High Performance Computing 2022, Assignment 1: Sparse LU Factorization

Deadline: Friday, October 21 before 23:59 hours.

In the first programming assignment, you will put what you have learned during the Sparse Matrix Computations lectures into practice. First, implement a Sparse LU Factorization kernel and secondly perform a benchmark of your implementation.

The deadline for the assignment is October 21. The assignment has to be completed individually. You are expected to hand in a tarball containing your source code and a report in PDF format detailing how you have dealt with the various implementation issues and your benchmark results. The assignments should be handed by e-mail to *mattvdnieuwenhuijzen+hpci2022 (at) gmail (dot) com.*

1 Implementation

The first part of the assignment consists of the implementation of a Sparse LU Factorization kernel that applies partial pivoting. You need to implement a *sequential* program. The kernel must be written in C/C++ using solely arrays. No pointers, aliasing or dynamic allocations may be used. Arrays must be sized in such a way that they fit the largest benchmark matrix data at all times. Refer to array elements using array subscripts. Note that Dense LU Factorization is not permitted.

On the website a skeleton code is provided that reads matrices in the “Matrix Market” format and stores these in Compressed Row Storage format. You can use this skeleton as a starting point for your implementation.

In your PDF report, explain how you have dealt with the various implementation issues you have come across, like pivot search, masking operations, garbage collection and permutation issues.

2 Benchmark

In the second part of the assignment, you will benchmark the LU factorization code you have implemented. First perform the factorization and then use the obtained L and U factors to solve the system for five b vectors, specified below. The benchmark includes the execution time required to factorize the system, the total time to compute all five solution vectors, and the relative errors $\frac{||\tilde{x}-x||}{||x||}$ for each solution. $||x||$ denotes the Euclidean Norm: $\sqrt{\sum x_i^2}$, x is the actual solution and $\tilde{x}$ is the solution computed by your implementation of LU factorization. The benchmarks need to be carried out using the Linux operating system on the Core i7 machines in room 302/304 at LIACS, or on one of the *huisuil* servers if working remotely. As test data, we will use 10 matrices from the SuiteSparse Matrix Collection (`https://sparse.tamu.edu/`):

HB/mcfe
Schenk_IBMNA/c-21
Oberwolfach/flowmeter5
Averous/epb1
Grund/meg4
Lucifora/cell1
Gaertner/nopoly
Bai/mhd4800b
FIDAP/ex10
Okunbor/aft01

For the initialization of your program you need to construct five b vectors which are obtained by multiplying a given matrix A with an actual solution vector x (so, $b = Ax$). Take for x the following vectors:

- vector x consisting of all ones.
- vector x consisting of 0.1 for every element.
- vector x consisting of alternating $+1$, -1 values: $+1, -1, +1, -1, +1, -1, \ldots$.

- vector x consisting of alternating $+5$, -5 values.
- vector x consisting of alternating $+100$, -100 values.

Note that the initialization is not part of the actual computation, so should not be included in the benchmark. A standard matrix multiplication should be used to compute these b vectors within your application before the benchmark starts.

Include the obtained results in your report, together with a brief discussion.

3 Input & output

Your submission is expected to include a *Makefile* which builds and outputs an executable called "`lu`" in the *same directory* as the Makefile. You can use the *gcc/g++* compilers for C/C++ code.

In order to easily benchmark your application, the executable is expected to take exactly one argument: the file name of a Matrix Market formatted file. It should output exactly 7 numbers to *stdout*. First it should output the execution time required to factorize the system in seconds, then the total time to compute all five solution vectors in seconds, and finally the relative errors for the five solutions individually. Any output on *stderr* is ignored.