This assignment covers the Message Passing Interface (MPI).

1 Amdahl's Law (1P)

The SpeedUp Sp of an parallel application running with p processes in comparison to a serial execution can be formalized through Amdahl's Law:

$$Sp = \frac{1}{s + (1 - s)/p} \tag{1}$$

$s \in [0, 1]$ represents the serial portion of the application (often we have parts that cannot be parallelized). Given Sp we can compute the parallel efficiency $Eff = Sp/p$ of a certain application.

1. You were able to prove that the sequential portion of a solver for linear systems of equations is $s = 10\%$. Determine the maximum number of processes p if you want to achieve a parallel efficiency of at least 70%.

2. Amdahl's law is kind of pessimistic. Please explain this property w.r.t. to the maximum number of processes p. Are there other laws in order to classify the parallel behaviour of an application?

2 Probing the Network (3P)

In this task we examine the Omni-Path network of the Linux-Cluster in different settings.

1. Do a literature research on the Omni-Path in general and the Omni-Path network used in the Linux Cluster. Explain why the bandwidth depends on the message size. What latencies and peak bandwidths can you expect?

2. Cluster support gives you the following information about the fabric:

   > CoolMUC3 has a fat tree topology with 2:1 pruning. Each of the 148 nodes has two 100 Gbit links to the fabric. Each of the 10 leaf switches has 48 ports, where 32 ports are connected to the nodes and 16 ports are connected to the spine switches (4 ports are connected to each of the 4 spine switches).

   Draw a picture of the topology using the supplied information. (That is, a graph showing all compute nodes, switches, and links. The picture should clarify the concept, i.e. you do not need to draw every individual compute node.)

3. Compute the bisection width and bisection bandwidth of 16 nodes and of 32 nodes w.r.t. the topology. Assume that 16 nodes are connected to each leaf switch.

   Bisection width: Minimum amount of edges (i.e. links, cables) that have to be removed to separate the network into two equal parts (i.e. same number of nodes in each part).

   Bisection bandwidth: Maximum bandwidth over the bisection line, i.e. the sum of the single bandwidth of all edges that are cut by bisecting the network.

4. Give a short explanation of the sub-benchmarks in the *Intel MPI Benchmarks* [1, 2].

5. Run the *Intel MPI Benchmarks* on the Linux Cluster using different number of nodes. Provide plots and interpretations for the *Single Transfer Benchmarks*, the *Parallel Transfer Benchmarks*, and the *Collective Benchmarks*.

3 Broadcast (3P)

1. Implement a method *broadcast* using MPI, which sends an array containing $\mathtt{n}$ `doubles` from root process 0 to all other processes within the ring.

2. Study three different algorithms:

   - `trivial`: root-process sends the array to all other $\mathtt{p}-1$ processes.
   - `tree`: data is send according to a tree structure. First process 0 sends the data to process $\mathtt{p}/2$. Afterwards, both, process 0 and process $\mathtt{p}/2$, repeat these calls in a recursive manner until all processes hold the data.
   - `bonus`: each process sends at most $\mathsf{O}(\mathtt{n})$ of `doubles`. In addition, also on the critical path only $\mathsf{O}(\mathtt{n})$ elements are sent.

3. Derive for all of your implementations the number of required MPI-messages and the amount of data sent on the critical path. In addition, measure the achieved bandwidth (bytes/time) w.r.t. to message size of your entire broadcast routine and compare these results to MPI's built-in routine `MPI_Bcast`.

Remark: How do you measure the bandwidth of a broadcast?

Assume we have only two peers and we send $\mathtt{n}$ doubles, then the bandwidth is readily determined as array size / time. However, for a broadcast this question is more delicate. A possibility would be to count the total communication volume and divide it by time, e.g. for the trivial algorithm, we have a total communication volume of $(\mathtt{p}-1)\cdot\mathtt{n}$ doubles. This would yield a value which could be interpreted as "collective network bandwidth". The problem with this approach is, that in order to improve the speed of the broadcast we might actually increase the communication volume as it is done in a tree algorithm. Hence, when comparing the bandwidth of the trivial algorithm with the tree algorithm, we would compare apples and oranges (i.e. a higher bandwidth does not imply a faster broadcast). Furthermore, for `MPI_Bcast` we cannot necessarily determine the communication volume as we do not know the algorithm that is chosen in a particular MPI implementation.

As a remedy we may take the following point of view: Think of the perfect network with M nodes, where every node has a dedicated wire to each other node (M links per node). Furthermore, every node can send to M nodes simultaneously with the full single-link bandwidth $\mathtt{B}$. Here, the time required by the trivial algorithm is array size / $\mathtt{B}$.

For the non-perfect network, we also define the bandwidth as array size / time. However, here the interpretation is not in terms of collective network bandwidth but it measures how close our bandwidth is to that of a perfect network.

The good thing about this definition is, that it is easy to apply and it allows for a fair comparison between algorithms. You just have to be careful with the interpretation.

4 Parallel CG (3P)

The conjugate gradient method (CG-method) is an iterative method for solving symmetric positive definite linear systems of equations given as $\mathtt{A}\mathtt{x} = \mathtt{b}$. Here, we are just solving the Laplacian as discussed during the lecture. `poisson.cpp` provides a serial implementation of the CG-method.

1. Parallelize this application using MPI.
   Hints: For best results sub-divide the grid into equal-sized patches (e.g. 2×2, 3×3 or 3×4, ...). According to this distribution, please create a virtual MPI communicator topology. During execution, processes need to exchange non-compact data; you should use `MPI_Type_vector` for this task.

2. After you have a working version (you can verify your code by comparing results with the sequential version and by plotting results with gnuplot), please perform a detailed performance analysis study featuring different grid sizes, different process-grids, as well as speed-ups and parallel efficiencies.

Deliverables

The following deliverables have to be handed in no later than 08:00 AM, Monday, 4 December 2017. If there is no submission until this deadline, the exercise sheet is graded with 0 points. Small files (<1 MB in total) can be send as an attachment directly to *uphoff AT in.tum.de* and *chaulio.ferreira AT tum.de*. Larger files have to be uploaded at a place of your choice, e.g. `https://gitlab.lrz.de/`, `http://home.in.tum.de/`, `https://syncandshare.lrz.de`. In either case inform us about the final state of your solution via e-mail.

- A short report which describes your work and answers all questions in this assignment.
- All of your code.
- Slides for the presentation during the next meeting.
- Output of all runs. Figures (e.g. scaling graphs) if applicable.
- Documentation how to build and use your code.

Literature

[1] Intel MPI benchmarks. https://software.intel.com/en-us/articles/intel-mpi-benchmarks/. Accessed: 2017-11-10.

[2] Intel MPI benchmarks user guide. https://software.intel.com/en-us/imb-user-guide. Accessed: 2017-11-10.

[3] Intel MPI library documentation. https://software.intel.com/en-us/articles/intel-mpi-library-documentation/. Accessed: 2017-11-10.

[4] MPI 3.0 standard. http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf. Accessed: 2017-11-10.

[5] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994. URL: http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.