Blocked algorithm for Pivoted LU Factorization Using Permutation in One-line Notation

Anže Marinko

Numerical High Performance Algorithms
University of Vienna
October 2019

1 Introduction

For solving linear systems we know a good algorithm for LU factorization with partial pivoting for a given matrix but we can do it even better if we use blocked algorithm. In that report we will look for an improvement of algorithm because we often multiply with a very dense permutation matrix in blocked algorithm and it increases runtime. Blocked algorithm already optimizes runtime, but we will show that permutation in one-line notation instead of permutation matrix optimizes runtime even more and memory too.

2 Method

We are solving linear system $Ax = b, A \in \Re^{n\times n}, b \in \Re^n$ to find exact $x \in \Re^n$ that solves system. We can compute LU decomposition ($PA = LU$), where L is lower-triangular, U upper-triangular and P permutation matrix. When we have it computed we make forward substitution so we find y that $Ly = \hat{b}, \hat{b} = Pb$. Triangular systems are simple to compute. Than we make back substitution so we find x that $Ux = y$. It is also triangular system.

We can make it using blocks as proposed in [1] and than we have to select size of blocks. But in blocked algorithm we use even more permutations and multiplying matrices is expensive, so we will try to improve it with use of a more compact notation of permutation matrix as permutation in one-line notation

$$\sigma = \begin{bmatrix}\sigma_1 & \sigma_2 & \dots & \sigma_n\end{bmatrix}$$

that means row i goes to $\sigma_i \in \{1, \dots, n\}$ if we multiply it with a matrix or other permutation.

For measuring accuracy we will use some statistics about results. Determine b such that exact solution is $\hat{x} = \begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix}^T$.

Relative forward error is $\frac{\|\hat{x}-x\|_1}{\|\hat{x}\|_1}$ where x is computed solution, *relative residual norm* is $\|r\|_1 := \frac{\|Ax-b\|_1}{\|b\|_1}$ and *relative factorization error* is $\frac{\|PA-LU\|_1}{\|A\|_1}$ where P, L, U are computed.

We measure time with a decorative function *timer* to measure runtime of function for LU factorization and for whole solving of a linear system. All

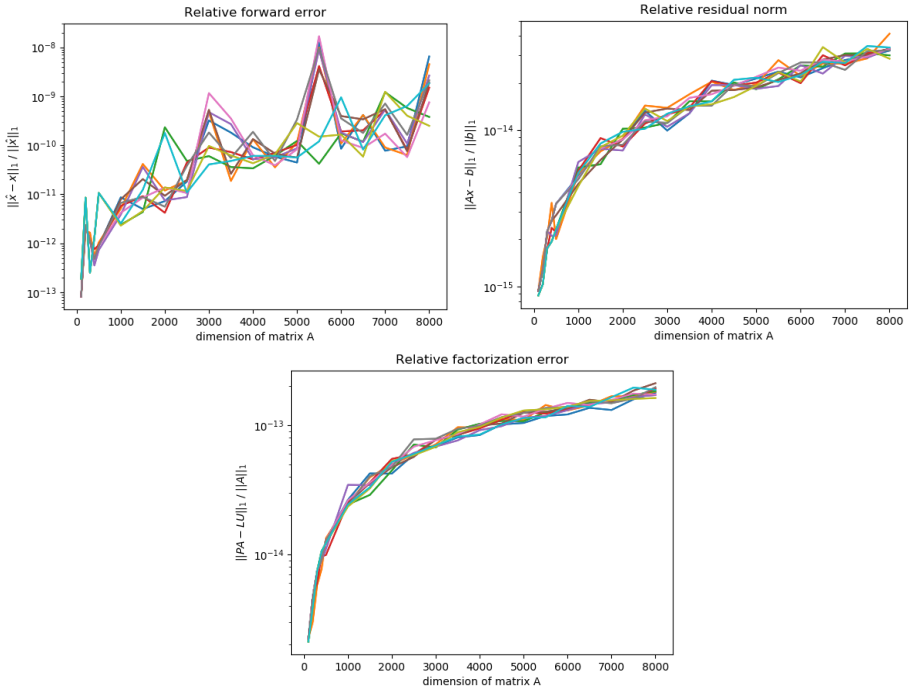

Figure 1: Some statistics about results

algorithms are implemented in Python. Because my hardware doesnt support counting *flops*, I cannot compute efficiency.

3 Results

On figure 1 we can see that all implemented algorithms have quite similar relative forwar and relative factorization error and relative residual norm that incrises nearly linearly with n. Basic LU factorization (later LU), basic blocked LU algorithm (later BLU) and my algorithm using permutations (later BLUP) are so quite equaly accurate.

On figure 2 we can see on the upper image that all settings of block size of BLU and BLUP are better than LU and that most of settings of block size of BLUP are better than the best parametrized BLU. On the lower left image we can even see the runtimes of LU and BLU and BLUP using the best block size depanding on n. On the lower left image we can see which block size out of $100, 200, \ldots, 1000$ is the best for BLU and BLUP.

On the figure 3 we can see relative runtime of BLU and BLUP comparing to the LU runtime on the left and relative runtime of BLUP comparing to the BLU on the right image.

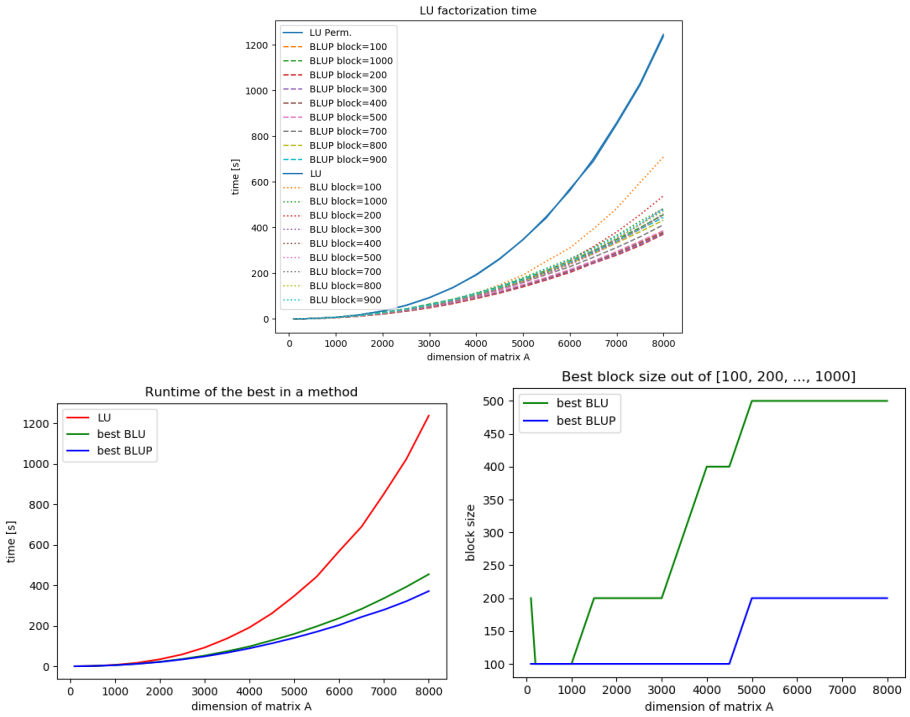

Figure 2: Testing runtimes (tested on a server), the best runtime of method (left down) and the best block size (right down)

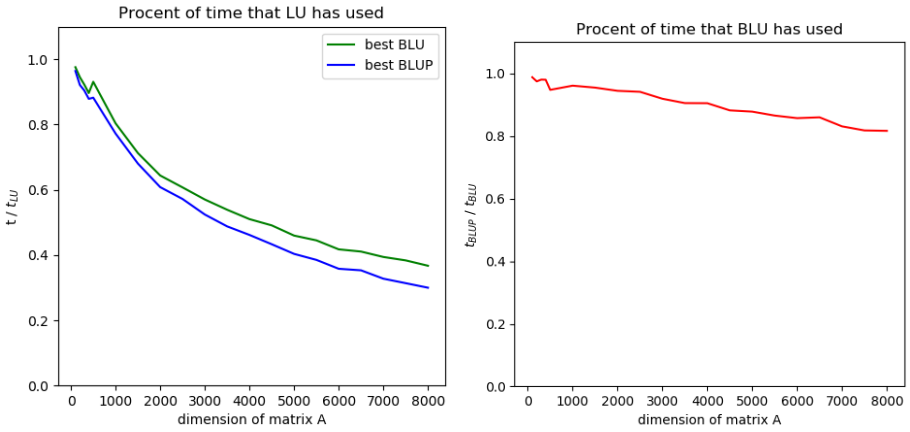

Figure 3: Improvement of basic LU reached by best of BLU and BLUP (left) and improvement of BLU by using permutations (right)

4 Discusion

We have seen that we can segnificately improve runtime of LU with a blocked algorithm and there is some improvement with use of permutation in one-line notation instead of permutation matrix too. It looks like getting even better

when n increases. For $n = 8000$ it is already nearly 20% faster. But we should try to implement it without use of Numpy library to get more precise information how much does it realy improve an algorithm.

5 References

1. J. Cole, *LU factorization measurement and optimisation*, Carnegie Mellon University
2. *https://en.wikipedia.org/wiki/Permutation*, 25th October 2019
3. *http://www.netlib.org/utk/papers/outofcore/node3.html*, 22th October 2019