University of Basel

Lecturer:
Prof. Dr. Florina M. Ciorba

Computer Lab. U1.001
Spiegelgasse 1,
CH-4051 Basel

Assistants:
Aurélien Cavelan, Ph.D.
Danilo Guerrera, M.Sc.
Ahmed Eleliemy, M.Sc.
Ali Mohammed, M.Sc.

http://informatik.unibas.ch/fs2018/lecture-high-performance-computing/

High Performance Computing (17164-02) **Spring Semester 2018**

Assignment 3: Advanced MPI *(20 Points)*

Starting Date: April 05, 2018
Deadline: April 18, 2018 - 23:59:59

Objectives:
1. Parallelize the execution of the programs using MPI + OpenMP.
2. Explore the effect of the scheduling in achieving load balance.
3. Explore the performance of MPI one sided communication.

1 Matrix Multiplication MPI+ OpenMP *(8 Points)*

Starting from the naive implementation of the matrix multiply program provided in *T1.c*, parallelize the matrix multiplication code using MPI + OpenMP.

a. Optimize and parallelize the source code for the matrix multiply using MPI. *(5 Points)*

b. Use OpenMP pragmas for fine grain parallelization. *(1 Point)*

c. Compare the performance of the implemented version with MPI+OpenMP parallelization with the version implemented in the previous assignment with OpenMP parallelization only for `numThreads = 2,4,8,16` for the OpenMP version, and `MPI ranks = 2, 4`, with `numThreads = 2, 4, 8, 16` for the MPI+ OpenMP hybrid version, and `matrix size = 4000`. *(2 Points)*

2 Load balancing in MPI *(7 Points)*

As you have learned in the previous assignment, performance of parallel applications suffer due to load imbalance. In this task, you will work with sparse matrix multiplication. You are required to distribute the work among the MPI ranks in order to achieve load balance.

a. Starting from the code in *T2.c*, modify the *fill_matrix* to fill only the upper triangle of the matrix with double values and the rest of the matrix with zeros. *(1 Point)*

b. Parallelize the code provided using MPI (do not use OpenMP). Work should be distributed equally among the MPI ranks in a cyclic fashion, where every rank is assigned a block of iterations = number of iterations/number of ranks. Report the performance of executing the code with 40 MPI ranks (20 rank per node, using 2 compute nodes) and `matrix size = 4000`. *(2 Points)*

c. Modify the code implemented in the previous step to distribute the work in a block cyclic fashion, where the block size can be changed to achieve a balanced better performance . *(3 Points)*

d. Change the block size to achieve the best performance for 40 MPI ranks on 2 nodes, and matrix size of 4000. Report the best achievable performance and compare it with the performance in step (b). *(1 Point)*

3 One Sided Communication *(5 Points)*

Implement a matrix multiplication program that uses the one sided communication functions (MPI_Get and MPI_Put). Use the provided code template *T3.c* to do this task. In *T3.c*, you will find some hits to help you in this task.

a. Use the one sided communication functions to divide rows of Matrix A. *(2 Points)*

b. Use the one sided communication functions to share Matrix B and then each MPI rank should multiply its part of A with B. *(2 Points)*

c. Let this code report its total execution time. Report the performance for 40 MPI ranks on 2 nodes, and matrix size of 4000. *(1 Point)*

Please make sure that your optimizations do not affect the correctness of the results.
You should use the miniHPC cluster to obtain all the results.
Please make sure that you use the Intel environment (Intel compiler and Intel MPI) by running `ml intel` on the miniHPC cluster.
Please make sure that you use `export I_MPI_FABRICS=shm:tmi` to achieve best performance.
The delivered solution should be in one tar file.