Some efficient methods for computing the determinant of large sparse matrices

E. KAMGNIA(1) — L. B. NGUENANG(1)

(1) Department of Computer Science, University of Yaounde I, Yaounde, Cameroon.
Emails: erkamgnia@yahoo.fr, lnguenang@yahoo.fr

ABSTRACT. The computation of determinants intervenes in many scientific applications, as for example in the localization of eigenvalues of a given matrix $A$ in a domain of the complex plane. When a procedure based on the application of the residual theorem is used, the integration process leads to the evaluation of the principal argument of the complex logarithm of the function $g(z) = \frac{\det((z + h)I - A)}{\det(zI - A)}$, and a large number of determinants is computed to insure that the same branch of the complex logarithm is followed during the integration. In this paper, we present some efficient methods for computing the determinant of a large sparse and block structured matrix. Tests conducted using randomly generated matrices show the efficiency and robustness of our methods.

RÉSUMÉ. Le calcul de déterminants intervient dans certaines applications scientifiques, comme par exemple dans le comptage du nombre de valeurs propres d’une matrice situées dans un domaine borné du plan complexe. Lorsqu’on utilise une approche fondée sur l’application du théorème des résidus, l’intégration nous ramène à l’évaluation de l’argument principal du logarithme complexe de la fonction $g(z) = \frac{\det((z + h)I - A)}{\det(zI - A)}$, en un grand nombre de points, pour ne pas sauter d’une branche à l’autre du logarithme complexe. Nous proposons dans cet article quelques méthodes efficaces pour le calcul du déterminant d’une matrice grande et creuse, et qui peut être transformée sous forme de blocs structurés. Les résultats numériques, issus de tests sur des matrices générées de façon aléatoire, confirment l’efficacité et la robustesse des méthodes proposées.

KEYWORDS : Determinant, eigenvalues, LU factorization, characteristic polynomial, Schur complement, SPIKE

MOTS-CLÉS : Déterminants, valeurs propres, polynôme caractéristique, factorisation LU, complément de Schur, SPIKE
1. Introduction

The computation of determinants is needed in scientific applications. One such application is the localisation of eigenvalues of a given matrix \( A \in \mathbb{R}^{n \times n} \) in a domain of the complex plane. When one proceeds as in [2, 3] by applying the residue theorem, the number \( N_\Gamma \) of complex eigenvalues which are surrounded by a given curve \((\Gamma)\) is computed by evaluating the integral

\[
N_\Gamma = \frac{1}{2i\pi} \int_{\Gamma} \frac{d}{dz} \log |(zI - A)| \, dz.
\]

The evaluation of this integral is also applied in works dealing with non-linear eigenvalue problems [4, 8]. For instance, to compute the number \( N_\Gamma \) of complex eigenvalues which are surrounded by a given curve \((\Gamma)\), the procedure EIGENCNT [3] implies evaluating the characteristic polynomial at many points of \((\Gamma)\) since the selected points must support the quadrature of

\[
N_\Gamma = \frac{1}{2i\pi} \int_{\Gamma} \frac{f'(z)}{f(z)} \, dz. \tag{1}
\]

Let \( z \) and \( z + h \) be two points of \((\Gamma)\). Since

\[
(z + h)I - A = (zI - A) + hI = (zI - A)(I + hR(z)),
\]

where \( R(z) = (zI - A)^{-1} \), it follows that

\[
f(z + h) = f(z) \det(I + hR(z)).
\]

Let \( \Phi_z(h) = \det(I + hR(z)) \), \( g(z) = \frac{f'(z)}{f(z)} \), then

\[
\int_z^{z+h} \frac{f'(z)}{f(z)} \, dz = \log(f(z + h)) - \log(f(z)) = \log \left( \frac{f(z + h)}{f(z)} \right) = \log(\Phi_z(h)) = \log |\Phi_z(h)| + i \arg(\Phi_z(h)).
\]

Given that this is a multivalued function, the goal is therefore to insure that a branch corresponding to a given determination of the complex logarithm can be followed while avoiding any jump to another determination. In [3], a stepsize control is introduced for insuring this property. For large matrices, the number of necessary determinant evaluations may become very high especially when many eigenvalues lie near the boundary \((\Gamma)\).

In this paper, we present some efficient techniques for computing the determinant of large matrices that can be put into block structured form. Such matrices arise in the discretization of partial differential equations especially in combination with domain decomposition and more generally from many sparse matrices after reordering, e.g. with the help of some graph partitioning tools see e.g. [1, 6].
Computing a determinant is efficiently done through an LU-factorization of the matrix with the standard permutation strategies. When the matrix is sparse, the LU-factorization is either with row permutation $PA = LU$ [11, 12] or with additional column permutation $PAQ = LU$ [10], where $P$ and $Q$ are permutation matrices, and where $L$ and $U$ are respectively a lower-triangular matrix with unit main diagonal and an upper-triangular matrix; then $\det(A) = \det(P) \det(U)$ or $\det(A) = \det(P) \det(U) \det(Q)$, where $\det(P) = \pm 1$, $\det(Q) = \pm 1$ and $\det(U)$ is the product of all the diagonal entries of $U$.

2. Avoiding overflows and underflows when computing a determinant

For any non-singular matrix $A \in \mathbb{C}^{n \times n}$, let us consider its LU factorization $PAQ = LU$ where $P$ and $Q$ are permutation matrices of signatures $\sigma_P$ and $\sigma_Q$. Then $\det(A) = \sigma_P \sigma_Q \prod_{i=1}^{n} u_{ii}$ where $u_{ii} \in \mathbb{C}$ are the diagonal entries of $U$. When the matrix $A$ is not correctly scaled, the product $\prod_{i=1}^{n} u_{ii}$ may generate an overflow or underflow. To avoid this, we encode the determinant using the triplet $(\rho, K, n)$ so that

$$\det(A) = \rho K^n$$

where:

$$\rho = \sigma_P \sigma_Q \prod_{i=1}^{n} \frac{|u_{ii}|}{u_{ii}}, \quad (\rho \in \mathbb{C} \text{ with } |\rho| = 1),$$

$$K = \sqrt[n]{\prod_{i=1}^{n} |u_{ii}|} \quad (K > 0).$$

The quantity $K$ is computed through its logarithm:

$$\log(K) = \frac{1}{n} \sum_{i=1}^{n} \log(|u_{ii}|).$$

In this way, the value of the determinant is safely computed even when the matrix is not properly scaled; i.e. diagonal elements of $U$ vary sharply in magnitude. Before raising to power $n$, and to protect from under- or overflow, the positive constant $K$ must be in the interval $[\sqrt[n]{M_{fl}}, \sqrt[n]{M_{fl}}]$ where $M_{fl}$ is the largest representable number in the underlying floating point system; otherwise the value of the determinant is not computed because it will lead to an overflow or underflow.

3. Preliminary result

If $A$ can be put in the form $A = I + UV$, where $U \in \mathbb{C}^{m \times n}, V \in \mathbb{C}^{n \times m}$, then the following proposition shows how to efficiently compute the determinant of $A$.

**Proposition 3.1** If $U \in \mathbb{C}^{m \times n}, V \in \mathbb{C}^{n \times m}$ then, $\det(I_m + UV) = \det(I_n + VU)$.

**Note:** this proposition appears as exercise 6.2.7 in [9].
Proof. Any eigenvalue of $UV$ is either zero or is an eigenvalue of $VU$. Indeed, let $\lambda$ be a nonzero eigenvalue of $UV$ and $w \neq 0$, an associated eigenvector: $(UV)w = \lambda w$. Therefore $(VU)(Vw) = \lambda (Vw)$. Let us prove by contradiction that $Vw \neq 0$: if $Vw = 0$, and since $U(Vw) = \lambda w$, therefore $\lambda w = 0$, and finally, $\lambda = 0$.

Since $Vw \neq 0$, it follows that $(\lambda, Vw)$ is an eigenpair of $VU$; therefore the nonzero eigenvalues of $UV$ are nonzero eigenvalues of $VU$. Since $U$ and $V$ can be interchanged without affecting the result, the two matrices have the same nonzero eigenvalues. An argument of continuity proves that their algebraic multiplicities are the same. For any square matrix $M$, let us denote its spectrum by $\Lambda(M)$. It then follows that, $\Lambda(UV) \cup \{0\} = \Lambda(VU) \cup \{0\}$, which implies $\Lambda(I_m + UV) \cup \{1\} = \Lambda(I_n + VU) \cup \{1\}$ and therefore, $\det(I_m + UV) = \det(I_n + VU)$. \hfill $\diamond$

This proposition is especially useful when, either $m \ll n$, or $n \ll m$, since it may drastically reduce the order of the matrix for which determinant is sought. We make use of this proposition in the following sections.

4. Computing the determinant of a q-block tridiagonal matrix

We now assume that the matrix $A$ is q-block tridiagonal:

$$
\begin{pmatrix}
A_1 & A_{1,2} & 0 & \ldots & 0 \\
A_{2,1} & A_2 & \ddots & \ddots & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots & A_{q-1,q} \\
0 & \ldots & 0 & A_{q,q-1} & A_q
\end{pmatrix},
$$

where for $i = 1, \cdots, q - 1$ the blocks $A_{i+1,i}$ and $A_{i,i+1}$ are corner matrices defined by:

$$
A_{i,i+1} = \begin{pmatrix} 0 & 0 \\ B_i & 0 \end{pmatrix},
$$

$$
A_{i+1,i} = \begin{pmatrix} 0 & C_{i+1} \\ 0 & 0 \end{pmatrix};
$$

with $A_i \in \mathbb{C}^{n_i \times n_i}$, $B_i \in \mathbb{C}^{b_i \times l_i}$, $C_{i+1} \in \mathbb{C}^{c_{i+1} \times r_{i+1}}$. We assume throughout that $c_i \leq n_i$, $b_i \leq n_i$, and $l_i - 1 + r_{i+1} \leq n_i$.

4.1. Sequential method

We begin with a block sequential algorithm with partial pivoting. From the LU factorization $P_1A_1 = L_1U_1$ and by partitioning $\bar{A} = PA$ where $P = \begin{pmatrix} P_1 & 0 \\ 0 & I \end{pmatrix}$ and $A = \begin{pmatrix} A_1 & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, then $\bar{A}$ admits the following LU decomposition:

$$
\bar{A} = \begin{pmatrix} I \\ A_{21}(P_1A_1)^{-1} \end{pmatrix} \begin{pmatrix} P_1A_1 & P_1A_{12} \\ 0 & S_1 \end{pmatrix}
$$
with the Schur complement \( S_1 = A_{22} - A_{21}U_1^{-1}L_1^{-1}P_1A_{12} \). Thus

\[
\det(A) = \det(P_1) \det(A_1) \det(S_1). \tag{6}
\]

Consider the block partition

\[
U_1^{-1} = \begin{bmatrix}
(U_1^{-1})_{11} & (U_1^{-1})_{12} \\
0 & (U_1^{-1})_{22}
\end{bmatrix} \in \mathbb{C}^{n_1 \times n_1},
\]

where \((U_1^{-1})_{22} \in \mathbb{C}^{r_2 \times r_2}\).

Let

\[
X = C_2(U_1^{-1})_{22} \in \mathbb{C}^{c_2 \times r_2}
\]

and

\[
G = L_1^{-1}P_1 \begin{bmatrix}
0 \\
B_1
\end{bmatrix} = \begin{bmatrix}
G_1 \\
G_2
\end{bmatrix}, \tag{7}
\]

with \( G_1 \in \mathbb{C}^{n_1 - r_2 \times l_1} \) and \( G_2 \in \mathbb{C}^{c_2 \times l_1} \); thus \( XG_2 \in \mathbb{C}^{c_2 \times l_1} \).

Then the Schur complement \( S_1 \) can be easily built by

\[
S_1 = A_{22} - \begin{bmatrix}
I & 0 & X \\
0 & L_1^{-1}P_1 & 0
\end{bmatrix} \begin{bmatrix}
0 & B \\
G_1 & G_2
\end{bmatrix} \begin{bmatrix}
I & 0
\end{bmatrix},
\]

Thus, only the leading \( c_2 \times l_1 \) block of \( A_2 \) is affected when \( A_{22} \) is transformed into \( S_1 \).

The computation of \( XG_2 \) requires solving \( l_1 \) triangular systems of order \( n_1 \) for \( G \), \( c_2 \) triangular systems of order \( r_2 \), for \( X \) and a \((c_2 \times r_2) \times (r_2 \times l_1)\) matrix multiplication for \( XG_2 \).

Let \( \tilde{A}_2 \) be the transformed block; this approach is then recursively applied to the trailing matrix

\[
A_{22} = \begin{bmatrix}
\tilde{A}_2 & A_{2,3} & \ldots & 0 \\
A_{3,2} & A_3 & \ldots & \vdots \\
0 & \ddots & \ddots & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
0 & \ldots & 0 & A_{q-1,q} \\
0 & \ldots & 0 & A_{q,q-1} \\
0 & \ldots & 0 & A_q
\end{bmatrix}, \tag{8}
\]

after \( A_{22} \) has been corrected. Repeating this process \((q - 1)\) times we deduce

\[
\det(A) = \prod_{i=1}^{q} \det(P_i) \det(U_i). \tag{9}
\]

**Definition 4.1** The method based on the formula (9) is called \( q \)-block sequential method.
It is worth pointing out that when the LU factorization is carried out with left and right permutation, i.e., when $P_1A_1Q_1 = LU$, the same process applies. Indeed let $F = U_1^{-1}G$, where $G$ is given by equation (7) and let $Y = Q_1F = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$, with $Y_1 \in \mathbb{C}^{n_1-r_2 \times l_1}$ and $Y_2 \in \mathbb{C}^{c_2 \times l_1}$; thus $C_2Y_2 \in \mathbb{C}^{c_2 \times l_1}$ and

$$S_1 = A_{22} = \begin{bmatrix} C_2Y_2 & 0 \\ 0 & 0 \end{bmatrix}.$$ 

Thus as is the previous case, only the leading $c_2 \times l_1$ block of $A_2$ is affected when $A_{22}$ is transformed into $S_1$; but computing $C_2Y_2$ is more expensive than computing $XG_2$ in the previous case as $c_2$ triangular systems of order $r_2$ to be solved are replaced by $l_1$ triangular systems of order $n_1$.

### 4.2. Parallel method

We now describe a parallel method for computing the determinant; the corresponding parallel algorithm is sketched in Algorithm 1. Considering the SPIKE transformation [7], let $D = \text{diag}(A_1, A_2, \cdots, A_q)$ be assumed to be nonsingular, and for $i = 1, \cdots, q$, let $P_iA_iQ_i = L_iU_i$ a sparse LU factorization of the diagonal blocks. Let $P = \text{diag}(P_1, P_2, \cdots, P_q), Q = \text{diag}(Q_1, Q_2, \cdots, Q_q)$ and

$$S \overset{def}{=} D^{-1}A = I_n + T,$$

where

$$I_n = \text{diag}(I_{s_1}, I_{r_2}, I_{t_1}, I_{q_2}, I_{r_3}, I_{q_2}, \cdots, I_{q_q}, I_{q_{q-1}}, I_{q_q}),$$

and where the block vectors are defined by:

$$V_1 \in \mathbb{C}^{n_1 \times l_1} = \begin{bmatrix} V_1^f \\ V_1^b \end{bmatrix} \begin{bmatrix} s_1 \\ r_2 \end{bmatrix},$$

$$= (A_1)^{-1} \begin{bmatrix} 0 \\ B_1 \end{bmatrix} = Q_1U_1^{-1}L_1^{-1}P_1 \begin{bmatrix} 0 \\ B_1 \end{bmatrix},$$
Determinants of large sparse matrices

and for \( i = 2, \cdots, q - 1 \),

\[
V_i \in \mathbb{C}^{n_i \times l_i} \equiv \begin{bmatrix}
V^t_i \\
V^\prime_i \\
V^b_i \\
\end{bmatrix} \equiv \\
\begin{bmatrix}
l_{i-1} \\
s_i \\
r_{i+1} \\
\end{bmatrix},
\]

(12)

\[
W_i \in \mathbb{C}^{n_i \times r_i} \equiv \begin{bmatrix}
W^t_i \\
W^\prime_i \\
W^b_i \\
\end{bmatrix} \equiv \\
\begin{bmatrix}
l_{i-1} \\
s_i \\
r_{i+1} \\
\end{bmatrix},
\]

(14)

and

\[
W_q \in \mathbb{C}^{n_q \times r_q} \equiv \begin{bmatrix}
W^t_q \\
W^\prime_q \\
W^b_q \\
\end{bmatrix} \equiv \\
\begin{bmatrix}
l_{q-1} \\
s_q \\
\end{bmatrix},
\]

(16)

These blocks are called spike vectors in [7]. With this partition, it is clear that:

\[
\det(A) = \left( \prod_{i=1}^{q} \text{sign}(P_i) \text{sign}(Q_i) \det(U_i) \right) \det(S).
\]

Let \( J \in \mathbb{R}^{m \times n} \) be obtained from \( I_n \) after deleting its block rows \( I_{s_1}, I_{s_2}, \cdots, I_{s_q} \), and \( U \in \mathbb{C}^{n \times m} \) from \( T \) after deleting its block columns \( I_{s_1}, I_{s_2}, \cdots, I_{s_q} \), where \( m = \text{l} \) when \( q=2 \) and \( m = n - (\sum_{i=1}^{q} s_i) = \sum_{i=2}^{q} (l_{i-1} + r_i) \) when \( q > 2 \). Then it can be shown that \( T = UJ \), so that

\[
S = I_n + UJ.
\]

(18)

The matrix \( J \) is a restriction/extension matrix; when it post-multiplies a matrix, it extends it and when it pre-multiplies a matrix, it restricts it by selecting some of its block rows. For example if \( V_i \) and \( W_i \) are defined by equations (12) and (14), then \( JU \) selects the block entries \( V^t_i, V^b_i, W^t_i \) and \( W^b_i \) from the matrix \( U \).

Let us denote:

\[
\hat{S} = I_m + JU.
\]

(19)

It follows from Proposition (3.1) and from equations (18) and (19) that the determinant of \( S \) and the inverse of \( S \) can be computed at reduced cost from \( \hat{S} \) as shown in the following proposition:

**Proposition 4.2** If \( S = I + UJ, U \in \mathbb{C}^{n \times m}, J \in \mathbb{C}^{m \times n} \), with \( m \leq n \), then

\[
\det(S) = \det(\hat{S})
\]

and

\[
S^{-1} = I_n - U\hat{S}^{-1}J.
\]
Proof. The second identity is a direct application of the Sherman-Morison-Woodbury formula \([5]\). The cost for computing the determinant of \(\hat{S}\) is therefore reduced since the dimension of \(\hat{S}\) is smaller than that of \(S\).

After having applied the block odd-even permutation \(\Pi\) on both sides of \(\hat{S}\), we get a nice pattern for \(\hat{S}\):

\[
\tilde{S} = \Pi^T \hat{S} \Pi = \begin{pmatrix}
I_{r_3} & V_{b_1}^h & V_{b_2}^h & \cdots & V_{b_{q-1}}^h \\
W_2 & I_{r_2} & V_{b_2}^h & \cdots & V_{b_{q-1}}^h \\
& \ddots & \ddots & \ddots & \ddots \\
& & W_q & I_{r_1} & V_{b_{q-1}}^h \\
& & & \ddots & \ddots \\
& & & & W_1
\end{pmatrix} \in \mathbb{C}^{m \times m}. \tag{20}
\]

Thus

\[
\det(S) = \det(\hat{S}) = \det(\tilde{S}).
\]

5. Special Case with only two blocks

We consider the special case of a two-block tridiagonal matrix because the formulas in that case simplify. Let the matrix \(A \in \mathbb{C}^{n \times n}\) be defined by:

\[
A = \begin{pmatrix}
A_1 & A_{12} \\
A_{21} & A_2
\end{pmatrix}, \tag{21}
\]

where \(A_1 \in \mathbb{C}^{n_1 \times n_1}, A_2 \in \mathbb{C}^{n_2 \times n_2}\) \((n = n_1 + n_2)\), and where \(A_{12}\) and \(A_{21}\) are corner matrices defined by:

\[
A_{12} = \begin{pmatrix}
0 & 0 \\
B & 0
\end{pmatrix}, \text{ and } A_{21} = \begin{pmatrix}
0 & C \\
0 & 0
\end{pmatrix},
\]

with \(B \in \mathbb{C}^{b \times l}, \text{ and } C \in \mathbb{C}^{c \times r}, b \leq n_1, c \leq n_2, l \leq n_1\) and \(r \leq n_2\). Two cases will be considered: the case where the LU factorization is carried out with left and right permutations, and the case where the LU factorization is carried out without right permutation.

5.1. Case of two-block with sparse LU factorization with left and right permutation

For the case of two-block, with the sparse LU factorization using left and right permutation, \(P_1A_1Q_1 = L_1U_1\) and \(P_2A_2Q_2L_2U_2\) we have:

\[
S = \begin{pmatrix}
I_{n_1-r} & V_1^h & 0 \\
0 & I_r & V_1^h \\
0 & W_2 & I_l \\
0 & W_2 & 0
\end{pmatrix}
\]

\[
= I_n + UJ \text{ and } \hat{S} = I_m + JU,
\]

\(A R I M A\)
with
\[ V_1 \in \mathbb{C}^{n_1 \times l} = \begin{bmatrix} V_1^t & V_1^b \end{bmatrix} \begin{bmatrix} m_1 - r \\ r \end{bmatrix} = Q_1 U_1^{-1} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B \end{pmatrix} \]
\[ W_2 \in \mathbb{C}^{n_2 \times r} = \begin{bmatrix} W_2^t & W_2^b \end{bmatrix} \begin{bmatrix} l \\ n_2 - l \end{bmatrix} = Q_2 U_2^{-1} L_2^{-1} P_2 \begin{pmatrix} C \\ 0 \end{pmatrix} \]

where \( m = l \),
\[ U = \begin{pmatrix} 0 & V_1^t \\ 0 & V_1^b \\ W_2^t & 0 \\ W_2^b & 0 \end{pmatrix} \in \mathbb{C}^{n \times m}, J = \begin{pmatrix} 0 & I_r & 0 & 0 \\ 0 & 0 & I_l & 0 \end{pmatrix} \in \mathbb{C}^{m \times n}. \]
Therefore
\[ S = \begin{pmatrix} I_r & V_1^b \\ W_2^t & I_l \end{pmatrix} = \begin{pmatrix} I_r & 0 \\ W_2^t & I_l \end{pmatrix} \begin{pmatrix} I_r & V_1^b \\ 0 & Z \end{pmatrix}, \]
in which \( Z = I - W_2^t V_1^b \in \mathbb{C}^{l \times l} \).

Thus we obtain the following proposition which follows straight from Proposition 4.2 when \( q = 2 \):

**Proposition 5.1** If \( A \in \mathbb{C}^{n \times n} \), can be written in the form
\[ A = \begin{pmatrix} A_1 \\ A_{21} \\ A_{21} \end{pmatrix}, \]
where \( A_1 \in \mathbb{C}^{n_1 \times n_1}, A_2 \in \mathbb{C}^{n_2 \times n_2} \) \( (n = n_1 + n_2) \), and where \( A_{12} \) and \( A_{21} \) are corner matrices defined by:
\[ A_{12} = \begin{pmatrix} 0 & 0 \\ B & 0 \end{pmatrix}, \text{and} A_{21} = \begin{pmatrix} 0 & C \\ 0 & 0 \end{pmatrix}, \]

with \( B \in \mathbb{C}^{b \times l} \), and \( C \in \mathbb{C}^{c \times r} \), \( b \leq n_1, c \leq n_2, l \leq n_1 \text{ and } r \leq n_2. \)

Let \( P_1 A_1 Q_1 = L_1 U_1, P_2 A_2 Q_2 = L_2 U_2 \) be the sparse LU factorization of \( A_1 \) and \( A_2 \). Let: \( P = \text{diag}(P_1, P_2), Q = \text{diag}(Q_1, Q_2), L = \text{diag}(L_1, L_2), U = \text{diag}(U_1, U_2) \), then
\[ \det(A) = \det(P_1) \det(P_2) \det(Q_1) \det(Q_2) \det(U_1) \det(U_2) \det(I_l + W_2^t V_1^b) \]
where
\[ V_1^b = \begin{bmatrix} 0 & I_r \end{bmatrix} Q_1 U_1^{-1} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B \end{pmatrix} \]
and
\[ W_2^t = \begin{bmatrix} I_l & 0 \end{bmatrix} Q_2 U_2^{-1} L_2^{-1} P_2 \begin{pmatrix} C \\ 0 \end{pmatrix}. \]

**Details on the computation of** \( W_2^t V_1^b \):
Since \( V_1^b = \begin{bmatrix} 0 & I_r \end{bmatrix} V_1 \) and \( W_2^t = \begin{bmatrix} I_l & 0 \end{bmatrix} W_2 \), it follows that \( W_2^t V_1^b \) can be written as:
\[ W_2^t V_1^b = \begin{bmatrix} I_l & 0 \end{bmatrix} Q_2 U_2^{-1} L_2^{-1} P_2 \begin{pmatrix} C \\ 0 \end{pmatrix} \begin{bmatrix} 0 & I_r \end{bmatrix} Q_1 U_1^{-1} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B \end{pmatrix}. \]

Thus the computation involves \( 2r \) triangular systems of dimension \( n_2 \) and \( 2l \) triangular systems of dimension \( n_1 \) which can be solved in parallel, and a matrix multiplication of dimension \((l \times r) \times (r \times l)\).
5.2. Case of two-block with sparse LU factorization without right permutation

We now consider the case of a two-block, with the sparse LU factorization without right permutation \( P_1 A_1 = L_1 U_1 \) and \( P_2 A_2 = L_2 U_2 \).

If \( Q_1 = I_{n_1}, Q_2 = I_{n_2} \), then

\[
V^*_1 = \begin{bmatrix} 0 & I_r \end{bmatrix} V_1, \quad (22)
\]

\[
= \begin{bmatrix} 0 & I_r \end{bmatrix} U_1^{-1} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B \end{pmatrix}, \quad (23)
\]

\[
= (U_1^{-1})^{22} \begin{bmatrix} 0 & I_r \end{bmatrix} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B \end{pmatrix}, \quad (24)
\]

\[
W^*_2 = \begin{bmatrix} I_l & 0 \end{bmatrix} W_2, \quad (25)
\]

\[
= \begin{bmatrix} I_l & 0 \end{bmatrix} U_2^{-1} L_2^{-1} P_2 \begin{pmatrix} C \\ 0 \end{pmatrix}, \quad (26)
\]

\[
= \begin{bmatrix} (U_2^{-1})^{11} (U_2^{-1})^{12} \end{bmatrix} L_2^{-1} P_2 \begin{pmatrix} C \\ 0 \end{pmatrix}, \quad (27)
\]

where \((U_2^{-1})^{11} \in \mathbb{C}^{(n_2-r)\times(n_2-r)}, (U_2^{-1})^{12} \in \mathbb{C}^{(n_2-r)\times r}, \) and \((U_2^{-1})^{22} \in \mathbb{C}^{r\times r}.

Details on the computation of \( W^*_2 V^*_1 \):

From equations (24) and (27), it follows that the computation of \( W^*_2 \) involves solving \( 2r \) triangular systems of order \( n_2 \) while the computation of \( V^*_1 \) involves less operations than in the previous case: \( r \) triangular systems of order \( l \) are solved instead of \( l \) triangular systems of order \( n_1 \). These systems can be solved in parallel. The solution of these systems is followed by a \((l \times r) \times (r \times l)\) matrix multiplication. The parallel algorithm corresponding to this case is sketched in Algorithm 2.

6. Parallel algorithms

We now describe the parallel algorithms for the computation of the determinant of a q-block tridiagonal matrix \( A \). The algorithm for the case of q-block with sparse LU factorization using row and column interchanges is sketched in Algorithm 1 and for the case of two-block without column interchange in Algorithm 2. We have assumed that local data needed by each processor have been provided in a preprocessing step. The factorization of \( S \) is carried out by one processor. This of course penalizes the efficiency of the algorithm, as the size of \( S \) increases with the number of blocks, especially when the spikes have large bandwidth.

7. Numerical tests

Numerical tests were conducted on a desktop, equipped with two processors, each with 6 cores Intel(R) Xeon(R); clock: 3.47GHz; RAM: 48GB. The parallel code was
Algorithm 1 PARALDETER1: Parallel algorithm: case of q-block - Program of processor \( P_k \)

Require:
\[ p = \text{number of processors}; \]
\[ q = \text{number of blocks}; \]
\[ \rho = \text{processors}; \]
\[ \text{in local memory} \ A_k, B_k, C_k \text{ if } 2 \leq k < p; \]
\[ \text{A}, \text{B}, \text{C} \text{ if } k=1; \]
\[ \text{A}, \text{B}, \text{C} \text{ if } k=p. \]

Ensure:
\[ \text{number of processor = number of block}; \]
1: Form the sparse LU factorization \( P_k A_k Q_k = L_k U_k \);
2: Compute \((\rho_k, K_k)\) as defined by equations (3) and (4) for the determinant of \( A_k \);
3: \( \text{if } 2 \leq k < p - 1, \text{then} \)
4: \( \text{Solve } P_k L_k U_k Q_k^T V_k = \begin{pmatrix} 0 \\ B_k \end{pmatrix} \) for \( V_k \);
5: \( \text{Solve } P_k L_k U_k Q_k^T W_k = \begin{pmatrix} C_k \\ 0 \end{pmatrix} \) for \( W_k \);
6: \( \text{end if} \)
7: \( \text{if } k = 1, \text{then} \)
8: \( \text{Solve } P_1 L_1 U_1 Q_1^T V_1 = \begin{pmatrix} 0 \\ B_1 \end{pmatrix} \) for \( V_1 \);
9: \( \text{end if} \)
10: \( \text{if } k = p, \text{then} \)
11: \( \text{Solve } P_p L_p U_p Q_p^T W_p = \begin{pmatrix} C_p \\ 0 \end{pmatrix} \) for \( W_p \);
12: \( \text{end if} \)
13: \( \text{if } 2 \leq k < p - 1, \text{then} \)
14: \( \text{send } Z_1^k = \begin{pmatrix} \rho_k \\ K_k \end{pmatrix}, Z_2^k = \begin{pmatrix} V_k^1 \\ V_k^2 \end{pmatrix}, \text{and } Z_3^k = \begin{pmatrix} W_k^1 \\ W_k^2 \end{pmatrix} \) to processor \( P_p \)
15: \( \text{end if} \)
16: \( \text{if } k = 1, \text{then} \)
17: \( \text{send } Z_1^1 = \begin{pmatrix} \rho_1 \\ K_1 \end{pmatrix}, Z_2^1 = \begin{pmatrix} V_1^1 \\ V_1^2 \end{pmatrix} \) to processor \( P_p \)
18: \( \text{end if} \)
19: \( \text{if } k = p, \text{then} \)
20: \( \text{receive } Z_1^1, Z_2^1 \text{ from processor } P_1 \)
21: \( \text{do for } i=2 \cdots p-1, \text{ then} \)
22: \( \text{receive } Z_1^i, Z_2^i \text{ and } Z_3^i \text{ from processor } P_i \)
23: \( \text{end} \)
24: \( \text{Assemble } \tilde{S} \text{ as defined by equation (20)}; \)
25: \( \text{Form the sparse LU factorization } P_\tilde{S} \tilde{Q}_\tilde{S} = L_\tilde{S} U_\tilde{S} \)
26: \( \text{Compute } (\rho_\tilde{S}, K_\tilde{S}) \text{ as defined by equations (3) and (4) for the determinant of } \tilde{S} \)
27: \( \text{Compute } \rho_A = \prod_{r=1}^{p} \rho_r \rho_\tilde{S} \text{ for the determinant of } A \)
28: \( \text{Compute } K_A = \prod_{r=1}^{p} K_r K_\tilde{S} \text{ for the determinant of } A \)
29: \( \text{end if} \)

written in MPI using the C programming language and compiled using the parallel option (mpicce of openMPI version 1.6) of the compiler. The code designed for q-block sequential method described in section 4.1 was written in C programming language and compiled using the sequential option (GCC version 4.6.5) of the compiler. The parallel code was executed using q cores, and only one core was running when the sequential code was used. UNFPAK [12, 15] procedures were used for the sparse LU factorizations. Test matrices are listed in increasing order of size in Table 1. All the matrices are real; the first two belong to the Matrix Market [16] set of tests matrices, the third matrix is obtained as iteration matrix when solving a BDF step in two discretizations of a transport diffusion.
Algorithm 2 PARALDETER2: Parallel algorithm: case of two-block without right permutation - Program of processor $P_q$

Require:
- in local memory $A_1, B$ if $q=1$, $A_2, C$ if $q=2$.
1: Form the sparse LU factorization $P_q A_q = L_q U_q$.
2: Compute $(\rho_q, K_q)$ as defined by equations (3) and (4) for the determinant of $A_q$.
3: if $q=1$, then
   4: Solve $L_1 T_1 = P_1 \begin{pmatrix} 0 & B \end{pmatrix}$ for $T_1$
5: else
6: Solve $L_2 T_2 = P_2 \begin{pmatrix} C & 0 \end{pmatrix}$ for $T_2$
7: end if
8: if $q=1$ then
9: Let $X_1 = \begin{bmatrix} 0 & I_r \end{bmatrix} T_1$
10: Solve $U_1 Y_1 = X_1$
11: else
12: Solve $X_2 U_2 = \begin{bmatrix} I_l & 0 \end{bmatrix}$ for $X_2$
13: Compute $Y_2 = X_2 * T_2$
14: end if
15: if $q=1$, then
16: Send $(\rho_q, K_q), Y_1$ to $P_2$
17: else
18: Receive $Y_1, (\rho_1, K_1)$ from $P_1$
19: Assemble $\tilde{S} = I_l - Y_2 * Y_1$
20: Compute $(\rho_{\tilde{S}}, K_{\tilde{S}})$ as defined by equations (3) and (4) for the determinant of $\tilde{S}$.
21: Compute $\rho_A = \prod_{i=1}^{\tilde{S}_{1}} \rho_{\tilde{S}_i}$ for the determinant of $A$.
22: Compute $K_A = \prod_{i=1}^{\tilde{S}_{1}} K_{\tilde{S}_i}$ for the determinant of $A$.
23: end if

process, the fourth and the fifth matrices were generated randomly using MATLAB function
sprand and the last matrix derives from the discretization on the Poisson Equation 2D.

7.1. Efficiency of the q-block parallel method

In this section we present the results for the q-block parallel method. Test results
show that the efficiency of the parallel algorithm increases until the cost of computing the
determinant of $\tilde{S}$, which in our case is carried by one processor, becomes dominant w.r.t
the cost for handling one block; several factors may contribute to this situation among
which: the size of $\tilde{S}$, its fill-in and its conditioning. Let $\hat{K}$ denote the number of blocks
for which the maximum efficiency is achieved and $\tilde{K}$ the number of blocks for which $m$
the size of $\tilde{S}$ is equal to average block size i.e. $m = u / \hat{K}$. For the matrix of Figure 1,
Table 2, $\hat{K} = 6$ while $\tilde{K} = 5$ and $\hat{K}$ is slightly larger than $\tilde{K}$; the same observation for
the matrix of Figure 2, Table 3 where $\hat{K} = 10$ and $\tilde{K} = 8$. For the matrices of Figure 3, Table
4 and Figure 6, Table 7, $\hat{K} = 7$ and $\tilde{K} = 9$ respectively, but in both cases $\hat{K} > 12$; the
fill-in and conditioning factors may have significantly influenced the cost for handling $\tilde{S}$;
it is worth mentioning that in UMFPACK, the LU factorization algorithm seeks to minimize
fill-in and to guarantee numerical stability. Finally for matrices of Figure 4, Table 5 and
Figure 5, Table 6, both $\hat{K}$ and $\tilde{K}$ are greater than 12.
7.2. Efficiency of the q-block sequential method

Test results show that the q-block sequential method is faster when \( n \) and \( k \) are large. As shown by the speedup curves of Figure 7 and Figure 8, in some cases the sequential q-block method performs even better than the q-block parallel method, especially when the number of blocks is small. For the matrices of Figure 1 and Figure 6, the sequential algorithm performs better when \( k < 4 \) and for the matrix of Figure 1 when \( k \geq 10 \).

7.3. Consequences from the efficiency of the proposed algorithms

From the test results obtained for the set of matrices, the following consequences could be drawn:

– the q-block sequential method allows one to consider on one processor large matrices that cannot be handled in one unit.

– The q-block parallel method is efficient on \( \hat{k} \) processors when \( \hat{k} \) is such that the average block size \( \hat{n} = \frac{n}{\hat{k}} \) is not small when compared to \( m \), the size of \( \hat{R} \).

**Table 1. Characteristics of the test matrices** (Name: Matrix Market name (if from Matrix Market); \( n \): order of the matrix; \( m \): order of \( \hat{S} \) for 12 blocks; NZ/line: number of non zeros elements par line.)

<table>
<thead>
<tr>
<th>Name</th>
<th>( n )</th>
<th>NZ/line</th>
<th>Type</th>
<th>( m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER40R5000</td>
<td>17,281</td>
<td>32.03</td>
<td>Real non-symmetric</td>
<td>13,002</td>
</tr>
<tr>
<td>S3DKQ4M2</td>
<td>90,449</td>
<td>48.95</td>
<td>Real symmetric</td>
<td>16,980</td>
</tr>
<tr>
<td>ITERATION MATRIX</td>
<td>300,000</td>
<td>7.4</td>
<td>Real non-symmetric</td>
<td>12,280</td>
</tr>
<tr>
<td>RANDOM MATRIX</td>
<td>500,000</td>
<td>30.99</td>
<td>Real non-symmetric</td>
<td>330</td>
</tr>
<tr>
<td>RANDOM MATRIX</td>
<td>1,000,000</td>
<td>40.99</td>
<td>Real non-symmetric</td>
<td>440</td>
</tr>
<tr>
<td>POISSON MATRIX</td>
<td>1,000,000</td>
<td>4.99</td>
<td>Real symmetric</td>
<td>15,694</td>
</tr>
</tbody>
</table>

**Table 2. Run time table for Matrix Market matrix e40r5000 where \( n = 17,281 \) NBlock: number of blocks; PTime: parallel time; STime: sequential time**

<table>
<thead>
<tr>
<th>NBlock</th>
<th>STime</th>
<th>PTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11.08</td>
<td>25.03</td>
</tr>
<tr>
<td>3</td>
<td>15.76</td>
<td>19.69</td>
</tr>
<tr>
<td>4</td>
<td>18.14</td>
<td>14.45</td>
</tr>
<tr>
<td>5</td>
<td>19.97</td>
<td>13.54</td>
</tr>
<tr>
<td>6</td>
<td>21.16</td>
<td>12.92</td>
</tr>
<tr>
<td>7</td>
<td>22.56</td>
<td>13.96</td>
</tr>
<tr>
<td>8</td>
<td>23.77</td>
<td>16.40</td>
</tr>
<tr>
<td>9</td>
<td>24.12</td>
<td>19.54</td>
</tr>
<tr>
<td>10</td>
<td>25.58</td>
<td>26.80</td>
</tr>
<tr>
<td>11</td>
<td>24.76</td>
<td>31.67</td>
</tr>
<tr>
<td>12</td>
<td>27.00</td>
<td>38.97</td>
</tr>
</tbody>
</table>
Table 3. Run time table for Matrix Market matrix s3dkq4m2 where n = 90 449. NBlock: number of blocks; PTime: parallel time; STime: sequential time

<table>
<thead>
<tr>
<th>NBlock</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>STime</td>
<td>30.68</td>
<td>38.56</td>
<td>38.50</td>
<td>32.68</td>
<td>34.40</td>
<td>31.51</td>
<td>32.56</td>
<td>33.93</td>
<td>31.45</td>
<td>32.44</td>
<td>33.27</td>
</tr>
<tr>
<td>PTime</td>
<td>49.76</td>
<td>32.34</td>
<td>33.01</td>
<td>30.99</td>
<td>25.60</td>
<td>24.05</td>
<td>22.65</td>
<td>21.02</td>
<td>20.51</td>
<td>21.98</td>
<td>21.54</td>
</tr>
</tbody>
</table>

Table 4. Run time table for the Iteration matrix A300000 where n = 300 000. NBlock: number of blocks; PTime: parallel time; STime: sequential time

<table>
<thead>
<tr>
<th>NBlock</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>STime</td>
<td>116.56</td>
<td>150.43</td>
<td>130.95</td>
<td>126.01</td>
<td>125.64</td>
<td>122.06</td>
<td>117.83</td>
<td>116.12</td>
<td>113.31</td>
<td>109.85</td>
<td>106.35</td>
</tr>
<tr>
<td>PTime</td>
<td>112.62</td>
<td>123.36</td>
<td>91.27</td>
<td>82.92</td>
<td>73.03</td>
<td>70.47</td>
<td>70.85</td>
<td>77.26</td>
<td>76.25</td>
<td>75.69</td>
<td>82.62</td>
</tr>
</tbody>
</table>

Table 5. Run time table for random matrix LB500000 where n = 500 000. NBlock: number of blocks; PTime: parallel time; STime: sequential time

<table>
<thead>
<tr>
<th>NBlock</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>STime</td>
<td>72.79</td>
<td>80.44</td>
<td>71.89</td>
<td>56.60</td>
<td>38.12</td>
<td>12.04</td>
<td>10.69</td>
<td>10.85</td>
<td>10.99</td>
<td>11.03</td>
<td></td>
</tr>
<tr>
<td>PTime</td>
<td>86.70</td>
<td>85.66</td>
<td>51.55</td>
<td>30.21</td>
<td>15.72</td>
<td>3.82</td>
<td>3.06</td>
<td>2.81</td>
<td>2.73</td>
<td>2.72</td>
<td>2.59</td>
</tr>
</tbody>
</table>

Table 6. Run time table the random matrix LBmillion where n = 1 000 000. NBlock: number of blocks; PTime: parallel time; STime: sequential time

<table>
<thead>
<tr>
<th>NBlock</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>STime</td>
<td>240.22</td>
<td>257.82</td>
<td>222.33</td>
<td>165.60</td>
<td>98.19</td>
<td>34.12</td>
<td>34.52</td>
<td>35.04</td>
<td>35.39</td>
<td>35.76</td>
<td>36.07</td>
</tr>
<tr>
<td>PTime</td>
<td>294.35</td>
<td>271.37</td>
<td>147.91</td>
<td>70.38</td>
<td>25.23</td>
<td>10.73</td>
<td>9.73</td>
<td>9.52</td>
<td>8.89</td>
<td>8.85</td>
<td>8.40</td>
</tr>
</tbody>
</table>

Table 7. Run time table for the 3D Poisson Matrix were n = 1 000 000. NBlock: number of blocks; PTime: parallel time; STime: sequential time

<table>
<thead>
<tr>
<th>NBlock</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>STime</td>
<td>99.86</td>
<td>93.36</td>
<td>103.26</td>
<td>98.95</td>
<td>98.93</td>
<td>98.07</td>
<td>97.02</td>
<td>94.35</td>
<td>94.94</td>
<td>92.33</td>
<td>91.92</td>
</tr>
<tr>
<td>PTime</td>
<td>1323.74</td>
<td>270.36</td>
<td>122.30</td>
<td>54.73</td>
<td>54.95</td>
<td>51.81</td>
<td>50.15</td>
<td>46.38</td>
<td>49.42</td>
<td>48.69</td>
<td>53.30</td>
</tr>
</tbody>
</table>
Figure 1. Run time curve for the Matrix Market matrix e40r5000 where n= 17 281. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core.

Figure 2. Run time curve for the Matrix Market matrix s3dkq4m2 where n= 90 449. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core.
Figure 3. Run time curve for the Iteration matrix $A_{300000}$ where $n = 300,000$. Parallel case: Algorithm 1 used. Sequential case: $q$-block sequential method used with one core.

Figure 4. Run time curve for the random matrix $LB_{500000}$ where $n = 500,000$. Parallel case: Algorithm 1 used. Sequential case: $q$-block sequential method used with one core.
Figure 5. Run time curve for the random matrix LBmillion where \( n = 1 \, 000 \, 000 \). Parallel case: Algorithm 1 used. Sequential case: \( q \)-block sequential method used with one core.

Figure 6. Run time curve for the 3D Poisson Matrix where \( n = 1 \, 000 \, 000 \). Parallel case: Algorithm 1 used. Sequential case: \( q \)-block sequential method used with one core.
Figure 7. Speedup curves for the Matrix Market matrix e40r5000 where $n = 17,281$, the Matrix Market matrix s3dkq4m2 where $n = 90,449$ and the Iteration matrix A300000 where $n = 300,000$.

Figure 8. Speedup curves for the random matrix LB500000 where $n = 500,000$, the random matrix LBmillion where $n = 1,000,000$ and the 3D Poisson Matrix where $n = 1,000,000$. 
8. Conclusions

We have described some efficient methods for computing the determinant of large block structured matrices. Test results indicate that the q-block sequential method is efficient when using one processor on large matrices. The q-block parallel method is efficient when the number of processors is such that the average block size $\frac{n}{k}$ is not small compared to the size of $\tilde{R}$. Tests conducted on a set of matrices confirm the efficiency of the proposed algorithms; the efficiency of the parallel algorithm, however is penalized when the spikes have large widths and when the number of blocks increases; this is due to the fact that in our current algorithm, the computation of the determinant of $\tilde{S}$ is carried by one processor; i.e. sequentially and Amdahl’s law applies. In future work, we intend to implement a parallel version of the LU factorization of $\tilde{S}$.

Acknowledgement

The authors would like to thank Bernard Philippe for all his suggestions and assistance. This work was supported in part by the projet LIRIMA through the projet team MOMAPLI, and was carried out during the authors visits within the research team SAGE at the INRIA Rennes-Bretagne Atlantique center.

9. References

[10] SuperLU; X. Sherry, L. J. Demmel, J. Gilbert, L. Grigori, M. Shao, and I. Yamazaki. SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of
linear equations on high performance machines http://crd-legacy.lbl.gov/ xiaoye/SuperLU


[12] UMFPACK; Unsymmetric Multifrontal Sparse LU Factorization Package; http://www.cise.ufl.edu/research/sparse/umfpack/


[16], Matrix Market, Service of the Mathematical and Computational Sciences Division
/ Information Technology Laboratory / National Institute of Standards and Technology,
http://math.nist.gov/MatrixMarket/.