A parallel block algorithm for exact triangularization of rectangular matrices

Jean-Guillaume Dumas   Jean-Louis Roch

Laboratoire Informatique et Distribution, ENSIMAG - antenne de Montbonnot. ZIRST - 51, av. Jean Kuntzmann, 38330 Montbonnot Saint-Martin, France.

{Jean-Guillaume.Dumas,Jean-Louis.Roch}@imag.fr; www-id.imag.fr/{~jgdumas,~jlroch}

ABSTRACT

A new block algorithm for triangularization of regular or singular matrices with dimension $m \times n$ is proposed. Taking benefit of fast block multiplication algorithms, it achieves the best known sequential complexity $O(m^{\omega-1}n)$ for any sizes and any rank. Moreover, the block strategy enables to improve locality with respect to previous algorithms as exhibited by practical performances.

1. INTRODUCTION

In this article, we study the parallelization of the exact LU factorization of matrices with arbitrary field elements. The matrix can be *singular* or even *rectangular*. Our main purpose is to compute the rank of large matrices. Therefore, we relax the conditions on L in order to obtain an *in place* TU factorization, where U is upper triangular as usual and T is block sparse (with some "T" patterns). Exact triangularization arises especially in computer algebra. For instance, one of the main tools for solving algebraic systems is the computation of Gröbner bases and to compute such standard bases one uses modular triangularization of large sparse rectangular matrices. Among other applications are combinatorics, fast determinant computation, Diophantine analysis, group theory and algebraic topology via the computation of the integer Smith normal form.

A first idea is to use a parallel direct method on matrices stored by rows (respectively columns). There, at stage k of the classical Gaussian elimination algorithm, eliminations are executed in parallel on the $n-k-1$ remaining rows; thus giving only a relatively small grain. The next idea is therefore to mimic numerical methods and use sub-matrices. Now, the problem is that usually, for symbolic computation, *these blocks are singular*. This fact prevents us from using classical numerical recursive blocked data formats [2], for instance. To solve this problem one has mainly two alternatives. One is to perform a dynamic cutting of the matrix and to adjust it so that the blocks are reduced and become invertible. Such a method is shown by Ibarra et al. in [3]. A recursive process is then used to perform the rank of the first region. Then, depending on this rank, the cutting is modified and the algorithm pursues with a new region. This way, Ibarra et al. were able to build the first algorithm computing the rank of an $m \times n$ matrix with arithmetic complexity $\mathcal{O}(m^{\omega-1}n)$, where the complexity of matrix multiplication is $O(m^{\omega})$. Unfortunately, their method is not so efficient in parallel: it induces synchronizations and significant communications at each stage in order to compute the block redistribution.

We therefore propose another method, called $TURBO$, using *singular static blocks* in order to avoid these synchronizations and redistributions. Our algorithm has also an *optimal sequential arithmetic complexity* and is able to *avoid 30%* of the communications.

2. A NEW BLOCK ALGORITHM: TURBO

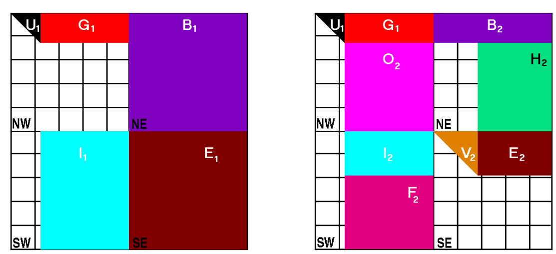

Figure 1: Step 1. Recursive TU triangularization in NW – Step 2. Recursive TU triangularization in SE

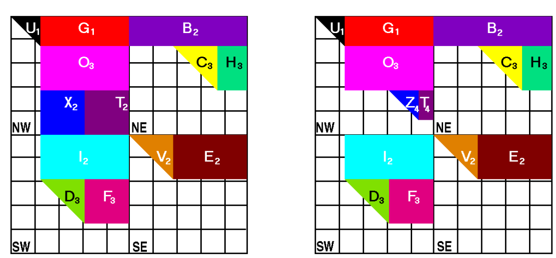

Figure 2: Step 3. Parallel recursive TU in SW and NE – Step 4. Small recursive TU in NW again

In $TURBO$, the elementary operation is a block operation and not a scalar one. In addition, the cutting of the matrix in blocks is carried out before the execution of the algorithm and is not modified, in order to limit the volume of communications. Our method recursively divides the matrix into four regions: $A = \left[\begin{smallmatrix} NW & NE \\ SW & SE \end{smallmatrix}\right]$. Local TU factorizations are then performed on each block. The method is applied recursively until the region size reaches a given threshold. We show here the algorithm for only one iteration. The factorization is done *in place*, i.e. the matrix A in input is not copied. The algorithm modifies its elements progressively as shown in figures 1 and 2. Next, virtually performing row and column permutations, one can easily see that rank(A) = $\text{rank}(U_1) + \text{rank}(V_2) + \text{rank}(C_3) + \text{rank}(D_3) + \text{rank}(Z_4)$.

3. ARITHMETIC COMPLEXITY

Let ω be the exponent of the arithmetic cost of matrix multiplication ($\omega \in [2,3]$ depending on the algebraic structure [1]). For the sake of simplicity, we will bound the cost of sequential multiplication of two $m \times n$ and $n \times l$ matrices by $M(h) = \mathcal{O}(h^\omega)$ where $h = max\{m; n; l\}$. We also consider that parallel triangular matrix multiplication and inversion costs are logarithmic (lower than $K_\infty \log_2^2(h)$).

Theorem 1. *Let $T_1(h)$ and $T_\infty(h)$ be the respsective sequential and parallel arithmetic complexity of our algorithm for a rectangular matrix of higher dimension h. Then,*

$$T_1(h) \leq 7.25M(h) + 2h^2 = \mathcal{O}(h^\omega)$$

$$T_\infty(h) \leq 3K_\infty h = \mathcal{O}(h).$$

Therefore, the arithmetic complexity of our algorithm is identical to the best known complexity for this problem [3, Theorem 2.1]. Unlike our method, Ibarra's algorithm groups *rows* into two regions. Then, depending on the rank of the first region, the matrix structure is *modified*. Using our block cutting, we instead guarantee that all the accesses are local. Also, the theoretical parallel complexity is linear, while rank computation is in NC^2: using $\mathcal{O}(n^{4.5})$ processors, the computation can be performed with parallel time $\mathcal{O}(\log_2^2(n))$ [4]. However, the best known parallel algorithm with optimal sequential time also achieve a parallel linear time [3, 1]. But, in practice, our technique is more interesting as it preserves locality. Further, we will see that it reduces the volume of communications on distributed architectures.

4. PRACTICAL COMMUNICATION PERFORMANCES

In this section, we compare the communication volumes between the row and block strategies. We now estimate the gain with our algorithm for a rectangular matrix $2m \times 2n$ of rank $r \leq min\{2m, 2n\}$. We denote by q, p, c, d and z the respective ranks of U_1, V_2, C_3, D_3 and Z_4 (then $r = q + p + c + d + z$). In the worst case, for only one phase (no recursion, only the steps previously shown), on 4 processors (one for each region), the total volume obtained is already quite complex: $C(2m, 2n, r, 4) = mn + 2qm + 2pn + q^2 + pm + dn - pq - dq$. In order to give a more precise idea of the gain of our method, we compare this result to the volume of communications obtained by row: $L(2m, 2n, r, P) = \sum_{k=1}^{r}(P-1)(2n-k) = r(2n - \frac{r+1}{2})(P-1)$. Next, table 1 shows the gain obtained with the previously introduced matrices. The total effective communicated volumes of both (row and $TURBO$) methods are compared. These matrices are quite sparse. Unfortunately, the first

Matrix	$2m \times 2n$	r	$\rho = \frac{L-C}{L}$
ch5-5.b2	600x200	176	-57.97%
mk9.b2	1260x378	343	-67.36%
ch6-6.b2	2400x450	415	-123.66%
ch4-4.b2	96x72	57	10.40%
ch5-5.b3	600x600	424	32.80%
mk9.b3	945x1260	875	11.80%
robot24_m5	404x302	262	9.08%
rkat7_m5	694x738	611	34.02%
f855_m9	2456x2511	2331	34.68%
cyclic8_m11	4562x5761	3903	21.02%

Table 1: Communication volume gain

version of our algorithm is implemented only for dense matrices. Still, we can see that our method is able to avoid some communications as soon as the matrices are not too special. In the table, the first three matrices are very unbalanced (very small number of columns compared to the number of rows): in that case a row method can be much more efficient since it can communicate only the smallest dimension. However, in all the other cases we are able to achieve very good performances: for the less rectangular matrices, we have a gain ρ very close to 33% in general.

5. CONCLUSIONS

To conclude, we developed a new block TU elimination algorithm. Its theoretical sequential and parallel arithmetic complexities are similar to those of the most efficient current elimination algorithms for this problem. Besides, it is particularly adapted to the singular matrices and makes it possible to compute the rank in an exact way. Furthermore, it allows a more flexible management of the scheduling (adaptive grain) and avoids a third of the communications when used with only one level of recursion on 4 processors. In addition, if the increase in locality reduces the number of communications, it also makes it possible to increase the speed by a greater benefit of the cache effects.

Lastly, there remains to study an effective parallel method to compute the TU factorization for sparse matrices. Indeed, designing an efficient *block reordering* technique seems to be an important open question.

6. REFERENCES

[1] D. Bini and V. Pan. *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms.* Birkhauser, Boston, 1994.

[2] F. Gustavson, A. Henriksson, I. Jonsson, and B. Kaagstroem. Recursive blocked data formats and BLAS's for dense linear algebra algorithms. *Lecture Notes in Computer Science*, 1541:195–206, 1998.

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

[4] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix. *Combinatorica*, 7(1):101–104, 1987.