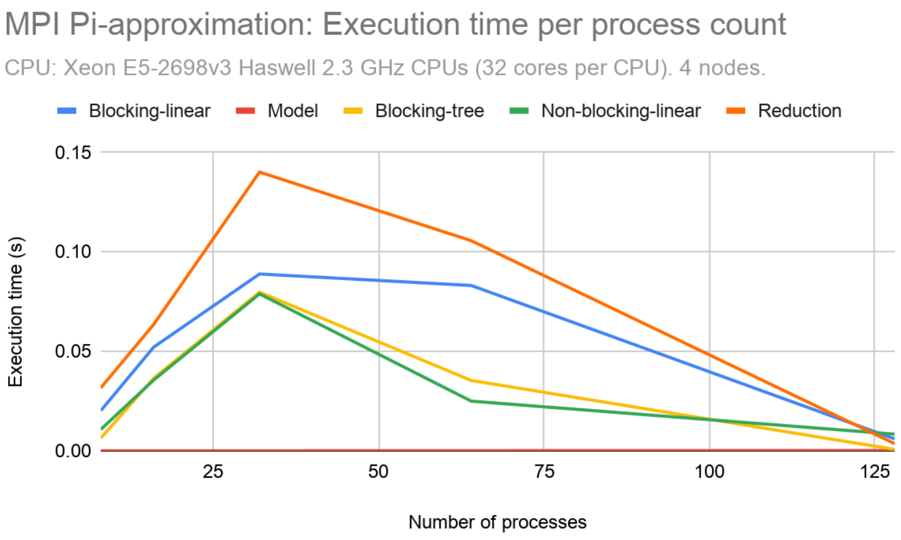
$$T = \frac{kn}{\min\left(R_N, \frac{kn}{\alpha + n/R_c}\right)}. \qquad (5)$$

   The proposed performance model takes into account Rc, and Rn, as well as k. Rc is the rate that each of the k processes individually can achieve when sending and receiving data. Rn is the rate that the network can provide the node.

   What the model actually says is that if the network bandwidth is small, then the execution time is inversely proportional to the network bandwidth. If the network bandwidth is high, then the ordinary ping pong model works. Especially, for very large n, the authors state that alpha is negligible. This can be seen from calculating the latencies in task 4. When including data points for large message sizes, the latency behaves oddly, which is an indication that large n takes away the importance of latency.

4.1 MPI Blocking Communication & Linear Reduction Algorithm

1. *Measure execution time for 8, 16, 32, 64, 128 MPI processes and plot it. How does the execution time scale with the number of processes? What is the MPI function for timing?*



The MPI-function for timing is MPI_Wtime, and the results show that the execution time is approximately halved when doubling the number of processes.

The execution time scales approximately linearly, however with a small constant up to 32 processes. After that it drastically increases in performance. I have no idea what is happening, but Beskow is onto something really really weird right now. The configuration does not behave as I would expect. At least, the time for 128 processes approaches the theoretical value. Data shown in table below.

Processes	Time	Stdev	Model
8	0.0202279	0.04008462961	0.0000112
16	0.0521535	0.04176498341	0.000024
32	0.0889128	0.1643866607	0.0000496
64	0.0831366	0.1139727282	0.0001008
128	0.0060677	0.00536737128	0.0002032

Table illustrates actual timings on beskow running linear implementation timings on the communication model.

2. *Develop a performance model for the execution time of the application using the postal communication model for the network performance. Use the values of bandwidth and latency you found in exercise 2. Compare the results from the measurements with the performance model results.*

   latency = 1.6μs (from assignment description due to negative latency

bandwidth = 7.977 GB/s (from exercise 2 for inter-node communication)
n = 4 (One integer sent per message)
msgs = #messages sent = #processes - 1 (all processes except 0)

T = msgs (latency + (1 / bandwidth) * n)
Since n / bandwidth << latency, the resulting time can be approximated as:
T = msgs * latency, which was used

4.2 MPI Blocking Communication & Binary Tree Reduction Communication Algorithm

1. *Measure the performance of the code (execution time) for 8, 16, 32, 64, 128, (possibly 256) MPI processes and plot it*
   See graph above

2. *Develop a performance model for the execution time of the application using the postal communication model for the network performance. Use the values of bandwidth and latency you found in exercise 2. Compare the results from the measurements with the performance model results.*
   The model is way faster, like for exercise 4.1 with linear blocking.

   In theory, the binary tree reduction model would behave like the linear model, except that some messages can be sent in parallel. For example, all odd-ranked processes will pass their message to the smallest neighboring even-rank process. And for each of the following even processes, the messages will be passed to the next process in binary bit-wise order. So, process 6 (binary 110) would pass its data to process 4, (100), etc…

   This would essentially mean that at any stage, we could parallelize half of the processes, which means that we would get log2(size) stages.

   So, a reasonable performance model would be:

   T = log2(msgs) * latency.

3. *How does the performance of binary tree reduction compare to the performance of linear reduction?*
   See graph above: Performs slightly better than blocking linear for all numbers of threads.

4. *Increasing the number of processes - let's say 10,000 processes - which approach (linear/tree) is going to perform better? Why? Think about the communication performance model.*
   Tree, since the linear model forces blocking waits between each received message, so that messages will be sent one by one to process 0. In the tree model, message transmission costs are grouped together and messages are sent in parallel to different processes.

4.3 MPI Non-Blocking Communication & Linear Reduction Algorithm

1. *Measure the performance of the code (execution time) for 8, 16, 32, 64, 128 (and possibly 256) MPI processes and plot it*
   See figure above.

2. *What are the MPI functions for non-blocking communication? What does the "I" in front of the function name stand for in non-blocking communication?*
   The I stands for immediate(ly), implying execution will continue immediately after issue and not block the thread. Non-blocking communication means that we tell the thread that we are expecting to receive some result, but we don’t want to block our thread from performing other tasks meanwhile. Then it is up to the thread to say when it wants to synchronize and make sure that the result is present using for example MPI_Wait/waitall.

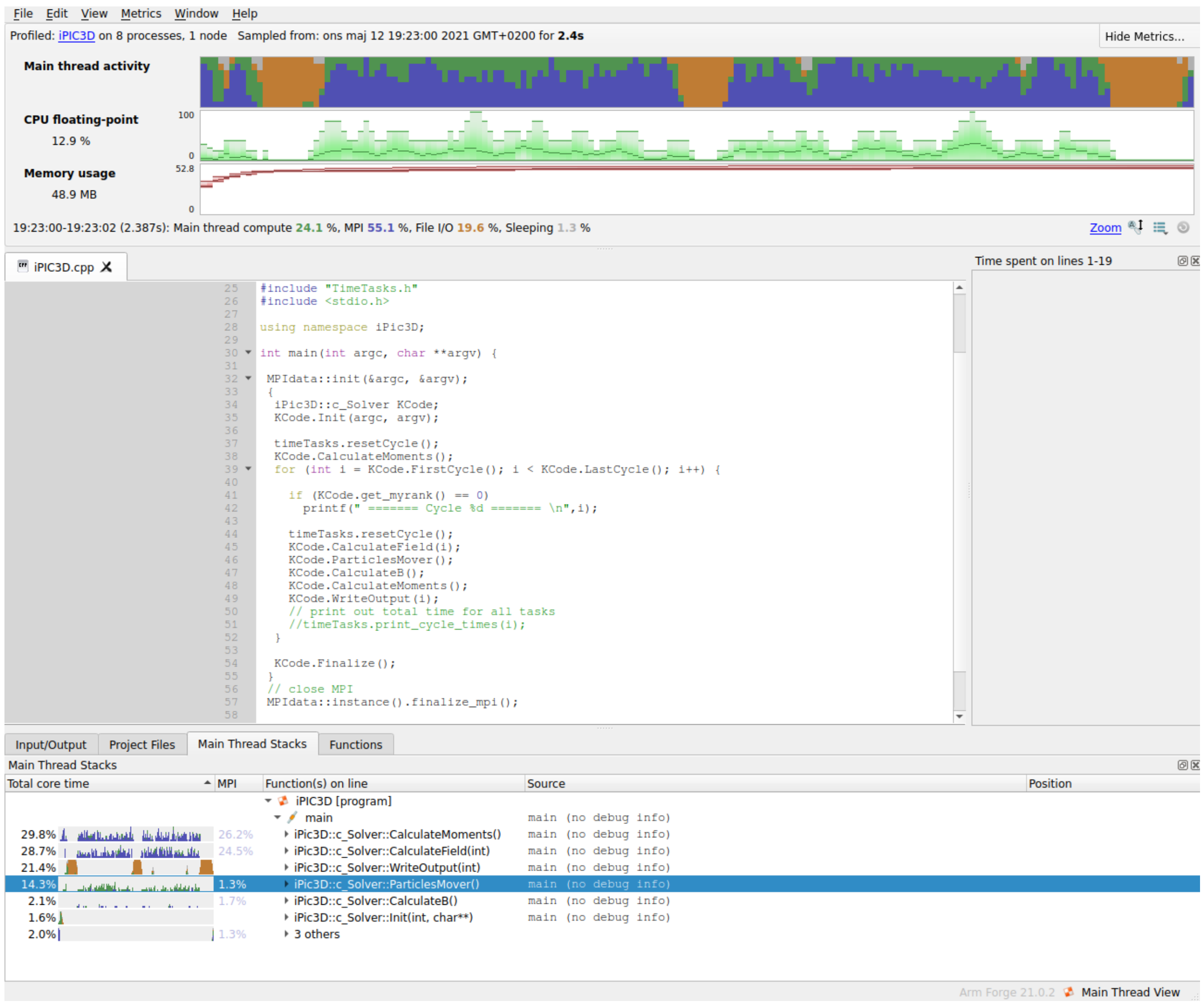
3. *How the performance of non-blocking communication compares to the performance of blocking communication. e.g. performance results in 4.1?*
   Compared to blocking communication, it performs slightly better, which is to be expected. Performs similarly to blocking tree-implementation.

4.4 MPI Collective: MPI_Reduce

1. *Measure the performance of the code (execution time) for 8, 16, 32, 64, 128 (and possibly 256) MPI processes and plot it.*
   See graph above. MPI_Reduce Performs worse than all others when measuring communication execution time. Not expected, should be fast.

Exercise 5 - iPIC3D Performance Monitoring with MAP

1. *By running MAP with iPIC3D, generate the .map traces. Move the `.map` file to your laptop or workstation and visualize it using the MAP viewer. Take a screenshot of the visualization of the MAP results and add it to the report.*



Profiled: iPIC3D on 8 processes, 1 node Sampled from: ons maj 12 19:23:00 2021 GMT+0200 for **2.4s**

19:23:00-19:23:02 (2.387s): Main thread compute 24.1 %, MPI 55.1 %, File I/O 19.6 %, Sleeping 1.3 %

```
#include "TimeTasks.h"
#include <stdio.h>

using namespace iPic3D;

int main(int argc, char **argv) {

 MPIdata::init(&argc, &argv);
 {
  iPic3D::c_Solver KCode;
  KCode.Init(argc, argv);

  timeTasks.resetCycle();
  KCode.CalculateMoments();
  for (int i = KCode.FirstCycle(); i < KCode.LastCycle(); i++) {

    if (KCode.get_myrank() == 0)
      printf(" ======= Cycle %d ======= \n",i);

    timeTasks.resetCycle();
    KCode.CalculateField(i);
    KCode.ParticlesMover();
    KCode.CalculateB();
    KCode.CalculateMoments();
    KCode.WriteOutput(i);
    // print out total time for all tasks
    //timeTasks.print_cycle_times(i);
  }

  KCode.Finalize();
 }
 // close MPI
 MPIdata::instance().finalize_mpi();
```

Main Thread Stacks

Total core time	MPI	Function(s) on line	Source	Position
		iPIC3D [program]		
		main	main (no debug info)	
29.8%	26.2%	iPic3D::c_Solver::CalculateMoments()	main (no debug info)	
28.7%	24.5%	iPic3D::c_Solver::CalculateField(int)	main (no debug info)	
21.4%		iPic3D::c_Solver::WriteOutput(int)	main (no debug info)	
14.3%	1.3%	iPic3D::c_Solver::ParticlesMover()	main (no debug info)	
2.1%	1.7%	iPic3D::c_Solver::CalculateB()	main (no debug info)	
1.6%		iPic3D::c_Solver::Init(int, char**)	main (no debug info)	
2.0%	1.3%	3 others		

2. *What are the iPIC3D functions that take most of the time? Check the MAP breakdown of different activities*
   CalculateMoments (26.2%) and **CalculateField(int)** (24.5%)

3. *What is the recorded bandwidth of MPI Sent and Received?*
   Main thread:
   Sent: 23.8MB/s
   Received: 0.60GB/s

   Any thread max:
   Sent: 62.7 MB/s (rank 1)
   Received: 8.22 GB/s (rank 1)

   During heavy IO-operations (reading/writing to disc most likely) then Sent and Received are zero.

References

1. Vinter, Brian & BJØRNDALEN, John & ANSHUS, Otto & LARSEN, Tore. (2010). A Comparison of Three MPI Implementations. 62.