DD2356 Software Project
Dense Linear Algebra with OpenMP and MPI

Corrected version: 2017-05-21

Deadline for hand-in: Monday, 2017-06-05

The project aims at the study of parallel algorithms in dense linear algebra. You will implement and analyse different algorithms of Gaussian elimination with MPI and OpenMP.

Recommended literature for the project is the book of Rauber and Rünger "Parallel Computing" that is referenced in the course literature.

The project work consists of programming activities and a report that documents your findings. The following six steps will guide you through the project.

Steps resulting in grades E, D, C, B

1. Study the description of Gauss elimination in the book of Rauber and Rünger (section 8.1.1). Add a short summary of this section to your report.
2. Write a complete MPI program for Gaussian elimination with a row-cyclic data distribution. It is recommended that you study the section 8.1.2 in the book and derive your implementation from the algorithm described there.
   Hint: The text gives descriptions of operations within the algorithm that can be used to implement functions that serve as building blocks for the implementation.
3. Now, implement a MPI program with a total cyclic data description. The algorithm is explained in detail in section 8.1.3 of the book.
4. Measure and compare the run-time of the algorithms you implemented in steps 2 and 3 for different matrix sizes and numbers of processors. Document and explain the observations.
5. Implement a Gaussian elimination algorithm for shared memory systems using OpenMP. You could start from the previously used algorithm with a row-cyclic data distribution. In the report, you need to explain how you express the parallelism in the implementation. Explain necessary synchronisations in order to make access of shared data safe too. Measure and explain the run-time of the algorithm you implemented.

Additional step necessary for grade A

6. Develop a MPI program for the Gaussian elimination that uses a column-cyclic data distribution. The implementation in step 2 used row-cyclic data distribution. Explain in your report how the communication within the algorithm has to be changed for the column-cyclic data distribution and implement the algorithm. Measure and explain the run-time of the algorithm with different matrix sizes and processor numbers, also in comparison to the two implementations done earlier.

Requirements on the project work

Deliverables: Implementation of the algorithms and report.

Implementation of the algorithms

- Your programs need to be provided via KTH Github. Create a project on https://gits-15.sys.kth.se/ and add your project supervisor as member to it.
- It must be possible to clone this project to the cluster Tegnér and to build it with GNU make.

Requirements in detail are:

- Each program (algorithm) shall be located within a sub-directory of the project's root that is named according to the step of the project work, i.e. "step_2", "step_3" etc. The name of the executable must be accordingly "step_2.x" etc. This executable must be located in the respective sub-directory too.
- The makefile you create shall be located in the projects root directory and must be able to build all executables. (target "all" or no target on the command line of the make activation). There shall be also targets available for every executable "step_2.x" etc.
- You need to provide a bash shell script, named "setenv", which loads all needed environment modules and makes other settings that you maybe need on Tegnér in order to run the executable. This script will be used with the shell command "source" in order to make necessary environment definitions before the compilation as well as program execution.
- The command line of your programs shall accept the following options
  ```
  -i <input file>
  -o <output file>
  -g <p1>x<p2>
  ```
  The input file contains the matrix that shall be processed by the algorithm for Gaussian elimination. The output file shall contain the solution vector after the Gaussian elimination has been applied to the input matrix (see description below). The option –g specifies how the cores are used for the decomposition. The matrix has a size n by n. The option –g specifies how many cores shall be used along the matrix rows (p1) and along the columns (p2). The product p1 times p2 is the total number of cores used in a run. For example, –g 5x1 could be used for a row-cyclic data distribution, -g 1x5 for a column-cyclic distribution and –g 3x2 for a total-cyclic distribution where each core gets data pieces from n/3 rows and n/2 columns.
- Output files shall have the same names like input files, but with the suffix ".out". In the case of MPI programs, the input data shall be read respective the output data shall be written by process rank 0 only. You need to distribute data respective collect the results internally in your program.
- The data format to use in the matrix files is the so called "Matrix Market" format. You find a description online at http://math.nist.gov/MatrixMarket/ Routines that

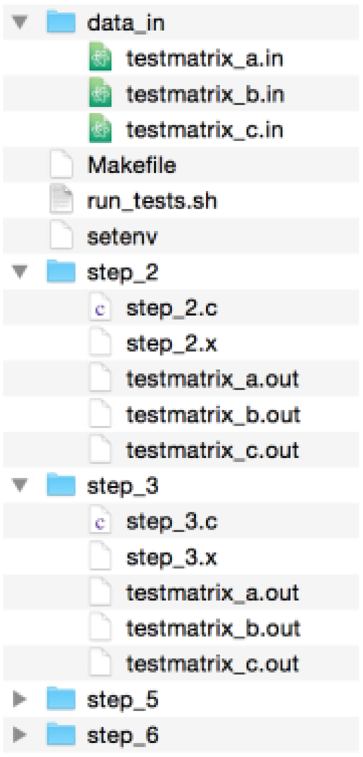
provide implementations for reading and writing such files as well as an example in the programming language C can be downloaded from http://math.nist.gov/MatrixMarket/mmio-c.html

- Finally, you need to provide a shell script that can be used to run tests of all your implementations with all test matrices that will be provided to you on eight processors resp. threads. This script shall be located in the root directory of the project and must execute all tests. The output data shall be placed into the sub-directory of the executable under test. The script shall run on a single node that has been allocated on Tegnér with the command *salloc*. Make sure that all necessary modules are loaded and that it is taken care of other, maybe necessary environment definitions.

  An example invocation of a test run from this script could look for example as follows:

  ```
  mpiexec -n 8 step_2/step_2.x -i data_in/testmatrix_a.in -o step_2/testmatrix_a.out
  mpiexec -n 8 step_3/step_3.x -i data_in/testmatrix_a.in -o step_3/testmatrix_a.out
  ```

  Your supervisor will run the tests and verify the correctness of the output file.



```
▼ data_in
    testmatrix_a.in
    testmatrix_b.in
    testmatrix_c.in
  Makefile
  run_tests.sh
  setenv
▼ step_2
    step_2.c
    step_2.x
    testmatrix_a.out
    testmatrix_b.out
    testmatrix_c.out
▼ step_3
    step_3.c
    step_3.x
    testmatrix_a.out
    testmatrix_b.out
    testmatrix_c.out
▶ step_5
▶ step_6
```

Figur 1Example of a project

- Code quality: All routines in your source code need to be documented by a short description of their functionality as well as of their parameters. Also, make the steps in the algorithms visible by comments in the source code.
- Do not duplicate utility routines like the mmio routines or help routines that are used in several algorithms into the different sub-directories. You must create a library instead that can be used in the implementation of all algorithms.

The report

- The structure of the report is given by the work steps. The report must be delivered in the PDF file format, paper size A4. At least your final report shall be added to the Github repository, however, you could use the repository as backup for your report source files, measurement data, or images too.
- Use an appropriate representation of results (tables as well diagrams if possible).
- Use the terms that have been introduced in the first part of the course for the discussion of your measurements (for example "speedup" and "efficiency").
- Your discussion of the algorithms and their execution times shall include a comparison of theoretical expectations and the real measurements of the execution times as functions of problem size and number of processors.
  For example, you could have measured an execution time that follows a time

complexity of O(n^2) as function of the problem size while the theoretical expectation for the algorithm would be O(n).

Additional information for the content of the output file

The output file shall contain the solution vector for the linear system with a right hand side that is a vector of ones. Earlier had been specified that the triangular matrix that is obtained from the Gaussian elimination should be saved in the output file. However, depending on the pivot strategy you could get different results. In order to get unique results, you shall solve the linear following linear system and save the solution vector x in the output file (as column vector). The calculation in Matlab's syntax looks as follows.

```
A = mmread('matrix_0008.mm')
b=ones([size(A,2),1])
x=A\b
mmwrite('result_0008.mm', x)
```

You can test the correctness resp. quality of your solution by using the following vector norm (again in Matlab syntax):

```
norm(A*x-b,Inf)
```

This shall give a very small number in the range of 1.E-16 (for smaller systems) to 1.E-11 (for very large systems like the test matrix with 1024 unknowns).