Computer Architecture and High Performance Computing SoSe 2020

Assignment 1: High Performance Dense Linear Solver

In this assignment, which consists of 5 parts, you will step by step develop and experimentally evaluate a high performance solver for dense linear systems of equations $Ax = b$.

Grading of Assignment 1: Assignment 1 determines 50% of your grade for the class (the other 50% are determined by Prof. Mehofer). To these 50%, each part contributes as indicated below.

Please note: Both Assignment 1 and Assignment 2 (of Prof. Mehofer) have to be positive for a positive grade in the course!

General remarks for all parts of Assignment 1:

- The main objective of all your codes is to maximize performance (minimize runtime) while providing accurate results.
- You have to write your own code. For Part 1, you are not allowed to call BLAS routines. For Parts 2-5, you may call BLAS routines.
- **Hardware target system**: You can use your own hardware system. We will compare the efficiency which you achieved in your experimental evaluations. Moreover, you should also run your codes on our server harris for which you got accounts. This will allow for direct comparison of runtimes.
- **Programming language**: ideally C or Fortran, but any other compiled high-lever language is acceptable.
- **Evaluation of accuracy**:
  - For the computed solution of a linear system:
    - Compute the *relative forward error*, i.e. $\frac{\|x-x\prime\|_1}{\|x\prime\|_1}$, where x is the computed solution and x' is the known exact solution.
    - Compute the *relative residual norm* of the computed solution vector x: $\|r\|_1 := \frac{\|Ax-b\|_1}{\|b\|_1}$. Do not count the work and the time needed for the accuracy evaluation in your performance evaluation!
  - For the LU factorization:
    - Compute the *relative factorization error* $\|PA - LU\|_1/\|A\|_1$, where LU is the computed factorization of the given matrix A.
- **Evaluation of runtime performance**: Measure the runtime of your routine. Make sure that you measure ONLY the time needed for the operation under consideration (without main routine, input data generation, accuracy evaluation, etc.). Explain carefully in your report how you measured the runtime.
  Remark: A very accurate performance measurement can be achieved with hardware performance counters, e.g. PAPI (http://icl.cs.utk.edu/papi/).
- Summarize the results of your experimental evaluations graphically for problem sizes n=100:100:N, where N as large as possible in reasonable time. Note that N depends on your code optimization and on the machine you are using!

Part 1 (10%): Triangular Solve

Write one routine which solves a given upper triangular linear system $U\,x\;=\;b$ for x. Write another routine which solves a given lower triangular linear system $L\,x\;=\;b$ for x. Evaluate accuracy, runtime performance and efficiency of your codes. For experimental evaluations, use randomly generated (non-singular) L and U.
Remark: Determine b such that the exact solution x' is a vector of all ones: $x'\;=\;[1\;1\;1\;...\;1]^T$.

Due date – submission of report (at most 2 pages) and code in Moodle: **10.4.2020, 15:00**

Part 2 (10%): Unblocked Right-looking LU Factorization

Write a routine which performs an unblocked right-looking LU factorization *without pivoting* of a given matrix $A \in \mathbb{R}^{nxn}$. Evaluate accuracy, runtime performance and efficiency of your codes. For the experimental evaluations, use randomly generated (nonsingular) A.

Due date – submission of report (at most 3 pages) and code in Moodle: **22.4.2020, 15:00**

Part 3 (14%): Unblocked Right-looking LU Factorization with Partial Pivoting

Write a routine which performs an unblocked right-looking LU factorization *with partial pivoting* of a given matrix $A \in \mathbb{R}^{nxn}$. Evaluate accuracy, runtime performance and efficiency of your codes. In particular, compare accuracy, runtime and efficiency with and without partial pivoting (from Part 2). For the experimental evaluations, use randomly generated (nonsingular) A.

Due date – submission of report (at most 3 pages) and code in Moodle: **22.4.2020, 15:00**

Part 4 (16%): Blocked Right-Looking LU Factorization with Partial Pivoting

Write a routine which performs a *blocked* right-looking LU factorization with pivoting of a given matrix $A \in \mathbb{R}^{nxn}$. Evaluate accuracy, runtime performance and efficiency of your codes. In particular, experimentally determine the best block size b and compare runtimes and efficiency of blocked and unblocked LU factorization (from Part 3). For the experimental evaluations, use randomly generated (nonsingular) A.

Remark: You will get a part of the available points for correctly implementing a blocked right-looking LU factorization *without* partial pivoting.

Hint: For detailed information about blocked LU factorization, please refer to the paper [Cole], to Section 3 of the paper [Toledo] or to http://www.netlib.org/utk/papers/outofcore/node3.html

Due date – submission of report (at most 4 pages) and code in Moodle: **30.4.2020, 15:00**

Part 5 (FOR VOLUNTEERS – BONUS POINTS – extra 15%): Left-looking LU Factorization, Crout Algorithm, etc.

Implement alternative variants of LU factorization (basically corresponding to different orders of the three nested loops) and compare their performance, efficiency and accuracy to the right-looking LU factorization which you implemented in Parts 2-4. You find more information about these other variants e.g. at http://www.netlib.org/utk/papers/outofcore/node4.html
http://en.wikipedia.org/wiki/Doolittle_decomposition
http://en.wikipedia.org/wiki/Crout_matrix_decomposition

Due date – submission of report (at most 4 pages) and code in Moodle: **30.4.2020, 15:00 (negotiable)**