Parallel Computing for Partial Differential Equations — Matthias K. Gobbert
Sommersemester 2012 — Universität Kassel
Homework 3 — due on Wednesday, April 25, 2012

1. [2 points.]

   (a) Familiarize yourself with the textbook. The purpose of this part is to determine what is available where in the book. I recommend the following simple, but effective strategy: Start flipping at the beginning of the book and note what types of things appear (such as Table of Contents, Preface, etc.) until you reach the first regular chapter. Then flip from the very back of the book and note what appears (such as Index, appendices, etc.) until you reach the last regular chapter. Report what you find.

   (b) Locate the homepage of the textbook at `http://www.cs.usfca.edu/~peter/ppmpi` (notice correction against the Preface of the textbook). Obtain the errata for yourself. Determine which printing of the textbook you are looking at; this will be useful to know when considering the list of errata. Obtain the source code for all examples in the textbook for future use.

   (c) Read the Preface and Chapter 1. Then read Chapters 3 and 4 that are relevant for this homework. You should read Chapter 5 for the next homework; that chapter is in many ways the heart of MPI, because a lot of useful work can be accomplished just by the material from that chapter.

   There will be no more precise reading assignments for the following homeworks. See the syllabus for the chapter coverage and read in the textbook as needed. At the maximum, I consider useful the Chapters 6, 7, 10, 13, but only portions of Chapters 8, 9, 11, 12, and 15. Appendix A is eventually extremely useful and you should know about it.

2. [6 points.] Stefan Kopecz has assembled a webpage with some English instructions on using the cluster. This page is presently located at `http://www.mathematik.uni-kassel.de/~kopecz/cluster.html`, whose link is also provided from the course webpage's detailed schedule. If it changes, I will post a new link in the detailed schedule page. Go to this page, and do parts (a) and (b) below using information in the compile and run tutorials; this is a standard way to try out that your account on such a system is working correctly. In more detail:

   (a) Download the sample program `hello_parallel.c` and compile it using `mpicc`; to control the name of the resulting output file (the name of the executable), use the `-o` option, so use the compile command

   ```
   mpicc hello_parallel.c -o hello_parallel
   ```

   The name of the executable is important, because the supplied submission script assumes this executable name.

   (b) Run the code now with

   ```
   sbatch mpihello.slurm
   ```

   Notice that stdout is retained in the file slurm.out and stderr is retained in the file slurm.err. You should now look through both slurm.err and slurm.out. What happens and is it correct?

   (c) Experiment with different choices of values `X` and `Y` in the line

   ```
   #SBATCH --nodes=X
   #SBATCH --tasks-per-node=Y
   ```

   in the submission script. Which values for them are legal on the nodes of the cluster that are using? Report your experiences.

3. [12 points.] Consider the program for the trapezoidal rule in Chapter 4 of Pacheco.

   (a) Obtain the code, run it, and verify that it works properly for you. Discuss any issues that you feel are short-comings of this code and how to fix them.

(b) This and the following parts of this problem are dedicated to applying some obvious fixes to Pacheco's original code.

Improve the output of the program by also outputting the true value of the integral, the true error, the quantity h^2, the step size h, number of intervals n, and the number of processes p. These numbers can be used to check that the code is performing properly. For optimal readability, organize each output in a separate line and align the numbers by proper formatting. Make sure to choose a suitable format conversion for all numbers; why did you choose the format conversion that you did choose?

(c) Program command-line input for the pertinent problem parameters a, b, and n. Notice that a and b are real numbers, while n is an integer.

(d) Install at least one error check to learn the use of the `MPI_Abort` function. In this context, consider n an input, whose validity needs to be checked. For example, if $n < 0$, then print an informative message from Process 0 and use the `MPI_Abort` function to abort with a non-zero error code.

(e) Run your code with as many nodes as available and with different numbers of processes per node. Present your results in one table of the form of Table 1.1 of the tech. report HPCF–2010–2, that is, the rows reflect the number of processes per node and the columns the number of nodes. (The case of 6 processes per node is a special case for this tech. report only; you do not need that case.) Discuss the performance you observe. Remember to check the output of the code as set up earlier in the assignment to ensure that the code is computing correct results for all cases of the performance study: *Code that computes incorrect answers is never acceptable!* Implement improvements to the code as needed and explain what you did.

Two notes on performance studies: (i) One big problem in parallel computing is communication time. Therefore, only wall clock time is an honest measure of whether your code performs well. This can simply be accomplished by using the MPI command `MPI_Wtime`, which saves a current time stamp in seconds. Notice that the time as such is meaningless (see its man-page) and that only differences of two such time stamps can be relied upon. To get the most conservative time, it is necessary to wait for the slowest process to complete its job, hence it is customary to force a synchronization of all processes with the `MPI_Barrier` command. Thus, the outline of code looks like

```
MPI_Barrier(MPI_COMM_WORLD);
startTime = MPI_Wtime();
/* ... code to time here ... */
MPI_Barrier(MPI_COMM_WORLD);
endTime = MPI_Wtime();
if (id == 0) { /* output from Process 0 only */
  tsec = endTime - startTime;
  printf ("elapsed wall clock time in seconds = %9.2f\n", tsec);
}
```

which prints the difference of the start and end time to `stdout`. (The above `printf` is just a starting point for your coding; you should work on the formatting of the output to get useful data for all cases.)

(ii) A practical aspect is how to organize the large number of studies that you want to do in a way that ensures that results do not overwrite earlier ones and such that you have a record saved in case of future questions. I have found that the only safe way is to have one directory for each run, which contains the job submission script and all other necessary files for each run. The only difference from one directory to the next are the job submission script's entries for `nodes` and `ppn`. A naming scheme for theses directories might thus be `nXXppnYY`, where `XX` = 01, 02, 04, etc. denotes the number of nodes and `YY` = 01, 02, 04, etc. the number of processes per node.

What to submit at minimum as part of your report: (1) There should be *one* final version of code; include it in your report as listing. (2) Include the contents of one `slurm.out` in your report that shows the output of your code for $n = 8192$ in its formatted form. (3) Include a table of observed wall clock times in the format of a summary table (rows for processes per node, columns for number of nodes, as Tables 1.1 and 1.2 in the tech. report HPCF–2010–2). (4) Discuss the changes you made to the code and whether your final code is optimal.