Universität Heidelberg
Institut für Technische Informatik

Holger Fröning
Benjamin Baumann

Introduction to High Performance Computing
Term 2014/2015 (Winter)

Exercise 2

- **Return electronically until Tuesday, 4.11.2014 14:00**
- **Include name on the top sheet. Stick several sheets together.**
- **A maximum of two students is allowed to work jointly on the exercises.**

Prerequisite

- Login: introhpcXX
    - password: changeme
- Login using ssh:
    - From extern: ssh {username}@door.ziti.uni-heidelberg.de
        - Be aware that multiple wrong attempts will block the IP for 24hrs
    - From door or internally: ssh creek[01-08]
    - It is highly recommended to set up ssh keys. Use Google if unsure about details.
- Change password immediately:
    - log to lsra01 (changing password only possible on lsra01)

```
$ ssh lsra01
$ passwd
```

2.1 MPI Ring Communication

- Write an MPI program for **n** processes, that exchanges **m** messages per process in the following way: *process i sends m messages to process ((i+1) % n)*. Include time measurement functions to report the total execution time and the average time per message. Ensure that your program cannot run into a deadlock.
- Execute the program on the nodes creek[01-08] with up to 24 processes total. Choose **m** in a way that the average time per message is stable (several consecutive runs should report similar results). Report latencies for 2, 4, 6,…, 24 processes. Create a graph with your results.
- Minimize the average time per message for 12 processes by optimizing the mapping. Explain why this mapping is optimal for this program.

(10+5+5=20 points)

2.2 Barrier Synchronization

- Now we want to measure the performance of an MPI barrier. First implement your own barrier, either a centralized barrier or a tree based barrier. It's your choice, however a centralized barrier might be easier to implement. Your choice won't have an impact on achievable points.
- Write a suitable test program to measure the performance of your own barrier implementation. For this purpose call the barrier within a for-loop and measure the overall execution time of the loop. The number of iterations should be passed to the program as a command line parameter. Use a sufficient number of iterations to get stable results. Report average barrier latencies (loop execution time divided by the number of iterations) for 2,4,6,...,24 processes.

- Do the same measurement with the MPI built-in barrier. Compare the performance and plot both results in one graph.

(15+10+5=30 points)

2.3 Matrix multiply – sequential version

- Implement a naïve – i.e. non-optimized – sequential version of the matrix multiply operation. Multiply two double-precision floating point matrices. Initialize the matrices using random values. Use appropriate time measurement functions like *gettimeofday()* or *clock()* to measure the execution of the multiply operation itself (i.e., without initialization or output).
- Execute the program on one idle node (one of creek[01-08]). Report the execution time and the achieved GFLOP/s for a matrix multiply of the size 2048x2048 elements.
- Explain the huge gap between achieved GFLOP/s and theoretical peak GFLOP/s, in particular which subsystem of this computer is the bottleneck for the execution of this program.
- Optimize your program and overcome the locality problem of this program, explain and implement your idea. Execute the improved version, measure again execution time and achieved GFLOP/s.

(10+5+5+10=30 points)

Total: 80 points