Some efficient methods for computing the determinant of large sparse matrices

Emmanuel Kamgnia, Louis Bernard Nguenang

▶ **To cite this version:**

Emmanuel Kamgnia, Louis Bernard Nguenang. Some efficient methods for computing the determinant of large sparse matrices. Revue Africaine de Recherche en Informatique et Mathématiques Appliquées, 2014, Volume 17 - 2014 - Special issue CARI'12, pp.73-92. 10.46298/arima.1968 . hal-01300060

HAL Id: hal-01300060
https://inria.hal.science/hal-01300060

Submitted on 8 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Special issue CARI'12

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 $g(z) = \det((z+h)I - A)/\det(zI - A)$, 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 $g(z) = \det((z+h)I - A)/\det(zI - A)$, 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 plaine. When one proceeds as in [2, 3] by applying the residue theorem, the number N_Γ of complex eigenvalues which are surrounded by a given curve (Γ) is computed by evaluating the integral

$$N_\Gamma = \frac{1}{2i\pi}\int_\Gamma \frac{d}{dz}\log\det(zI-A)dz.$$

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_Γ of complex eigenvalues which are surrounded by a given curve (Γ), the procedure EIGENCNT [3] implies evaluating the characteristic polynomial at many points of (Γ) 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 (Γ). Since

$$\begin{aligned}(z+h)I-A &= (zI-A)+hI\\ &= (zI-A)(I+hR(z)),\end{aligned}$$

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

$$\begin{aligned}\int_z^{z+h}\frac{f^{'}(z)}{f(z)}dz &= \log(f(z+h))-\log(f(z))\\ &= \log\left(\frac{f(z+h)}{f(z)}\right)\\ &= \log(\Phi_z(h))\\ &= \log|\Phi_z(h)|+i\ \arg(\Phi_z(h)).\end{aligned}$$

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 (Γ).

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 and 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 σ_P and σ_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 (ρ, K, n) so that

$$\det(A) = \rho K^n \tag{2}$$

where:

$$\rho = \sigma_P\sigma_Q \prod_{i=1}^{n} \frac{u_{ii}}{|u_{ii}|}, \quad (\rho \in \mathbb{C} \text{ with } |\rho| = 1), \tag{3}$$

$$K = \sqrt[n]{\prod_{i=1}^{n} |u_{ii}|} \quad (K > 0). \tag{4}$$

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 $[\frac{1}{\sqrt[n]{M_{fl}}}, \sqrt[n]{M_{fl}}]$ 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 exercice 6.2.7 in [9].

Proof. Any eigenvalue of UV is either zero or is an eigenvalue of VU. Indeed, let λ 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 (λ, 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, $\det(I_m + UV) = \det(I_n + VU)$. ◇

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:

$$\begin{pmatrix} A_1 & A_{1,2} & 0 & \dots & 0 \\ A_{2,1} & A_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & A_{q-1,q} \\ 0 & \dots & 0 & A_{q,q-1} & A_q \end{pmatrix}, \tag{5}$$

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 $P_1 A_1 = L_1 U_1$ and by partitioning $\tilde{A} = PA$ 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 $\tilde{A}$ admits the following LU decomposition:

$$\tilde{A} = \begin{pmatrix} I & 0 \\ A_{21}(P_1 A_1)^{-1} & I \end{pmatrix} \begin{pmatrix} P_1 A_1 & P_1 A_{12} \\ 0 & S_1 \end{pmatrix}$$

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 $G_2 \in \mathbb{C}^{r_2 \times l_1}$; thus $XG_2 \in \mathbb{C}^{c_2 \times l_1}$.
Then the Schur complement S_1 can be easily built by

$$\begin{aligned} S_1 &= A_{22} - \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{bmatrix} 0 & X \end{bmatrix} L_1^{-1}P_1 \begin{bmatrix} 0 \\ B \end{bmatrix} \begin{bmatrix} I & 0 \end{bmatrix}, \\ &= A_{22} - \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{bmatrix} 0 & X \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} \begin{bmatrix} I & 0 \end{bmatrix}, \\ &= A_{22} - \begin{bmatrix} XG_2 & 0 \\ 0 & 0 \end{bmatrix}. \end{aligned}$$

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

$$A_{22} = \begin{bmatrix} \tilde{A_2} & A_{2,3} & 0 & \dots & 0 \\ A_{3,2} & A_3 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & A_{q-1,q} \\ 0 & \dots & 0 & A_{q,q-1} & A_q \end{bmatrix}, \tag{8}$$

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 $P_1 A_1 Q_1 = LU$, the same process applies. Indeed let $F = U_1^{-1} G$, where G is given by equation (7) and let $Y = Q_1 F = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$, 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 $C_2 Y_2 \in \mathbb{C}^{c_2 \times l_1}$ and

$$S_1 = A_{22} - \begin{bmatrix} C_2 Y_2 & 0 \\ 0 & 0 \end{bmatrix}.$$

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 $C_2 Y_2$ 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 $D = \mathrm{diag}(A_1, A_2, \cdots, A_q)$ be assumed to be nonsingular, and for $i = 1, \cdots, q$, let $P_i A_i Q_i = L_i U_i$ a sparse LU factorization of the diagonal blocks. Let $P = \mathrm{diag}(P_1, P_2, \cdots, P_q)$, $Q = \mathrm{diag}(Q_1, Q_2, \cdots, Q_q$ and

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

where

$$I_n = \mathrm{diag}(I_{s_1}, I_{r_2}, I_{l_1}, I_{s_2}, I_{r_3}, I_{l_2} \cdots, I_{r_q}, I_{l_{q-1}}, I_{s_q}),$$

$$T = \begin{pmatrix}
0_{s_1} & 0 & V_1' & 0 & 0 & & & & & & & \\
0 & 0_{r_2} & V_1^b & 0 & 0 & & & & & & & \\
0 & W_2^t & 0_{l_1} & 0 & 0 & V_2^t & & & & & & \\
0 & W_2' & 0 & 0_{s_2} & 0 & V_2' & & & & & & \\
0 & W_2^b & 0 & 0 & 0_{r_3} & V_2^b & & & & & & \\
 & & & & W_3^t & 0_{l_2} & 0 & 0 & V_3^t & & & \\
 & & & & W_3' & 0 & 0_{s_3} & 0 & V_3' & & & \\
 & & & & W_3^b & 0 & 0 & 0_{r_4} & V_3^b & & & \\
 & & & & & & & & \ddots & & & \\
 & & & & & & & & & 0 & V_{q-1}^t \\
 & & & & & & & & & 0 & V_{q-1}' \\
 & & & & & & & & & 0_{r_q} & 0 & V_{q-1}^b \\
 & & & & & & & & 0 & W_q^t & 0_{l_{q-1}} & 0 \\
 & & & & & & & & 0 & W_q' & 0 & 0_{s_q}
\end{pmatrix}$$

and where the block vectors are defined by:

$$V_1 \in \mathbb{C}^{n_1 \times l_1} \equiv \left| \begin{matrix} V_1' \\ V_1^b \end{matrix} \right| \begin{matrix} s_1 \\ r_2, \end{matrix} \tag{10}$$

$$= (A_1)^{-1} \begin{pmatrix} 0 \\ B_1 \end{pmatrix} = Q_1 U_1^{-1} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B_1 \end{pmatrix}, \tag{11}$$

and for $i = 2, \cdots, q-1$,

$$V_i \in \mathbb{C}^{n_i \times l_i} \equiv \left| \begin{array}{c} V_i^t \\ V_i' \\ V_i^b \end{array} \right| \begin{array}{c} l_{i-1} \\ s_i \\ r_{i+1}, \end{array} \tag{12}$$

$$= (A_i)^{-1} \begin{pmatrix} 0 \\ B_i \end{pmatrix} = Q_i U_i^{-1} L_i^{-1} P_i \begin{pmatrix} 0 \\ B_i \end{pmatrix}, \tag{13}$$

$$W_i \in \mathbb{C}^{n_i \times r_i} \equiv \left| \begin{array}{c} W_i^t \\ W_i' \\ W_i^b \end{array} \right| \begin{array}{c} l_{i-1} \\ s_i \\ r_{i+1} \end{array} \tag{14}$$

$$= (A_i)^{-1} \begin{pmatrix} C_i \\ 0 \end{pmatrix} = Q_i U_i^{-1} L_i^{-1} P_i \begin{pmatrix} C_i \\ 0 \end{pmatrix}, \tag{15}$$

and

$$W_q \in \mathbb{C}^{n_q \times r_q} \equiv \left| \begin{array}{c} W_q^t \\ W_q' \end{array} \right| \begin{array}{c} l_{q-1} \\ s_q \end{array} \tag{16}$$

$$= (A_q)^{-1} \begin{pmatrix} C_q \\ 0 \end{pmatrix} = Q_q U_q^{-1} L_q^{-1} P_q \begin{pmatrix} C_q \\ 0 \end{pmatrix}. \tag{17}$$

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

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

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 $m = l$ 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

$$S = I_n + UJ. \tag{18}$$

The matrix J is a rectriction/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_i^t, V_i^b, W_i^t and W_i^b from the matrix U.

Let us denote:

$$\hat{S} = I_m + JU. \tag{19}$$

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 S is therefore reduced since the dimension of $\hat{S}$ is smaller than that of S ◇

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

$$\tilde{S} = \Pi^T \hat{S} \Pi = \left(\begin{array}{cccc|cccc} I_{r_2} & & & & V_1^b & & & \\ W_2^b & I_{r_3} & & & & V_2^b & & \\ & \ddots & \ddots & & & & \ddots & \\ & & W_{q-1}^b & I_{r_q} & & & & V_{q-1}^b \\ \hline W_2^t & & & & I_{l_1} & V_2^t & & \\ & \ddots & & & & \ddots & \ddots & \\ & & \ddots & & & & \ddots & V_{q-1}^t \\ & & & W_q^t & & & & I_{l_{q-1}} \end{array}\right) \in \mathbb{C}^{m \times m}. \quad (20)$$

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:

$$A = \left(\begin{array}{cc} A_1 & A_{12} \\ A_{21} & A_2 \end{array}\right), \quad (21)$$

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:

$$A_{12} = \left(\begin{array}{cc} 0 & 0 \\ B & 0 \end{array}\right), \text{ and } A_{21} = \left(\begin{array}{cc} 0 & C \\ 0 & 0 \end{array}\right),$$

with $B \in \mathbb{C}^{b \times l}$, and $C \in \mathbb{C}^{c \times r}$, $b \leq n_1$, $c \leq n_2$, $l \leq n_1$ 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, $P_1 A_1 Q_1 = L_1 U_1$ and $P_2 A_2 Q_2 = L_2 U_2$ we have:

$$\begin{aligned} S &= \left(\begin{array}{cccc} I_{n_1 - r} & 0 & V_1' & 0 \\ 0 & I_r & V_1^b & 0 \\ 0 & W_2^t & I_l & 0 \\ 0 & W_2' & 0 & I_{n_2 - l} \end{array}\right) \\ &= I_n + UJ \text{ and } \hat{S} = I_m + JU, \end{aligned}$$

with

$$V_1 \in \mathbb{C}^{n_1 \times l} = \left| \begin{array}{c|c} V_1' & n_1 - r \\ V_1^b & r \end{array} \right| = Q_1 U_1^{-1} L_1^{-1} P_1 \begin{pmatrix} 0 \\ B \end{pmatrix}$$

$$W_2 \in \mathbb{C}^{n_2 \times r} = \left| \begin{array}{c|c} W_2^t & l \\ W_2' & n_2 - l \end{array} \right| = Q_2 U_2^{-1} L_2^{-1} P_2 \begin{pmatrix} C \\ 0 \end{pmatrix}$$

where $m = l$,

$$U = \begin{pmatrix} 0 & V_1' \\ 0 & V_1^b \\ W_2^t & 0 \\ W_2' & 0 \end{pmatrix} \in \mathbb{C}^{n \times m}, J = \begin{pmatrix} 0 & I_r & 0 & 0 \\ 0 & 0 & I_l & 0 \end{pmatrix} \in \mathbb{C}^{m \times n}.$$

Therefore

$$\hat{S} = \begin{pmatrix} I_r & V_1^b \\ W_2^t & I_l \end{pmatrix} = \begin{pmatrix} I_r & 0 \\ W_2^t & I_l \end{pmatrix} \begin{pmatrix} I_r & V_1^b \\ 0 & Z \end{pmatrix},$$

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*

$$A \;=\; \begin{pmatrix} A_1 & A_{12} \\ A_{21} & A_2 \end{pmatrix},$$

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 $C \in \mathbb{C}^{c \times r}$, $b \leq n_1$, $c \leq n_2$, $l \leq n_1$ and $r \leq n_2$.

Let $P_1 A_1 Q_1 = L_1 U_1$, $P_2 A_2 Q_2 = L_1 U_2$ be the sparse LU factorization of A_1 and A_2. Let: $P = \mathrm{diag}(P_1, P_2)$, $Q = \mathrm{diag}(Q_1, Q_2)$, $L = \mathrm{diag}(L_1, L_2)$, $U = \mathrm{diag}(U_1, L_2)$, then

$$\det(A) = \det(P_1)\det(P_2)\det(Q_1)\det(Q_2)\det(U_1)\det(U_2)\det\left(I_l + W_2^t V_1^b\right)$$

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 2 r triangular systems of dimension n_2 and 2 l 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

$$V_1^b = \left[\begin{array}{cc} 0 & I_r \end{array}\right] V_1, \tag{22}$$

$$= \left[\begin{array}{cc} 0 & I_r \end{array}\right] U_1^{-1} L_1^{-1} P_1 \left(\begin{array}{c} 0 \\ B \end{array}\right), \tag{23}$$

$$= (U_1^{-1})^{22} \left[\begin{array}{cc} 0 & I_r \end{array}\right] L_1^{-1} P_1 \left(\begin{array}{c} 0 \\ B \end{array}\right), \tag{24}$$

$$W_2^t = \left[\begin{array}{cc} I_l & 0 \end{array}\right] W_2, \tag{25}$$

$$= \left[\begin{array}{cc} I_l & 0 \end{array}\right] U_2^{-1} L_2^{-1} P_2 \left(\begin{array}{c} C \\ 0 \end{array}\right), \tag{26}$$

$$= \left[\begin{array}{cc} (U_2^{-1})^{11} & (U_2^{-1})^{12} \end{array}\right] L_2^{-1} P_2 \left(\begin{array}{c} C \\ 0 \end{array}\right), \tag{27}$$

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^t V_1^b:

From equations (24) and (27), it follows that the computation of W_2^t involves solving $2\ r$ triangular systems of order n_2 while the computation of V_1^b 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 system 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 $\tilde{S}$ is carried out by one processor. This of course penalizes the efficiency of the algorithm, as the size of $\tilde{S}$ 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:
$p =$ number of processors;
$q =$ number of blocks;
in local memory A_k, B_k, C_k if $2 \leq k < p$, A_1, B_1 if k=1, A_k, C_k if k=p.
Ensure: number of processor = number of block ;
Form the sparse LU factorization $P_k A_k Q_k = L_k U_k$;
Compute (ρ_k, K_k) as defined by equations (3) and (4)for the determinant of A_k;
if $2 \leq k \leq p-1$, **then**
  Solve $P_k L_k U_k Q_k^T V_k = \begin{pmatrix} 0 \\ B_k \end{pmatrix}$ for V_k;
  Solve $P_k L_k U_k Q_k^T W_k = \begin{pmatrix} C_k \\ 0 \end{pmatrix}$ for W_k;
end if
if $k = 1$, **then**
  Solve $P_1 L_1 U_1 Q_1^T V_1 = \begin{pmatrix} 0 \\ B_1 \end{pmatrix}$ for V_1;
end if
if $k = p$, **then**
  Solve $P_p L_b U_p Q_p^T W_p = \begin{pmatrix} C_p \\ 0 \end{pmatrix}$ for W_p;
end if
if $2 \leq k < p-1$, **then**
  send $Z_k^1 = \begin{pmatrix} \rho_k \\ K_k \end{pmatrix}$, $Z_k^2 = \begin{pmatrix} V_k^t \\ V_k^b \end{pmatrix}$, and $Z_k^3 = \begin{pmatrix} W_k^t \\ W_k^b \end{pmatrix}$ to processor P_p
end if
if $k = 1$, **then**
  send $Z_1^1 = \begin{pmatrix} \rho_1 \\ K_1 \end{pmatrix}$, $Z_1^2 = \begin{pmatrix} V_1^t \\ V_1^b \end{pmatrix}$ to processor P_p
end if
if $k = p$, **then**
  receive Z_1^1, Z_1^2 from processor P_1
  do for i=2$\cdots$ p-1,
    receive Z_i^1, Z_i^2 and Z_i^3 from processor P_i
  end
  Assemble $\tilde{S}$ as defined by equation (20);
  Form the sparse LU factorization $P_{\tilde{S}} \tilde{S} Q_{\tilde{S}} = L_{\tilde{S}} U_{\tilde{S}}$
  Compute $(\rho_{\tilde{S}}, K_{\tilde{S}})$ as defined by equations (3) and (4) for the determinant of $\tilde{S}$
  Compute $\rho_A = \prod_{i=1}^{p} \rho_i \rho_{\tilde{S}}$ for the determinant of A
  Compute $K_A = \prod_{i=1}^{p} K_i K_{\tilde{S}}$ for the determinant of A
end if

written in `MPI` using the `C` programming language and compiled using the parallel option (mpice 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. `UMFPACK` [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.
Form the sparse LU factorization $P_q A_q = L_q U_q$
Compute (ρ_q, K_q) as defined by equations (3) and (4)for the determinant of A_q;
if q=1, **then**
  Solve $L_1 T_1 = P_1 \begin{pmatrix} 0 \\ B \end{pmatrix}$ for T_1
else
  Solve $L_2 T_2 = P_2 \begin{pmatrix} C \\ 0 \end{pmatrix}$ for T_2
end if
if q=1 **then**
  Let $X_1 = \begin{bmatrix} 0 & I_r \end{bmatrix} T_1$
  Solve $U_1 Y_1 = X_1$
else
  Solve $X_2 U_2 = \begin{bmatrix} I_l & 0 \end{bmatrix}$ for X_2
  Compute $Y_2 = X_2 * T_2$
end if
if q=1, **then**
  Send (ρ_q, K_q), Y_1 to P_2
else
  Receive Y_1, (ρ_1, K_1) from P_1
  Assemble $\tilde{S} = I_l - Y_2 * Y_1$
  Compute $(\rho_{\tilde{S}}, K_{\tilde{S}})$ as defined by equations (3) and (4) for the determinant of $\tilde{S}$;
  Compute $\rho_A = \prod_{i=1}^{2} \rho_i \rho_{\tilde{S}}$ for the determinant of A
  Compute $K_A = \prod_{i=1}^{2} K_i K_{\tilde{S}}$ for the determinant of A
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. $m = n/\tilde{K}$. 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 $\hat{K} = 9$ respectively, but in both cases $\tilde{K} > 12$; 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 $\tilde{R}$.

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

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

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

NBblock	2	3	4	5	6	7	8	9	10	11	12
STime	11.08	15.76	18.14	19.97	21.16	22.56	23.77	24.12	25.58	24.76	27.00
PTime	25.03	19.69	14.45	13.54	12.92	13.96	16.40	19.54	26.80	31.67	38.97

Table 3. *Run time table for Matrix Market matrix s3dkq4m2 where n= 90 449 NBlock: number of blocks; PTime: parallel time; STime: sequential time*

NBblock	2	3	4	5	6	7	8	9	10	11	12
STime	30.68	38.56	38.50	32.68	34.40	31.51	32.56	33.93	31.45	32.44	33.27
PTime	49.76	32.34	33.01	30.99	25.60	24.05	22.65	21.02	20.51	21.98	21.54

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

NBblock	2	3	4	5	6	7	8	9	10	11	12
STime	116.56	150.43	130.95	126.01	125.64	122.06	117.83	116.12	113.31	109.85	106.35
PTime	112.62	123.36	91.27	82.92	75.03	70.47	70.85	77.26	76.25	75.69	82.62

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

NBblock	2	3	4	5	6	7	8	9	10	11	12
STime	72.79	80.44	71.89	56.60	38.12	12.04	10.69	10.85	10.85	10.99	11.03
PTime	86.70	85.66	51.55	30.21	15.72	3.82	3.06	2.81	2.73	2.72	2.59

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

NBblock	2	3	4	5	6	7	8	9	10	11	12
STime	240.22	257.82	222.33	165.60	98.19	34.12	34.52	35.04	35.39	35.76	36.07
PTime	294.35	271.37	147.91	70.38	25.23	10.73	9.73	9.52	8.89	8.85	8.40

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*

NBblock	2	3	4	5	6	7	8	9	10	11	12
STime	99.86	93.36	103.26	98.95	98.93	98.07	97.02	94.35	94.94	92.33	91.92
PTime	1323.74	270.36	122.30	54.73	54.95	51.81	50.15	46.38	49.42	48.69	53.30

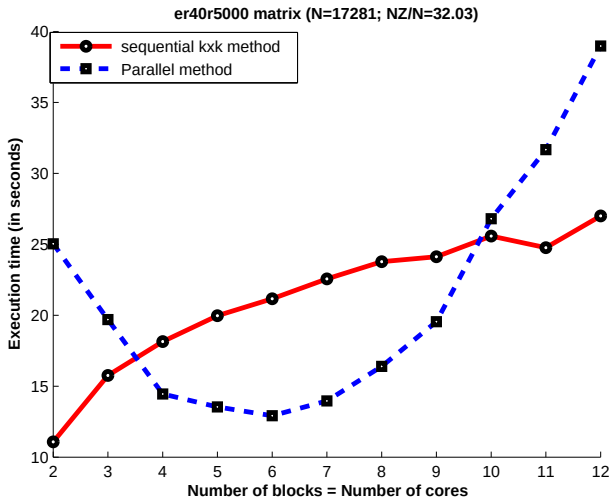

Figure 1. *Run time curve for the Matrix Market matrix e40r5000 where n= 17 281. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core*

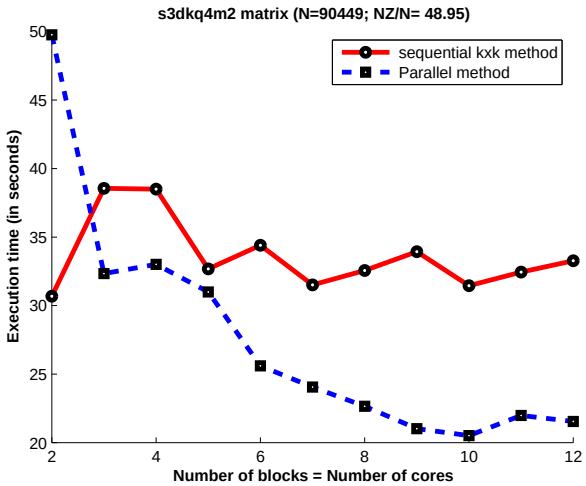

Figure 2. *Run time curve for the Matrix Market matrix s3dkq4m2 where n= 90 449. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core*

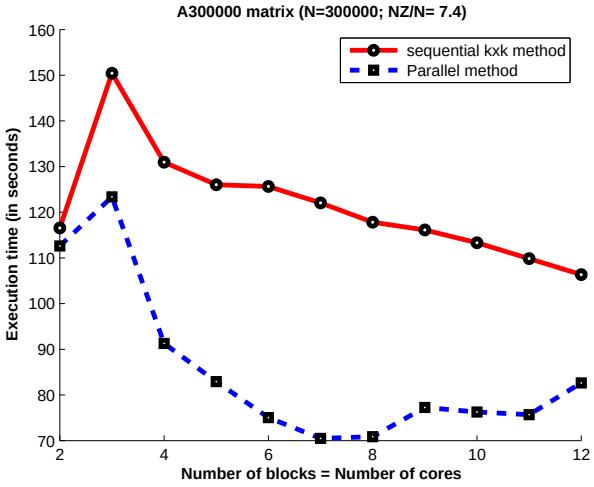

Figure 3. *Run time curve for the Iteration matrix A300000 where n= 300 000. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core*

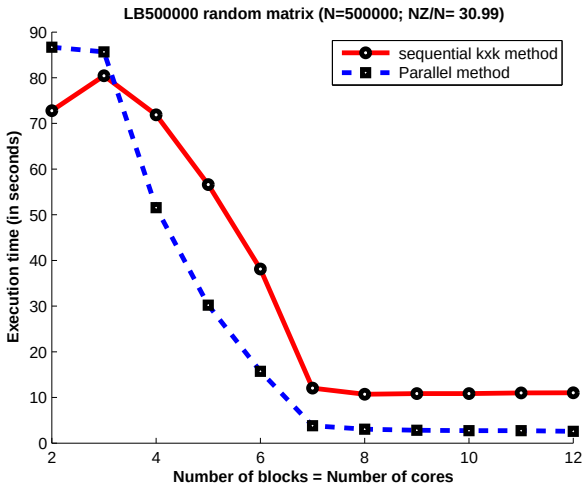

Figure 4. *Run time curve for the random matrix LB500000 where n = 500 000. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core*

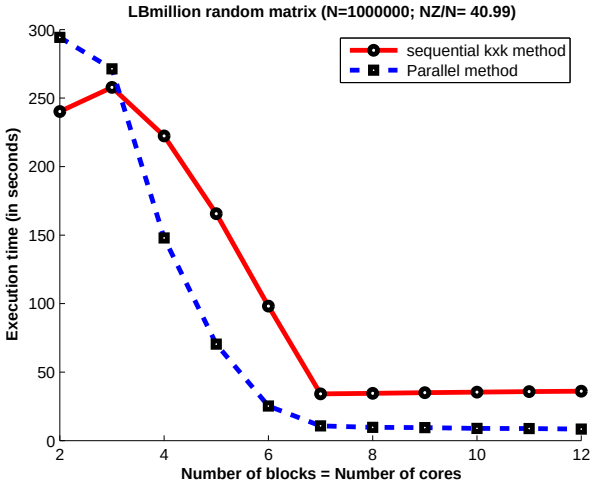

Figure 5. *Run time curve for the random matrix LBmillion where n = 1 000 000. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core.*

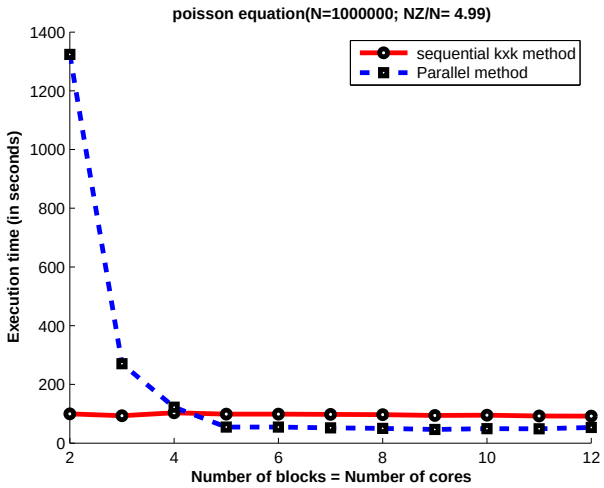

Figure 6. *Run time curve for the 3D Poisson Matrix where n = 1 000 000. Parallel case: Algorithm 1 used. Sequential case: q-block sequential method used with one core.*

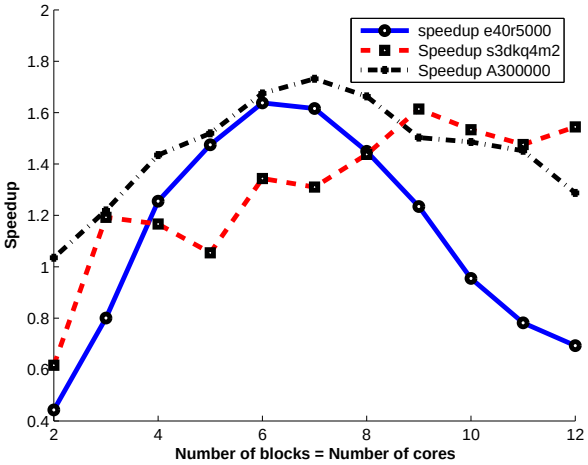

Figure 7. *Speedup curves for the Matrix Market matrix e40r5000 where n= 17 281, the Matrix Market matrix s3dkq4m2 where n= 90 449 and the Iteration matrix A300000 where n= 300 000.*

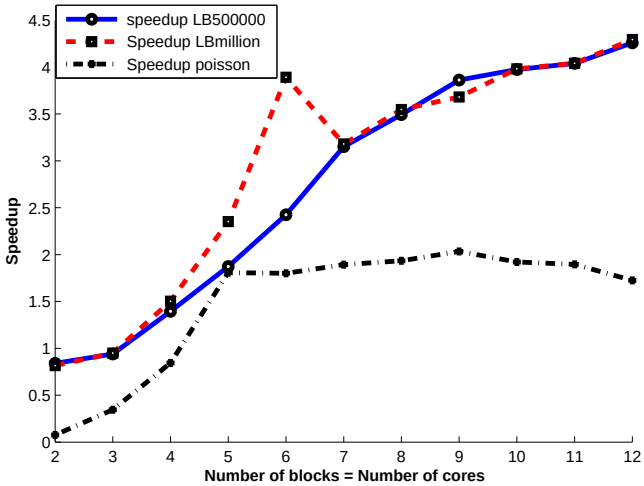

Figure 8. *Speedup curves for the random matrix LB500000 where n = 500 000, the random matrix LBmillion where n = 1 000 000 and the 3D Poisson Matrix where n = 1 000 000.*

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 k 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 be 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

[1] G. A. Atenekeng-Kahou, L. Grigori, and M. Sosonkina. A partitioning algorithm for block-diagonal matrices with overlap. *Parallel Computing*, 34:332 – 344, 2008.

[2], O. Bertrand and B. Philippe. Counting the eigenvalues surrounded by a closed curve, *Sib. Zh. Ind. Mat.*,4:73 – 94, 2001.

[3] E. Kamgnia and B. Philippe. Counting eigenvalues in domains of the complex field. *Electronic Transactions on Numerical Analysis.*,40:1 – 16, 2013.

[4] D. Bindel Bounds and error estimates for nonlinear eigenvalue problems, *Berkeley Applied Mathematical Seminar*, October 2008

[5] G. Golub and C. V. Loan *Matrix Computation*, p51; The Johns Hopkins University Press, Second Edition, 1989.

[6] G. Karypis and V. Kumar. METIS - unstructured graph partitioning and sparse matrix ordering system, version 2.0. University of Minnesota, CS Dept, Technical report, 1995.

[7] E. Polizzi and A. Sameh A parallel hybrid banded systems solver: the SPIKE algorithm *Parallel Computing*, 32:177–194, 2006.

[8] Maeda, Yasuyuki, Futamura, Yasunori, Sakurai and Tetsuya Stochastic estimation method of eigenvalue density for nonlinear eigenvalue problem on the complex plane *JSIAM Letters* , 3:61 – 64, 2011

[9] C. Meyer Matrix Analysis and Applied Linear Algebra *SIAM*, pp 483, 2000

[10] SuperLU; X. Sherry, L. J. Demmel, J. Gilbert, L. Grigori, M. Shao, and I. Yamazaki. SuperLU is a general purpose bibrary for the direct solution of large, sparse, nonsymmetric systems of

linear equations on high performance machines *http://crd-legacy.lbl.gov/ xiaoye/SuperLU*

[11] MUMPS; Multifrontal Massively Parallel sparse direct Solver *http://graal.ens-lyon.fr/MUMPS/*

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

[13] T. A. Davis Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method *ACM Transactions on Mathematical Software*, 30:2, 196 – 199, 2004.

[14] T. A. Davis and I. S. Duff A combined unifrontal/multifrontal method for unsymmetric sparse matrices, *ACM Transactions on Mathematical Software*, 25.1, 1 – 19, 1999.

[15] T. A. Davis and I. S. Duff, An unsymmetric-pattern multifrontal method for sparse LU factorization *SIAM Journal on Matrix Analysis and Applications*, 18.1, 140 – 158, 1997.

[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/.