Simultaneous Computation of the Row and Column Rank Profiles*

Jean-Guillaume Dumas
Université de Grenoble
Laboratoire LJK
UMR CNRS, INRIA, UJF, UPMF, GINP
51, av. des Mathématiques,
F38041 Grenoble, France
Jean-Guillaume.Dumas@imag.fr

Clément Pernet
Université de Grenoble
INRIA, Laboratoire LIG
UMR CNRS, INRIA, UJF, UPMF, GINP
Inovallée, 655, av. de l'Europe,
F38334 St Ismier Cedex,
France
Clement.Pernet@imag.fr

Ziad Sultan
Université de Grenoble
Laboratoires LJK and LIG
UMR CNRS, INRIA, UJF, UPMF, GINP
Inovallée, 655, av. de l'Europe,
F38334 St Ismier Cedex,
France
Ziad.Sultan@imag.fr

ABSTRACT

Gaussian elimination with full pivoting generates a PLUQ matrix decomposition. Depending on the strategy used in the search for pivots, the permutation matrices can reveal some information about the row or the column rank profiles of the matrix. We propose a new pivoting strategy that makes it possible to recover at the same time both row and column rank profiles of the input matrix and of any of its leading sub-matrices. We propose a rank-sensitive and quad-recursive algorithm that computes the latter PLUQ triangular decomposition of an $m \times n$ matrix of rank r in $O\left(mnr^{\omega-2}\right)$ field operations, with ω the exponent of matrix multiplication. Compared to the LEU decomposition by Malashonock, sharing a similar recursive structure, its time complexity is rank sensitive and has a lower leading constant. Over a word size finite field, this algorithm also improves the practical efficiency of previously known implementations.

Categories and Subject Descriptors

G.4 [**Mathematics and Computing**]: Mathematical Software—*Algorithm Design and Analysis*; I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation

Keywords

FFLAS-FFPACK, Finite field, Gaussian elimination, Rank profile

1. INTRODUCTION

Triangular matrix decomposition is a fundamental building block in computational linear algebra. It is used to solve linear systems, compute the rank, the determinant, the nullspace or the row and column rank profiles of a matrix. The LU decomposition, defined for matrices whose leading principal minors are all nonsingular, can be generalized to arbitrary dimensions and ranks by introducing pivoting on sides, leading e.g. to the LQUP decomposition of [6] or the PLUQ decomposition [5, 8]. Many algorithmic variants exist allowing fraction free computations [8], in-place computations [2, 7] or sub-cubic rank-sensitive time complexity [11, 7]. More precisely, the pivoting strategy reflected by the permutation matrices P and Q is the key difference between these PLUQ decompositions. In numerical linear algebra [5], pivoting is used to ensure a good numerical stability, good data locality, and reduce the fill-in. In the context of exact linear algebra, the role of pivoting differs. Indeed, only certain pivoting strategies for these decompositions will reveal the rank profile of the matrix. The latter is crucial in many applications using exact Gaussian elimination, such as Gröbner basis computations [4] and computational number theory [10].

The *row rank profile* of an $m \times n$ matrix A with rank r is the lexicographically smallest sequence of r indices of linearly independent rows of A. Similarly the *column rank profile* is the lexicographically smallest sequence of r indices of linearly independent columns of A.

The common strategy to compute the row rank profile is to search for pivots in a row-major fashion: exploring the current row, then moving to the next row only if the current row is zero. Such a $\bar{P}LU\bar{Q}$ decomposition can be transformed into a CUP decomposition (where $P = \bar{Q}$ and $C = \bar{P}L$ is in column echelon form) and the first r values of the permutation associated to P are exactly the row rank profile [7]. A block recursive algorithm can be derived from this scheme by splitting the row dimension [6]. Similarly, the column rank profile can be obtained in a column major search: exploring the current column, and moving to the next column only if the current one is zero. The $\bar{P}LU\bar{Q}$ decomposition can be transformed into a PLE decomposition (where $P = \bar{P}$ and $E = U\bar{Q}$ is in row echelon form) and the first r values of Q are exactly the column rank profile [7]. The corresponding block recursive algorithm uses a splitting of the column dimension.

Recursive elimination algorithms splitting both row and column dimensions include the TURBO algorithm [3] and the LEU decomposition [9]. No connection is made to the computation of the rank profiles in any of these algorithms. The TURBO algorithm does not compute the lower triangular matrix L and performs five recursive calls. It therefore

*This work is partly funded by the HPAC project of the French Agence Nationale de la Recherche (ANR 11 BS02 013).

ISSAC'13, June 26–29, 2013, Boston, Massachusetts, USA.

implies an arithmetic overhead compared to classic Gaussian elimination. The LEU avoids permutations but at the expense of many additional matrix products. As a consequence its time complexity is not rank-sensitive.

We propose here a pivoting strategy following a Z-curve structure and working on an incrementally growing leading sub-matrix. This strategy is first used in a recursive algorithm splitting both rows and columns which recovers simultaneously both row and column rank profiles. Moreover, the row and column rank profiles of *any leading sub-matrix* can be deduced from the P and Q permutations. We show that the arithmetic cost of this algorithm remains rank sensitive of the form $O(mnr^{\omega-2})$ where ω is the exponent of matrix multiplication. The best currently known upper bound for ω is 2.3727 [12]. To the best of our knowledge, this is the first reduction to matrix multiplication for the problem of computing the column and row rank profiles of all leading sub-matrices of an input matrix.

As for the CUP and PLE decompositions, this PLUQ decomposition can be computed in-place, at the same cost. Compared to the CUP and PLE decompositions, this new algorithm has the following new salient features:

- it computes *simultaneously* both rank profiles at the cost of one,
- it preserves the squareness of the matrix passed to the recursive calls, thus allowing more efficient use of the matrix multiplication building block,
- it uses less modular reductions in a finite field,
- a CUP and a PLE decompositions can be obtained from it, with row and column permutations only.

Compared to the LEU decomposition,

- it is in-place,
- its time complexity bound is rank sensitive and has a better leading constant,
- a LEU decomposition can be obtained from it, with row and column permutations.

In Section 2 we present the new block recursive algorithm. Section 3 shows the connection with the LEU decomposition and Section 4 states the main property about rank profiles. We then analyze the complexity of the new algorithm in terms of arithmetic operations: first we prove that it is rank sensitive in Section 5 and second we show in Section 6 that, over a finite field, it reduces the number of modular reductions when compared to state of the art techniques. We then propose an iterative variant in Section 7 to be used as a base-case to terminate the recursion before the dimensions get too small. Experiments comparing computation time and cache efficiency are presented in Section 8.

2. A RECURSIVE PLUQ ALGORITHM

We first recall the name of the main sub-routines being used: `MM` stands for matrix multiplication, `TRSM` for triangular system solving with matrix unknown (left and right variants are implicitly indicated by the parameter list), `PermC` for matrix column permutation, `PermR` for matrix row permutation, etc. For instance, we will use:
$\texttt{MM}(C, A, B)$ to denote $C \leftarrow C - AB$,

$\texttt{TRSM}(U, B)$ for $B \leftarrow U^{-1}B$ with U upper triangular,

$\texttt{TRSM}(B, L)$ for $B \leftarrow BL^{-1}$ with L lower triangular.

We also denote by $T_{k,l}$ the transposition of indices k and l and by $L\backslash U$, the storage of the two triangular matrices L and U one above the other. Further details on these subroutines and notations can be found in [7]. In block decompositions, we allow for zero dimensions. By convention, the product of any $m \times 0$ matrix by an $0 \times n$ matrix is the $m \times n$ zero matrix. The notation $j = i : k$ being inclusive on the left only (i.e. $j = i : k$ means $j \in \mathbb{Z}$ and $i \leq j < k$).

We now present the block recursive Algorithm 1, computing a PLUQ decomposition.

It is based on a splitting of the matrix in four quadrants. A first recursive call is done on the upper left quadrant followed by a series of updates. Then two recursive calls can be made on the anti-diagonal quadrants if the first quadrant exposed some rank deficiency. After a last series of updates, a fourth recursive call is done on the bottom right quadrant. Figure 1 illustrates the position of the blocks computed in the course of Algorithm 1, before and after the final permutation with matrices S and T.

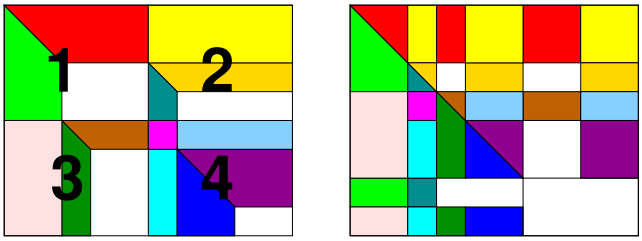

Figure 1: Block recursive Z-curve PLUQ decomposition and final block permutation.

This framework differs from the one in [3] by the order in which the quadrants are treated, leading to only four recursive calls in this case instead of five in [3]. We will show in Section 4 that this fact together with the special form of the block permutations S and T makes it possible to recover rank profile information. The correctness of Algorithm 1 is proven in Appendix A.

REMARK 1. *Algorithm 1 is in-place (as defined in [7, Definition 1]): all operations of the `TRSM`, `MM`, `PermC`, `PermR` subroutines work with $O(1)$ extra memory allocations except possibly in the course of fast matrix multiplications. The only constraint is for the computation of $J \leftarrow L_3^{-1}I$ which would overwrite the matrix I that should be kept for the final output. Hence a copy of I has to be stored for the computation of J. The matrix I has dimension $r_3 \times r_2$ and can be stored transposed in the zero block of the upper left quadrant (of dimension $(\frac{m}{2} - r_1) \times (\frac{n}{2} - r_1)$, as shown on Figure 1).*

Algorithm 1 PLUQ

Input: $A = (a_{ij})$ a $m \times n$ matrix over a field
Output: P, Q: $m \times m$ and $n \times n$ permutation matrices
Output: r: the rank of A
Output: $A \leftarrow \begin{bmatrix} L\backslash U & V \\ M & 0 \end{bmatrix}$ where L is $r \times r$ unit lower triangular, U is $r \times r$ upper triangular, and

$$A = P \begin{bmatrix} L \\ M \end{bmatrix} \begin{bmatrix} U & V \end{bmatrix} Q.$$

if m=1 **then**
  if $A = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}$ **then** $P \leftarrow \begin{bmatrix} 1 \end{bmatrix}, Q \leftarrow I_n, r \leftarrow 0$
  else
    $i \leftarrow$ the col. index of the first non zero elt. of A
    $P \leftarrow \begin{bmatrix} 1 \end{bmatrix}; Q \leftarrow T_{1,i}, r \leftarrow 1$
    Swap $a_{1,i}$ and $a_{1,1}$
  end if
  Return (P, Q, r, A)
end if
if n=1 **then**
  if $A = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^T$ **then** $P \leftarrow I_m; Q \leftarrow \begin{bmatrix} 1 \end{bmatrix}, r \leftarrow 0$
  else
    $i \leftarrow$ the row index of the first non zero elt. of A
    $P \leftarrow \begin{bmatrix} 1 \end{bmatrix}, Q \leftarrow T_{1,i}, r \leftarrow 1$
    Swap $a_{i,1}$ and $a_{1,1}$
    for $j = i+1 : m$ **do** $a_{j,1} \leftarrow a_{j,1} a_{1,1}^{-1}$
    end for
  end if
  Return (P, Q, r, A)
end if

▷ Splitting $A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$ where A_1 is $\lfloor \frac{m}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$.

Decompose $A_1 = P_1 \begin{bmatrix} L_1 \\ M_1 \end{bmatrix} \begin{bmatrix} U_1 & V_1 \end{bmatrix} Q_1$ ▷ PLUQ(A_1)

$\begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \leftarrow P_1^T A_2$ ▷ PermR(A_2, P_1^T)

$\begin{bmatrix} C_1 & C_2 \end{bmatrix} \leftarrow A_3 Q_1^T$ ▷ PermC(A_3, Q_1^T)

Here $A = \left[\begin{array}{cc|c} L_1\backslash U_1 & V_1 & B_1 \\ M_1 & 0 & B_2 \\ \hline C_1 & C_2 & A_4 \end{array}\right]$.

$D \leftarrow L_1^{-1} B_1$ ▷ TRSM(L_1, B_1)
$E \leftarrow C_1 U_1^{-1}$ ▷ TRSM(C_1, U_1)
$F \leftarrow B_2 - M_1 D$ ▷ MM(B_2, M_1, D)
$G \leftarrow C_2 - E V_1$ ▷ MM(C_2, E, V_1)
$H \leftarrow A_4 - E D$ ▷ MM(A_4, E, D)

Here $A = \left[\begin{array}{cc|c} L_1\backslash U_1 & V_1 & D \\ M_1 & 0 & F \\ \hline E & G & H \end{array}\right]$.

Decompose $F = P_2 \begin{bmatrix} L_2 \\ M_2 \end{bmatrix} \begin{bmatrix} U_2 & V_2 \end{bmatrix} Q_2$ ▷ PLUQ(F)

Decompose $G = P_3 \begin{bmatrix} L_3 \\ M_3 \end{bmatrix} \begin{bmatrix} U_3 & V_3 \end{bmatrix} Q_3$ ▷ PLUQ(G)

$\begin{bmatrix} H_1 & H_2 \\ H_3 & H_4 \end{bmatrix} \leftarrow P_3^T H Q_2^T$ ▷ PermR(H, P_3^T); PermC(H, Q_2^T)

$\begin{bmatrix} E_1 \\ E_2 \end{bmatrix} \leftarrow P_3^T E$ ▷ PermR(E, P_3^T)

$\begin{bmatrix} M_{11} \\ M_{12} \end{bmatrix} \leftarrow P_2^T M_1$ ▷ PermR(M_1, P_2^T)

$\begin{bmatrix} D_1 & D_2 \end{bmatrix} \leftarrow D Q_2^T$ ▷ PermR(D, Q_2^T)

$\begin{bmatrix} V_{11} & V_{12} \end{bmatrix} \leftarrow V_1 Q_3^T$ ▷ PermR(V_1, Q_3^T)

Here $A = \left[\begin{array}{ccc|cc} L_1\backslash U_1 & V_{11} & V_{12} & D_1 & D_2 \\ M_{11} & 0 & 0 & L_2\backslash U_2 & V_2 \\ M_{12} & 0 & 0 & M_2 & 0 \\ \hline E_1 & L_3\backslash U_3 & V_3 & H_1 & H_2 \\ E_2 & M_3 & 0 & H_3 & H_4 \end{array}\right]$.

$I \leftarrow H_1 U_2^{-1}$ ▷ TRSM(H_1, U_2)
$J \leftarrow L_3^{-1} I$ ▷ TRSM(L_3, I)
$K \leftarrow H_3 U_2^{-1}$ ▷ TRSM(H_3, U_2)
$N \leftarrow L_3^{-1} H_2$ ▷ TRSM(L_3, H_2)
$O \leftarrow N - J V_2$ ▷ MM(N, J, V_2)
$R \leftarrow H_4 - K V_2 - M_3 O$ ▷ MM(H_4, K, V_2); MM(H_4, M_3, O)

Decompose $R = P_4 \begin{bmatrix} L_4 \\ M_4 \end{bmatrix} \begin{bmatrix} U_4 & V_4 \end{bmatrix} Q_4$ ▷ PLUQ(R)

$\begin{bmatrix} E_{21} & M_{31} & 0 & K_1 \\ E_{22} & M_{32} & 0 & K_2 \end{bmatrix} \leftarrow P_4^T \begin{bmatrix} E_2 & M_3 & 0 & K \end{bmatrix}$ ▷ PermR

$\begin{bmatrix} D_{21} & D_{22} \\ V_{21} & V_{22} \\ 0 & 0 \\ O_1 & O_2 \end{bmatrix} \leftarrow \begin{bmatrix} D_2 \\ V_2 \\ 0 \\ O \end{bmatrix} Q_4^T$ ▷ PermC

Here $A = \left[\begin{array}{ccc|ccc} L_1\backslash U_1 & V_{11} & V_{12} & D_1 & D_{21} & D_{22} \\ M_{11} & 0 & 0 & L_2\backslash U_2 & V_{21} & V_{22} \\ M_{12} & 0 & 0 & M_2 & 0 & 0 \\ \hline E_1 & L_3\backslash U_3 & V_3 & I & O_1 & O_2 \\ E_{21} & M_{31} & 0 & K_1 & L_4\backslash U_4 & V_4 \\ E_{22} & M_{32} & 0 & K_2 & M_4 & 0 \end{array}\right]$.

$S \leftarrow \begin{bmatrix} I_{r_1+r_2} & & & & \\ & & I_{k-r_1-r_2} & & \\ & I_{r_3+r_4} & & & \\ & & & & I_{m-k-r_3-r_4} \end{bmatrix}$

$T \leftarrow \begin{bmatrix} I_{r_1} & & & & \\ & & I_{r_2} & & \\ & I_{r_3} & & & \\ & & & I_{r_4} & \\ & & I_{k-r_1-r_3} & & \\ & & & & I_{n-k-r_2-r_4} \end{bmatrix}$

$P \leftarrow \mathrm{Diag}(P_1 \begin{bmatrix} I_{r_1} & \\ & P_2 \end{bmatrix}, P_3 \begin{bmatrix} I_{r_3} & \\ & P_4 \end{bmatrix}) S$

$Q \leftarrow T \mathrm{Diag}(\begin{bmatrix} I_{r_1} & \\ & Q_3 \end{bmatrix} Q_1, \begin{bmatrix} I_{r_2} & \\ & Q_4 \end{bmatrix} Q_2)$

$A \leftarrow S^T A T^T$ ▷ PermR(A, S^T); PermC(A, T^T)

Here $A = \left[\begin{array}{cccccc} L_1\backslash U_1 & D_1 & V_{11} & D_{21} & V_{12} & D_{22} \\ M_{11} & L_2\backslash U_2 & 0 & V_{21} & 0 & V_{22} \\ E_1 & I & L_3\backslash U_3 & O_1 & V_3 & O_2 \\ E_{21} & K_1 & M_{31} & L_4\backslash U_4 & 0 & V_4 \\ M_{12} & M_2 & 0 & 0 & 0 & 0 \\ E_{22} & K_2 & M_{32} & M_4 & 0 & 0 \end{array}\right]$

Return $(P, Q, r_1 + r_2 + r_3 + r_4, A)$

3. FROM PLUQ TO LEU

We now show how to compute the LEU decomposition of [9] from the PLUQ decomposition. The idea is to write

$$P \begin{bmatrix} L \\ M \end{bmatrix} [U\, V]\, Q = \underbrace{P \begin{bmatrix} L & 0 \\ M & I_{m-r} \end{bmatrix} P^T}_{\overline{L}} \underbrace{P \begin{bmatrix} I_r \\ 0 \end{bmatrix}}_{E} \underbrace{Q\, Q^T \begin{bmatrix} U & V \\ & I_{n-r} \end{bmatrix} Q}_{\overline{U}}$$

and show that $\overline{L}$ and $\overline{U}$ are respectively lower and upper triangular. This is not true in general, but turns out to be satisfied by the P, L, M, U, V and Q obtained in Algorithm 1.

THEOREM 1. *Let $A = P\begin{bmatrix}L\\M\end{bmatrix}[U\ V]\,Q$ be the PLUQ decomposition computed by Algorithm 1. Then for any unit lower triangular matrix Y and any upper triangular matrix Z, the matrix $P\begin{bmatrix}L\\M\ Y\end{bmatrix}P^T$ is unit lower triangular and $Q^T\begin{bmatrix}U\ V\\ \ \ Z\end{bmatrix}Q$ is upper triangular.*

PROOF. Proceeding by induction, we assume that the theorem is true on all four recursive calls, and show that it is true for the matrices $P\begin{bmatrix}L\\M\ Y\end{bmatrix}P^T$ and $Q^T\begin{bmatrix}U\ V\\ \ \ Z\end{bmatrix}Q$. Let $Y = \begin{bmatrix}Y_1\\Y_2Y_3\end{bmatrix}$ where Y_1 is unit lower triangular of dimension $k - r_1 - r_2$. From the correctness of Algorithm 1 (see e.g. Equation A), $S\begin{bmatrix}L\\MY\end{bmatrix}S^T = \left[\begin{array}{llll} L_1 & & & \\ M_{11} & L_2 & & \\ M_{12} & M_2Y_1 & & \\ \hline E_1 & I & L_3 & \\ E_{21} & K_1 & M_{31} & L_4 \\ E_{22} & K_2Y_2 & M_{32} & M_4Y_3\end{array}\right]$

Hence $P\begin{bmatrix}L\\MY\end{bmatrix}P^T$ equals

$$\begin{bmatrix}P_1\\ \ \ P_3\end{bmatrix}\left[\begin{array}{llll} I_{r_1} & & & \\ & P_2 & & \\ & & I_{r_3} & \\ & & & P_4\end{array}\right]\left[\begin{array}{llll} L_1 & & & \\ M_{11} & L_2 & & \\ M_{12} & M_2Y_1 & & \\ \hline E_1 & I & L_3 & \\ E_{21} & K_1 & M_{31} & L_4 \\ E_{22} & K_2Y_2 & M_{32} & M_4Y_3\end{array}\right]\times \left[\begin{array}{llll} I_{r_1} & & & \\ & P_2^T & & \\ & & I_{r_3} & \\ & & & P_4^T\end{array}\right]\begin{bmatrix}P_1^T\\ \ \ P_3^T\end{bmatrix}$$

By induction hypothesis, the matrices $\overline{L_2} = P_2\begin{bmatrix}L_2\\M_2Y_1\end{bmatrix}P_2^T$, $\overline{L_4} = P_4\begin{bmatrix}L_4\\M_4Y_3\end{bmatrix}P_4^T$, $P_1\begin{bmatrix}L_1\\M_1\overline{L_2}\end{bmatrix}P_1^T$ and $P_3\begin{bmatrix}L_3\\M_3\overline{L_4}\end{bmatrix}P_3^T$ are unit lower triangular. Therefore the matrix $P\begin{bmatrix}L\\M\ Y\end{bmatrix}P^T$ is also unit lower triangular.

Similarly, let $Z = \begin{bmatrix}Z_1Z_2\\ \ \ Z_3\end{bmatrix}$ where Z_1 is upper triangular of dimension $k - r_1 - r_2$. The matrix $T^T\begin{bmatrix}UV\\ \ \ Z\end{bmatrix}T$ equals

$$T^T\left[\begin{array}{lll|lll} U_1 & V_{11} & V_{12} & D_1 & D_{21} & D_{22} \\ 0 & 0 & U_2 & V_{21} & V_{22} \\ U_3 & V_3 & 0 & O_1 & O_2 \\ 0 & & U_4 & V_4 & \\ Z_1 & & & Z_2 & \\ & & & Z_3 & \end{array}\right] = \left[\begin{array}{lll|lll} U_1 & V_{11} & V_{12} & D_1 & D_{21} & D_{22} \\ & U_3 & V_3 & & O_1 & O_2 \\ & & Z_1 & & & Z_2 \\ 0 & 0 & & U_2 & V_{21} & V_{22} \\ & 0 & & & U_4 & V_4 \\ & & & & & Z_3 \end{array}\right]$$

Hence $Q^T\begin{bmatrix}UV\\ \ \ Z\end{bmatrix}Q$ equals

$$\begin{bmatrix}Q_1^T\\ \ \ Q_2^T\end{bmatrix}\left[\begin{array}{llll} I_{r_1} & & & \\ & Q_3^T & & \\ & & I_{r_2} & \\ & & & Q_4^T\end{array}\right]\left[\begin{array}{lll|lll} U_1 & V_{11} & V_{12} & D_1 & D_{21} & D_{22} \\ & U_3 & V_3 & & O_1 & O_2 \\ & & Z_1 & & & Z_2 \\ 0 & 0 & & U_2 & V_{21} & V_{22} \\ & 0 & & & U_4 & V_4 \\ & & & & & Z_3 \end{array}\right]\times \left[\begin{array}{llll} I_{r_1} & & & \\ & Q_3 & & \\ & & I_{r_2} & \\ & & & Q_4\end{array}\right]\begin{bmatrix}Q_1\\ \ \ Q_2\end{bmatrix}.$$

By induction hypothesis, the matrices $\overline{U_3} = Q_3^T\begin{bmatrix}U_3V_3\\ \ \ Z_1\end{bmatrix}Q_3$, $\overline{U_4} = Q_4^T\begin{bmatrix}U_4V_4\\ \ \ Z_3\end{bmatrix}P_4^T$, $Q_1^T\begin{bmatrix}U_1V_1\\ \ \ \overline{U_3}\end{bmatrix}Q_1$ and $Q_2^T\begin{bmatrix}U_2V_2\\ \ \ \overline{U_4}\end{bmatrix}Q_2$ are upper triangular. Consequently the matrix $Q^T\begin{bmatrix}U\ V\\ \ \ Z\end{bmatrix}Q$ is upper triangular.

For the base case with $m = 1$. The matrix $\overline{L}$ has dimension 1×1 and is unit lower triangular. If $r = 0$, then $\overline{U} = I_n^T Z I_n$ is upper triangular. If $r = 1$, then $Q = T_{1,i}$ where i is the column index of the pivot and is therefore the column index of the leading coefficient of the row $[UV]\,Q$. Applying Q^T on the left only swaps rows 1 and i, hence row $[UV]\,Q$ is the ith row of $Q^T\begin{bmatrix}UV\\ \ \ Z\end{bmatrix}Q$. The latter is therefore upper triangular. The same reasoning can be applied to the case $n = 1$. □

COROLLARY 1. *Let $\overline{L} = P\begin{bmatrix}L\\MI_{m-r}\end{bmatrix}P^T, E = P\begin{bmatrix}I_r\\0\end{bmatrix}Q$ and $\overline{U} = Q^T\begin{bmatrix}UV\\0\end{bmatrix}Q$. Then $A = \overline{L}E\overline{U}$ is a LEU decomposition of A.*

REMARK 2. *The converse is not always possible: given $A = L, E, U$, there are several ways to choose the last $m-r$ columns of P and the last $n-r$ rows of Q. The LEU algorithm does not keep track of these parts of the permutations.*

4. COMPUTING THE RANK PROFILES

We prove here the main feature of the PLUQ decomposition computed by Algorithm 1: it reveals the row and column rank profiles of all leading sub-matrices of the input matrix. We recall in Lemma 1 basic properties of rank profiles.

LEMMA 1. *For any matrix,*

1. *the row rank profile is preserved by right multiplication with an invertible matrix and by left multiplication with an invertible upper triangular matrix.*
2. *the column rank profile is preserved by left multiplication with an invertible matrix and by right multiplication with an invertible lower triangular matrix.*

LEMMA 2. *Let $A = PLUQ$ be the PLUQ decomposition computed by Algorithm 1. Then the row (resp. column) rank profile of any leading (k,t) submatrix of A is the row (resp. column) rank profile of the leading (k,t) submatrix of $P\begin{bmatrix}I_r\\0\end{bmatrix}Q$.*

PROOF. With the notations of Corollary 1, we have:

$$A = P\begin{bmatrix}L\\MI_{m-r}\end{bmatrix}\begin{bmatrix}I_r\\0\end{bmatrix}\begin{bmatrix}U\ V\\ \ \ I_{n-r}\end{bmatrix}Q = \overline{L}P\begin{bmatrix}I_r\\0\end{bmatrix}Q\overline{U}$$

Hence

$$[I_k 0]\,A\begin{bmatrix}I_t\\0\end{bmatrix} = \overline{L_1}\,[I_k 0]\,P\begin{bmatrix}I_r\\0\end{bmatrix}Q\overline{U_1},$$

where $\overline{L_1}$ is the $k\times k$ leading submatrix of $\overline{L}$ (hence it is an invertible lower triangular matrix) and $\overline{U_1}$ is the $t\times t$ leading submatrix of $\overline{U}$ (hence it is an invertible upper triangular matrix). Now, Lemma 1 implies that the rank profile of $[I_k 0]\,A\begin{bmatrix}I_t\\0\end{bmatrix}$ is that of $[I_k 0]\,P\begin{bmatrix}I_r\\0\end{bmatrix}Q\begin{bmatrix}I_t\\0\end{bmatrix}$. □

From this lemma we deduce how to compute the row and column rank profiles of any (k,t) leading submatrix and more particularly of the matrix A itself.

COROLLARY 2. *Let $A = PLUQ$ be the PLUQ decomposition of a $m \times n$ matrix computed by Algorithm 1. The row (resp. column) rank profile of any (k,t)-leading submatrix of a A is the sorted sequence of the row (resp. column) indices of the non zero rows (resp. columns) in the matrix*

$$R = [I_k 0]\, P \begin{bmatrix} I_r \\ 0 \end{bmatrix} Q \begin{bmatrix} I_t \\ 0 \end{bmatrix}$$

COROLLARY 3. *The row (resp. column) rank profile of A is the sorted sequence of row (resp. column) indices of the non zero rows (resp. columns) of the first r columns of P (resp. first r rows of Q).*

5. COMPLEXITY ANALYSIS

We study here the time complexity of Algorithm 1 by counting the number of field operations. For the sake of simplicity, we will assume here that the dimensions m and n are powers of two. The analysis can easily be extended to the general case for arbitrary m and n.

For $i = 1, 2, 3, 4$ we denote by T_i the cost of the i-th recursive call to PLUQ, on a $\frac{m}{2} \times \frac{n}{2}$ matrix of rank r_i. We also denote by $T_{\text{TRSM}}(m,n)$ the cost of a call TRSM on a rectangular matrix of dimensions $m \times n$, and by $T_{\text{MM}}(m,k,n)$ the cost of multiplying an $m \times k$ by an $k \times n$ matrix.

THEOREM 2. *Algorithm 1, run on an $m \times n$ matrix of rank r, performs $O\left(mnr^{\omega-2}\right)$ field operations.*

PROOF. Let $T = T_{\text{PLUQ}}(m,n,r)$ be the cost of Algorithm 1 run on a $m \times n$ matrix of rank r. From the complexities of the subroutines given, e.g., in [2] and the recursive calls in Algorithm 1, we have:

$$\begin{aligned}
T=&T_1 + T_2 + T_3 + T_4 + T_{\text{TRSM}}(r_1, \frac{m}{2}) + T_{\text{TRSM}}(r_1, \frac{n}{2}) \\
&+T_{\text{TRSM}}(r_2, \frac{m}{2}) + T_{\text{TRSM}}(r_3, \frac{n}{2}) + T_{\text{MM}}(\frac{m}{2} - r_1, r_1, \frac{n}{2}) \\
&+T_{\text{MM}}(\frac{m}{2}, r_1, \frac{n}{2} - r_1) + T_{\text{MM}}(\frac{m}{2}, r_1, \frac{n}{2}) \\
&+T_{\text{MM}}(r_3, r_2, \frac{n}{2} - r_2) + T_{\text{MM}}(\frac{m}{2} - r_3, r_2, \frac{n}{2} - r_2 - r_4) \\
&+T_{\text{MM}}(\frac{m}{2} - r_3, r_3, \frac{n}{2} - r_2 - r_4) \\
\leq&T_1 + T_2 + T_3 + T_4 + K\left(\frac{m}{2}(r_1^{\omega-1} + r_2^{\omega-1}) + \frac{n}{2}(r_1^{\omega-1}\right. \\
&\left.+r_3^{\omega-1}) + \frac{m}{2}\frac{n}{2}r_1^{\omega-2} + \frac{m}{2}\frac{n}{2}r_2^{\omega-2} + \frac{m}{2}\frac{n}{2}r_3^{\omega-2}\right) \\
\leq&T_1 + T_2 + T_3 + T_4 + K' mnr^{\omega-2}
\end{aligned}$$

for some constants K and K' (we recall that $a^{\omega-2}+b^{\omega-2} \leq 2^{3-\omega}(a+b)^{\omega-2}$ for $2 \leq \omega \leq 3$).

Let $C = max\{\frac{K'}{1-2^{4-2\omega}}; 1\}$.Then we can prove by a simultaneous induction on m and n that $T \leq Cmnr^{\omega-2}$.

Indeed, if $(r = 1, m = 1, n \geq m)$ or $(r = 1, n = 1, m \geq n)$ then $T \leq m - 1 \leq Cmnr^{\omega-2}$. Now if it is true for $m = 2^j, n = 2^i$, then for $m = 2^{j+1}, n = 2^{i+1}$, we have

$$\begin{aligned}
T\leq&\frac{C}{4}mn(r_1^{\omega-2} + r_2^{\omega-2} + r_3^{\omega-2} + r_4^{\omega-2}) + K' mnr^{\omega-2} \\
\leq&\frac{C(2^{3-\omega})^2}{4}mnr^{\omega-2} + K' mnr^{\omega-2} \\
\leq&K'\frac{2^{4-2\omega}}{1-2^{4-2\omega}}mnr^{\omega-2} + K' mnr^{\omega-2} \leq Cmnr^{\omega-2}.
\end{aligned}$$

□

In order to compare this algorithm with usual Gaussian elimination algorithms, we now refine the analysis to compare the leading constant of the time complexity in the special case where the matrix is square and has a generic rank profile: $r_1 = \frac{m}{2} = \frac{n}{2}, r_2 = 0, r_3 = 0$ and $r_4 = \frac{m}{2} = \frac{n}{2}$ at each recursive step.

Hence, with C_ω the constant of matrix multiplication, we have

$$\begin{aligned}
T_{\text{PLUQ}}=&2T_{\text{PLUQ}}(\frac{n}{2}, \frac{n}{2}, \frac{n}{2}) + 2T_{\text{TRSM}}(\frac{n}{2}, \frac{n}{2}) + T_{\text{MM}}(\frac{n}{2}, \frac{n}{2}, \frac{n}{2}) \\
=&2T_{\text{PLUQ}}(\frac{n}{2}, \frac{n}{2}, \frac{n}{2}) + 2\frac{C_\omega}{2^{\omega-1}-2}\left(\frac{n}{2}\right)^\omega + C_\omega\left(\frac{n}{2}\right)^\omega
\end{aligned}$$

Writing $T_{\text{PLUQ}}(n,n,n) = \alpha n^\omega$, the constant α satisfies:

$$\alpha = C_\omega\frac{1}{(2^\omega - 2)}\left(\frac{1}{2^{\omega-2}-1} + 1\right) = C_\omega\frac{2^{\omega-2}}{(2^\omega-2)(2^{\omega-2}-1)}.$$

which is equal to the constant of the CUP and LUP decompositions [7, Table 1]. In particular, it equals 2/3 when $\omega = 3, C_\omega = 2$, matching the constant of the classical Gaussian elimination.

6. COUNTING MODULAR REDUCTIONS OVER A PRIME FIELD

In the following we suppose that the operations are done with full delayed reduction for a single multiplication and any number of additions: operations of the form $\sum a_i b_i$ are reduced only once at the end of the addition, but $a \cdot b \cdot c$ requires two reductions. In practice, only a limited number of accumulations can be done on an actual mantissa without overflowing, but we neglect this in this section for the sake of simplicity. See e.g. [2] for more details. For instance, with this model, the number of reductions required by a classic multiplication of matrices of size $m \times k$ by $k \times n$ is simply: $m \cdot n$. We denote this by $R_{MM}(m,k,n) = mn$. This extends e.g. also for triangular solving:

THEOREM 3. *Over a prime field modulo p, the number of reductions modulo p required by $TRSM(m,n)$ with full delayed reduction is:*

$$\begin{aligned}
R_{UnitTRSM}(m,n)=&mn \quad \textit{if the triangular matrix is unitary,} \\
R_{TRSM}(m,n) \quad =&2mn \textit{ in general.}
\end{aligned}$$

PROOF. If the matrix is unitary, then a fully delayed reduction is required only once after the update of each row of the result. In the generic case, we invert each diagonal element first and multiply each element of the right hand side by this inverse diagonal element, prior to the update of each row of the result. This gives mn extra reductions. □

Next we show that the new pivoting strategy is more efficient in terms of number of integer division.

THEOREM 4. *Over a prime field modulo p and on a full-rank square $m \times m$ matrix with generic rank profile, and m a power of two, the number of reductions modulo p required by the elimination algorithms with full delayed reduction is:*

$$\begin{aligned} R_{PLUQ}(m,m) &= 2m^2 + o\left(m^2\right), \\ R_{PLE}(m,m) = R_{CUP}(m,m) &= \left(1 + \tfrac{1}{4}\log_2(m)\right) m^2 + o\left(m^2\right) \end{aligned}$$

PROOF. If the top left square block is full rank then PLUQ reduces to one recursive call, two square TRSM (one unitary, one generic) one square matrix multiplication and a final recursive call. In terms of modular reductions, this gives: $R_{\text{PLUQ}}(m) = 2R_{\text{PLUQ}}(\frac{m}{2}) + R_{\text{UnitTRSM}}(\frac{m}{2},\frac{m}{2}) + R_{\text{TRSM}}(\frac{m}{2},\frac{m}{2}) + R_{\text{MM}}(\frac{m}{2},\frac{m}{2},\frac{m}{2})$. Therefore, using Theorem 3, the number of reductions within PLUQ satisfies $T(m) = 2T(\frac{m}{2}) + m^2$ so that it is $R_{PLUQ}(m,m) = 2m^2 - 2m$ if m is a power of two.

For row or column oriented elimination this situation is more complicated since the recursive calls will always be rectangular even if the intermediate matrices are full-rank. We in fact prove, by induction on m, the more generic:

$$R_{\text{PLE}}(m,n) = \log_2(m)(\frac{mn}{2} - \frac{m^2}{4}) + m^2 + o\left(mn + m^2\right) \quad (1)$$

First $R_{\text{PLE}}(1,n) = 0$ since $[1] \times [a_1, \ldots, a_n]$ is a triangular decomposition of the $1 \times n$ matrix $[a_1, \ldots, a_n]$. Now suppose that Equation (1) holds for $k = m$. Then we follow the row oriented algorithm of [2, Lemma 5.1] which makes two recursive calls, one TRSM and one MM to get $R_{\text{PLE}}(2m,n) = R_{\text{PLE}}(m,n) + R_{\text{PLE}}(m,m) + R_{\text{MM}}(m,m,n-m) + R_{\text{PLE}}(m,n-m) = R_{\text{PLE}}(m,n) + R_{\text{PLE}}(m,n-m) + m(n+m)$. We then apply the induction hypothesis on the recursive calls to get

$$\begin{aligned} R_{\text{PLE}}(2m,n) =& \frac{1}{2}\log_2(m)mn - \frac{1}{4}\log_2(m)m^2 + m^2 + \\ & \frac{1}{2}\log_2(m)m(n-m) - \frac{1}{4}\log_2(m)m^2 + m^2 + \\ & m(n+m) + o\left(mn + m^2\right) \\ =& \log_2(m)(mn - m^2) + 3m^2 + mn + o\left(mn + m^2\right). \end{aligned}$$

The latter is also obtained by substituting $m \hookleftarrow 2m$ in Equation (1) so that the induction is proven. □

This shows that the new algorithm requires fewer modular reductions, as soon as m is larger than 32. Over finite fields, since reductions can be much more expensive than multiplications or additions by elements of the field, this is a non negligible advantage. We show in Section 8 that this participates to the better practical performance of the PLUQ algorithm.

7. A BASE CASE ALGORITHM

We propose in Algorithm 2 an *iterative* algorithm computing the same PLUQ decomposition as Algorithm 1. The motivation is to offer an alternative to the recursive algorithm improving the computational efficiency on small matrix sizes. Indeed, as long as the matrix fits the cache memory, the number of page faults of the two variants are similar, but the iterative variant reduces the number of row and column permutations. The block recursive algorithm can then be modified so that it switches to the iterative algorithm whenever the matrix dimensions are below some threshold.

Unlike the common Gaussian elimination, where pivots are searched in the whole current row or column, the strategy is here to proceed with an incrementally growing leading sub-matrix. This implies a Z-curve type search scheme, as shown on Figure 2. This search strategy is meant to ensure the properties on the rank profile that have been presented in Section 4.

Algorithm 2 PLUQ iterative base case

Input: A a $m \times n$ matrix over a field
Output: P, Q: $m \times m$ and $n \times n$ permutation matrices
Output: r: the rank of A
Output: $A \leftarrow \begin{bmatrix} L\backslash UV \\ M \quad 0 \end{bmatrix}$ where L is $r \times r$ unit lower triang., U is $r \times r$ upper triang. and such that $A = P \begin{bmatrix} L \\ M \end{bmatrix} [UV]\, Q$.

```
r ← 0; i ← 0; j ← 0
while i < m or j < n do
    ▷ Let v = [A_{i,r} ... A_{i,j-1}] and w = [A_{r,j} ... A_{i-1,r}]^T
    if j < n and w ≠ 0 then
        p ← row index of the first non zero entry in w
        q ← j; j ← max(j + 1, n)
    else if i < m and v ≠ 0 then
        q ← column index of the first non zero entry in v
        p ← i; i ← max(i + 1, m)
    else if i < m and j < n and A_{i,j} ≠ 0 then
        (p, q) ← (i, j)
        i ← max(i + 1, m); j ← max(j + 1, n)
    else
        i ← max(i + 1, m); j ← max(j + 1, n)
        continue
    end if                          ▷ At this stage, A_{p,q} is a pivot
    for k = p + 1 : n do
        A_{k,q} ← A_{k,p} A_{p,q}^{-1}
        A_{k,q+1:n} ← A_{k,q+1:n} − A_{k,q} A_{p,q+1:n}
    end for
                    ▷ Cyclic shifts of pivot column and row
    A_{0:m,r:q} ← A_{0:m,r>>>1 q}
    A_{r:p,0:n} ← A_{r>>>1 p,0:n}
    P ← P_{r>>>1 p,*};
    Q ← Q_{*,r>>>1 q}
    r ← r + 1
end while
```

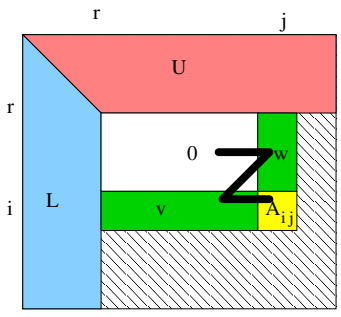

Figure 2: Iterative base case PLUQ decomposition

In order to perform the correct updates on the remaining parts, when a pivot is found its whole row and column have to be permuted to the current diagonal location, see Figure 2. But then, in order to preserve the row and column rank profiles, all the rows and column in between have to be

shifted by 1 location. Therefore after the elimination step, the rows and columns of the matrix, as well as the rows of the left permutation matrix and the columns of the right permutation matrix have to be cyclically shifted accordingly. This is presented in the last steps of Algorithm 2, where the notation $A_{*,i>>>_1 j}$ means that in matrix A, columns i through j, both inclusive, have to be shifted by 1 location, cyclically to the right.

REMARK 3. *Applying the cyclic permutations in steps 22 to 25 may cost in worst case a cubic number of operations. Instead one can delay these permutations and leave the pivots at the position where they were found. These positions are then used to form the matrices P and Q, only after the end of the* `while` *loop. Then applying these permutations to the current matrix gives the final decomposition* $\begin{bmatrix} L\backslash U & V \\ M & 0 \end{bmatrix}$.

REMARK 4. *In order to further improve the data locality, this iterative algorithm can be transformed into a left-looking variant [1]. Over a finite field, this variant performs fewer modular operations: Step 19 of Algorithm 2 requires a modular reduction after each multiplication while a left-looking variant will delay these reductions within block operations.*

Updating Algorihtm 2 with Remarks 3 and 4 would be too technical to be presented here, but this is how we implemented the base case used for the experiments of Section 8.

8. EXPERIMENTS

Algorithm 1 combined with the base case Algorithm 2 has been implemented in the `FFLAS-FFPACK` library[1] and is available from revision svn@361. We present here experiments comparing its efficiency with the implementation of the CUP/PLE decomposition, called `LUdivine` in this same library. We ran our tests on a single core of an Intel Xeon E5-4620@2.20GHz using gcc-4.7.2.

In Figure 3, the matrices are dense, with full rank. The computation times are similar, the PLUQ algorithm with base case being slightly faster than `LUdivine`. In Figures 4

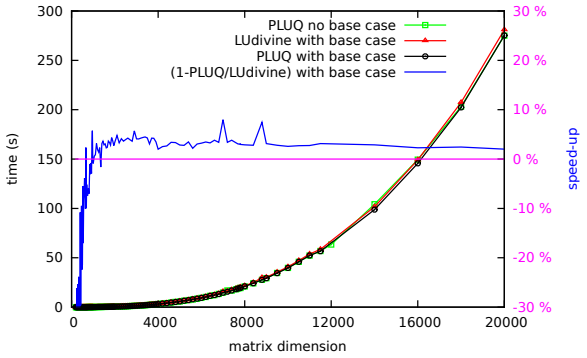

Figure 3: Dense full rank matrices modulo 1009

and 5, the matrices are square, dense with a rank equal to half the dimension. To ensure non trivial row and column rank profiles, they are generated from a LEU decomposition, where L and U are uniformly random non-singular lower and upper triangular matrices, and E is zero except on $r = n/2$ positions, chosen uniformly at random, set to one. The cutoff dimension for the switch to the base case has been set

[1] `http://linalg.org/projects/fflas-ffpack`

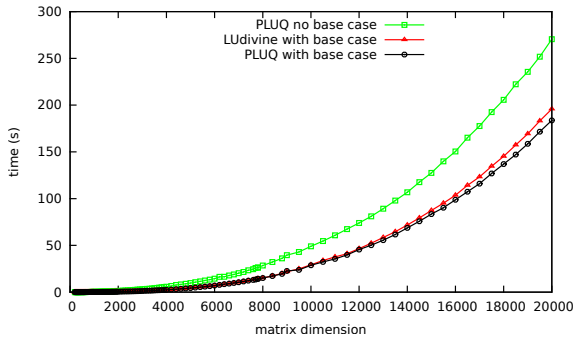

Figure 4: Computation time with dense rank deficient matrices (rank is half the dimension)

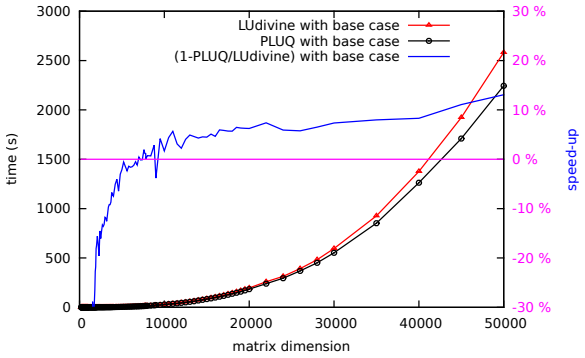

Figure 5: Computation time with dense rank deficient matrices of larger dimension (half rank)

to an optimal value of 288 by experiments. Figure 4 shows how the base case greatly improves the efficiency for PLUQ, presumably for it reduces the number of row and column permutations. More precisely, PLUQ becomes faster than `LUdivine` for dimensions above 7000. Figure 5 shows that, on larger matrices, PLUQ can be up to 13% faster.

Table 1 shows the cache misses reported by the callgrind tool (valgrind emulator version 3.8.1). We also report in the last column the corresponding computation time (without emulator). We used the same matrices as in Figure 4, with rank half the dimension. We first notice the impact of the base case on the PLUQ algorithm: although it does not change the number of cache misses, it strongly reduces the total number of memory accesses (fewer permutations), thus improving the computation time. Now as the dimension grows, the amount of memory accesses and of cache misses plays in favor of PLUQ which becomes faster than `LUdivine`.

9. CONCLUSION AND PERSPECTIVES

We showed the first reduction to matrix multiplication of the problem of computing both row and column rank profiles of all leading sub-matrices of an input matrix.

The decomposition that we propose can first be viewed as an improvement over the LEU decomposition, introducing a finer treatment of rank deficiency that reduces the number of arithmetic operations, makes the time complexity rank sensitive and allows to perform the computation in-place.

Second, viewed as a variant of the existing CUP/PLE decompositions, this new algorithm produces more information on the rank profile and is more efficient, as it deals with ma-

Matrix	Algorithm	Accesses	L1 Misses	L3 Misses	L3/Accesses	Timing (s)
A4K	`LUdivine`	1.529E+10	1.246E+09	2.435E+07	.159	**2.31**
	PLUQ-no-base-case	1.319E+10	**7.411E+08**	1.523E+07	**.115**	5.82
	PLUQ-base-case	**8.105E+09**	7.467E+08	**1.517E+07**	.187	2.48
A8K	`LUdivine`	7.555E+10	9.693E+09	2.205E+08	.292	15.2
	PLUQ-no-base-case	6.150E+10	**5.679E+09**	1.305E+08	**.212**	28.4
	PLUQ-base-case	**4.067E+10**	5.686E+09	**1.303E+08**	.321	**15.1**
A12K	`LUdivine`	2.003E+11	3.141E+10	7.943E+08	.396	46.5
	PLUQ-no-base-case	1.575E+11	**1.911E+10**	4.691E+08	**.298**	73.9
	PLUQ-base-case	**1.111E+11**	1.913E+10	**4.687E+08**	.422	**45.5**
A16K	`LUdivine`	4.117E+11	7.391E+10	1.863E+09	.452	103
	PLUQ-no-base-case	3.142E+11	4.459E+10	1.092E+09	**.347**	150
	PLUQ-base-case	**2.299+11**	**4.458E+10**	**1.088E+09**	.473	**98.8**

Table 1: Cache misses for dense matrices with rank equal half of the dimension

trices of more evenly balanced dimensions. It also performs fewer modular reductions when computing over a finite field.

Overall the new algorithm is also faster in practice than previous implementations with large enough matrices.

Lastly, it also exhibits more parallelism than classical Gaussian elimination since the recursive calls in step 2 and 3 are independent. This is also the case for the TURBO algorithm of [3], but it has a higher arithmetic complexity. Further experiments and analysis of communication costs should be done in both shared and distributed memory settings.

10. ACKNOWLEDGEMENT

We are gratefull to Jean-Louis Roch and an anonymous referee for their helpful remarks and suggestions.

11. REFERENCES

[1] J. J. Dongarra, L. S. Duff, D. C. Sorensen, and H. A. V. Vorst. *Numerical Linear Algebra for High Performance Computers*. SIAM, 1998.

[2] J.-G. Dumas, P. Giorgi, and C. Pernet. Dense linear algebra over prime fields. *ACM TOMS*, 35(3):1–42, Nov. 2008. URL: http://arxiv.org/abs/cs/0601133.

[3] J.-G. Dumas and J.-L. Roch. On parallel block algorithms for exact triangularizations. *Parallel Computing*, 28(11):1531–1548, Nov. 2002.

[4] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, June 1999. URL: http://www-salsa.lip6.fr/~jcf/Papers/F99a.pdf.

[5] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[6] O. H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. of Algorithms*, 3(1):45–56, Mar. 1982.

[7] C.-P. Jeannerod, C. Pernet, and A. Storjohann. Rank profile revealing Gaussian elimination and the CUP matrix decomposition. *Journal of Symbolic Computations*, 2013. To appear. report, arXiV cs.SC/1112.5717.

[8] D. J. Jeffrey. LU factoring of non-invertible matrices. *ACM Comm. Comp. Algebra*, 44(1/2):1–8, July 2010. URL: http://www.apmaths.uwo.ca/~djeffrey/Offprints/David-Jeffrey-LU.pdf.

[9] G. I. Malaschonok. Fast generalized Bruhat decomposition. In *CASC'10*, volume 6244 of *LNCS*, pages 194–202. Springer-Verlag, Berlin, Heidelberg, 2010.

[10] W. Stein. *Modular forms, a computational approach*. Graduate studies in mathematics. AMS, 2007. URL: http://wstein.org/books/modform/modform.

[11] A. Storjohann. *Algorithms for Matrix Canonical Forms*. PhD thesis, ETH-Zentrum, Zürich, Switzerland, Nov. 2000. doi:10.3929/ethz-a-004141007.

[12] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC'12*, pages 887–898, New York, NY, USA, 2012. ACM. URL: http://www.cs.berkeley.edu/~virgi/matrixmult.pdf.

APPENDIX

A. CORRECTNESS OF ALGORITHM 1

First note that $S\begin{bmatrix}L\\M\end{bmatrix} = \left[\begin{array}{ccc|c} L_1 & & & \\ M_{11} & L_2 & & \\ M_{12} & M_2 & 0 & \\ \hline E_1 & I & L_3 & \\ E_{21} & K_1 & M_{31} & L_4 \\ E_{22} & K_2 & M_{32} & M_4 0 0 \end{array}\right]$

Hence $P\begin{bmatrix}L\\M\end{bmatrix} = \begin{bmatrix}P_1 & \\ & P_3\end{bmatrix}\left[\begin{array}{ccc} L_1 & & \\ M_1 P_2 & \begin{bmatrix}L_2\\M_2\end{bmatrix} & \\ \hline E_1 & I & L_3 \\ E_2 & K & M_3 P_4 \begin{bmatrix}L_4\\M_4\end{bmatrix} \end{array}\right]$

Similarly, $[UV]T = \left[\begin{array}{ccc|ccc} U_1 V_{11} & V_{12} & & D_1 D_{21} & D_{22} \\ 0 & 0 & & U_2 V_{21} & V_{22} \\ U_3 & V_3 & & 0 \; O_1 & O_2 \\ & & & U_4 & V_4 \\ & & & & 0 \end{array}\right]$ and $[UV]Q = \left[\begin{array}{cc|cc} U_1 & V_1 & D_1 & D_2 \\ & 0 & U_2 & V_2 \\ & [U_3 V_3] Q_3 & 0 & O \\ & & & [U_4 V_4] Q_4 \end{array}\right]\begin{bmatrix}Q_1 & \\ & Q_2\end{bmatrix}$.

Now as $H_1 = IU_2, H_2 = IV_2 + L_3 O, H_3 = KU_2$ and $H_4 = KV_2 + M_3 O + P_4\begin{bmatrix}L_4\\M_4\end{bmatrix}[U_4 V_4] Q_4$ we have

$$
\begin{aligned}
P\begin{bmatrix}L\\M\end{bmatrix}[UV]Q =& \begin{bmatrix}P_1 & \\ & P_3\end{bmatrix}\left[\begin{array}{ccc} L_1 & & \\ M_1 P_2 & \begin{bmatrix}L_2\\M_2\end{bmatrix} & \\ \hline E_1 & I & L_3 \\ E_2 & K & M_3 P_4 \begin{bmatrix}L_4\\M_4\end{bmatrix} \end{array}\right] \\
& \left[\begin{array}{cc|cc} U_1 & V_1 & D_1 & D_2 \\ & 0 & U_2 & V_2 \\ & [U_3 V_3] Q_3 & 0 & O \\ & & & [U_4 V_4] Q_4 \end{array}\right]\begin{bmatrix}Q_1 & \\ & Q_2\end{bmatrix} \\
=& \begin{bmatrix}P_1 & \\ & P_3\end{bmatrix}\left[\begin{array}{ccc} L_1 & & \\ M_1 P_2 & \begin{bmatrix}L_2\\M_2\end{bmatrix} & \\ \hline E_1 & & I_{r_3} \\ E_2 & & I_{m-k-r_3} \end{array}\right] \\
& \left[\begin{array}{cc|cc} U_1 & V_1 & D_1 & D_2 \\ & 0 & U_2 & V_2 \\ L_3 & [U_3 V_3] Q_3 & H_1 & H_2 \\ M_3 & [U_3 V_3] Q_3 & H_3 & H_4 \end{array}\right]\begin{bmatrix}Q_1 & \\ & Q_2\end{bmatrix} \\
=& \begin{bmatrix}P_1 & \\ & I_{m-k}\end{bmatrix}\left[\begin{array}{cc} L_1 & \\ M_1 & \\ \hline E & 0 I_{m-k} \end{array}\right]\left[\begin{array}{cc|c} U_1 V_1 & & D \\ 0 & & F \\ & G & H \end{array}\right] \\
& \begin{bmatrix}Q_1 & \\ & I_{n-k}\end{bmatrix} \\
=& \begin{bmatrix}P_1 & \\ & I_{m-k}\end{bmatrix}\left[\begin{array}{cc|c} L_1 U_1 & L_1 V_1 & B_1 \\ M_1 U_1 & M_1 V_1 & B_2 \\ \hline C_1 & C_2 & A_4 \end{array}\right]\begin{bmatrix}Q_1 & \\ & I_{n-k}\end{bmatrix} \\
=& A
\end{aligned}
$$