Numerical High Performance Algorithms - Assignment 1

Winter Semester 2018

October 21, 2018

1 High Performance LU Factorization

1.1 Task Description & Introduction

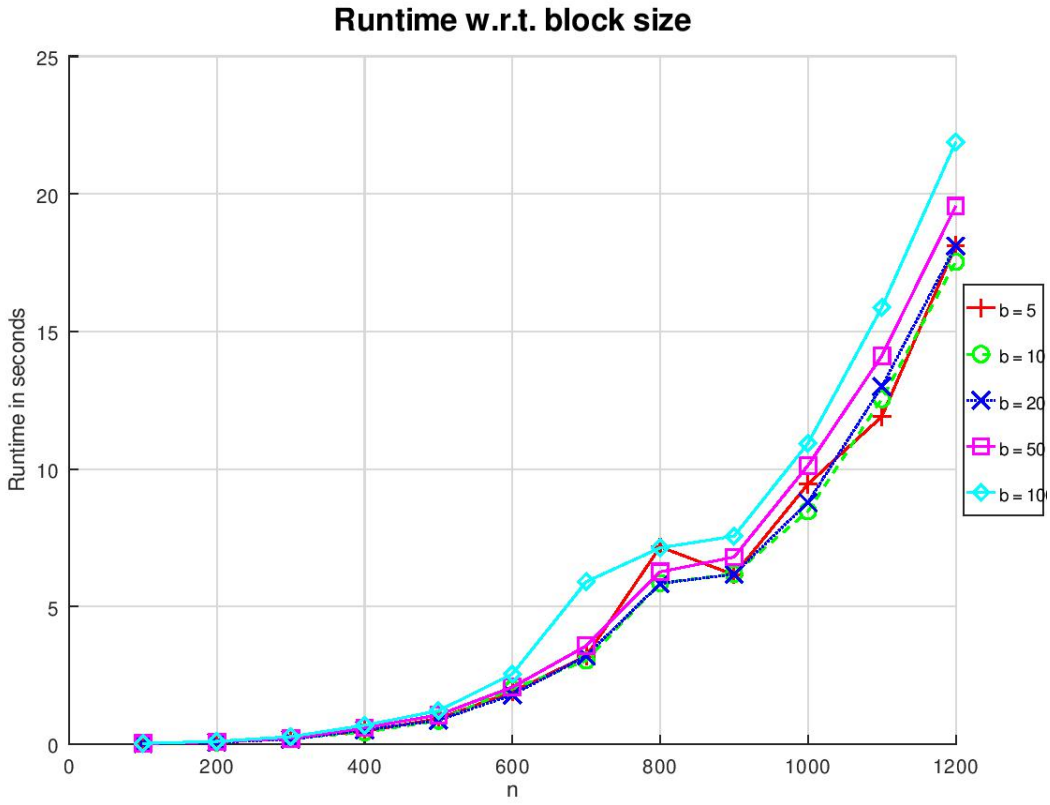
We implement the blocked and non-blocked LU decomposition of square, non-singular matrices and evaluate the implementation's accuracy and performance in terms of run time. We followed the pseudo-code describing the blocked, pivoted LU decomposition in a recursive fashion as presented in [1]. Further inspiration and explanation were drawn from the literature presented in the lecture [2][3].

Notes regarding the implementation:

- Run time was measured using `tic` / `toc`, since we wanted to capture CPU time as well as time spent for other activities, e.g. copying matrices. Only the actual call of `plu` respectively `uplu` and the extraction of L and U from the resulting matrix was measured.
- The functions `plu`, `uplu`, `upluStats` and `pluStats` and stored in the respective `.m` files. Some preliminary measurements regarding the effect of different block sizes have been made in `blocksize_eval.m`. The code executing the principal experiment is to be found in `assignment1.m`. Various other file holding auxiliary functions are included in the uploaded archive.
- We made extensive use of vector operations in our implementation of the unblocked LU decomposition to increase performance. We used the same routine to decompose sub-matrices for the blocked LU decomposition.

1.2 Effect of Varying Block Sizes on the Run Time

We examined the influence of different block sizes of the run time. The measurements can be seen in Fig. 1a, whereas $b \in \{5, 10, 20, 50, 100\}$. The data indicates that a block size of `b = 10` usually yields the best results on the system used, while $b > 10$ seems to increase the run time in proportion with b. We assume a trade-off between overhead costs for the recursive call of the blocked LU routine and the exponentially growing costs of a LU decomposition for bigger sub-matrices.
Thus, we use $b = 10$ for the remainder of the examinations.



(a) Run times w.r.t. problem size n and block size b.

1.3 Evaluation of Results

We measure the following metrics:

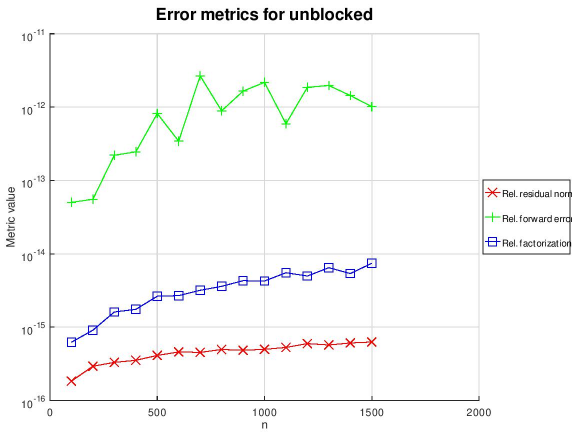
1. *Accuracy*: The correctness of the achieved result in terms of (i) relative forward error $foe = \frac{||x-x'||_1}{||x'||_1}$[1], (ii) relative residual norm $rn = \frac{||Ax-b||_1}{||b||_1}$, and (iii) relative factorization error $fae = \frac{||PA-LU||_1}{||A||_1}$.

2. *Run time*: Time needed for the actual LU decomposition including the extraction of L and U from the resulting matrix.

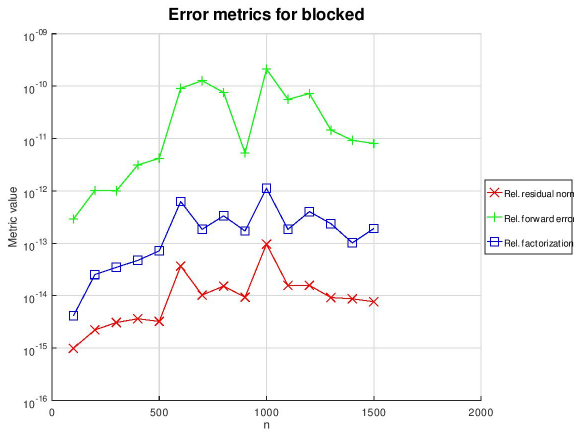
3. *Efficiency*: We used the total number of flops needed to compute the LU decomposition, $n_{flops} = 2 \cdot \frac{n^3}{3}$, and divided it by the run time in seconds needed for the LU decomposition. Since the resulting number represents the machine-dependent efficiency, we further divide it by the number of flops the machine running the code is able to conduct[2].

[1] x is the computed and x' the exact solution (a vector of all ones).

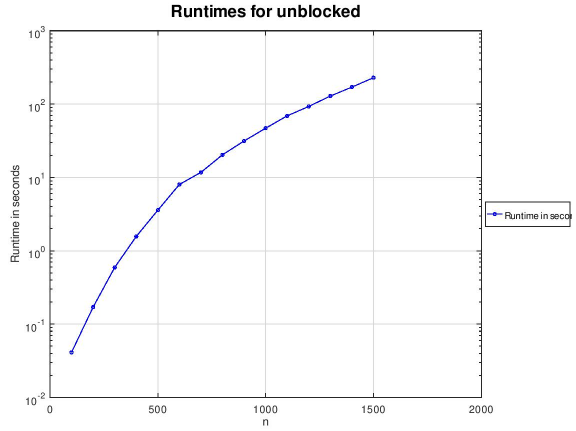
[2] For example: https://asteroidsathome.net/boinc/cpu_list.php lists the number of flops for various CPUs, e. g. the one our machine is equipped with - a i7-6500U CPU running at 2.50GHz. According to this list this CPU can process simultaneously $3.29 * 10^{12}$ flops. Whether this number is totally accurate or application-specific might influence the absolute efficiency; but the comparison of unblocked to blocked efficiencies are still viable.



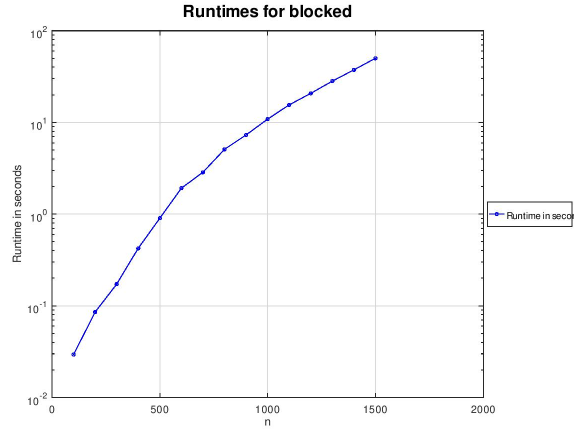
(a) Accuracy metrics for unblocked LU decomposition.



(b) Accuracy metrics for blocked LU decomposition.



(a) Run times for unblocked LU decomposition.



(b) Run times for blocked LU decomposition.

Both blocked and unblocked LU decomposition are run on the same randomly generated non-singular, square matrices with $n \in \{100 : 100 : 1500\}$.

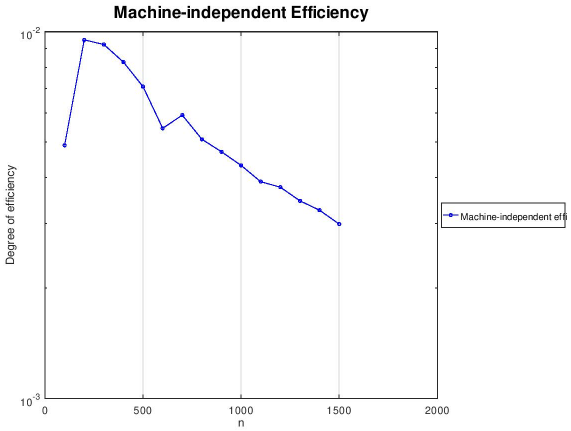
The result accuracy values are displayed in Figures 2b and 2a; the run times are plotted in Figures 3b and 3a.

In terms of accuracy and speed, the unblocked version seems preferable: Although we implemented a pivoted version of both the blocking *and* unblocking LU algorithm, forward error and residual norm seem to react more sensitively to the input matrix' condition and are mostly larger using the blocked version. While we can't offer a definitive proof in this regard, our intuition is that repeated multiplication of the smaller (sub-)matrices might reinforce the error margin.

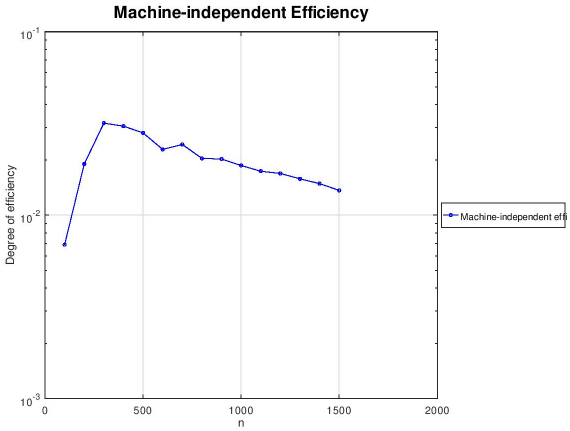
Regarding performance, the blocked version is superior: With our largest n of 1500 there was a relative speedup of roughly 5.[3]

For both blocked and unblocked versions, the machine-independent efficiency - as seen in Figures 4a and 4b - decreases with increasing n. It does so faster for the unblocked than for

[3]Note: This number was estimated, gathering data from plots Figures 3a and 3b at $n = 1500$.



(a) Efficiency for unblocked LU decomposition.



(b) Efficiency for blocked LU decomposition.

the blocked version. The absolute degree of efficiency is also higher with the blocked version.

1.4 Conclusions

We conclude that while the blocked LU decomposition is vastly faster than the unblocked version - more examinations including further optimizations for both approaches would be interesting - the distinct loss of accuracy is surprising. While the resulting accuracy might still be sufficient for many use-cases, we are keen to know whether the lower accuracy is due to our implementation or a natural consequence of the (recursive) blocked LU decomposition algorithm.

The blocked version is not just faster, but also more efficient than the unblocked alternative in terms of flops actually calculated in a given time period.

References

[1] Golub, G. H., & Van Loan, C. F. (2012). *Matrix computations (Vol. 4)*. Johns Hopkins University Press.

[2] Cole, J. *LU Factorization Measurement and Optimization.*

[3] Toledo, S. (1997). *Locality of reference in LU decomposition with partial pivoting*. SIAM Journal on Matrix Analysis and Applications, 18(4), 1065-1081.