ELSEVIER

Parallel Computing 28 (2002) 199–221

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Parallel computation of pseudospectra of large sparse matrices

Dany Mezher [a], Bernard Philippe [b,*]

[a] *ESIB-USJ, Campus des Sciences et Technologies, Beirut, Lebanon*
[b] *IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France*

Received 1 November 2000; received in revised form 1 May 2001

Abstract

The parallel computation of the pseudospectrum is presented. The Parallel Path following Algorithm using Triangles (PPAT) is based on the Path following Algorithm using Triangles (PAT). This algorithm offers total reliability and can handle singular points along the level curve without difficulty. Furthermore, PPAT offers a guarantee of termination even in the presence of round-off errors and makes use of the large granularity for parallelism in PAT. This results in large speedups and high efficiency. The PPAT is able to trace multiple level curves simultaneously and takes into account the symmetry of the pseudospectrum in the case of real matrices to reduce the total computational cost.

Keywords: Pseudospectrum; Lanczos algorithm; Multifrontal QR decomposition; Parallelism

1. Introduction

Many scientific applications require the localization of the eigenvalues of matrices in some predefined region of the complex plane. Eigenvalues and spectra may reveal information about the behavior of linear and non-linear systems, stability, resonance and accessibility to matrix iterations and preconditioners.

Following an early idea from Landau, Godunov [10] and Trefethen [23] independently defined the notion of pseudospectrum. For each $\epsilon > 0$, the pseudospectrum of a matrix is the set in the complex plane defined by

* Corresponding author.
E-mail addresses: dany.mezher@fi.usj.edu.lb (D. Mezher), bernard.philippe@irisa.fr (B. Philippe).

0167-8191/02/$ - see front matter
PII: S0167-8191(01)00136-3

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : z \text{ is an eigenvalue of } A + E \text{ where } ||E||_2 \leqslant \epsilon\}$$
$$= \{z \in \mathbb{C} : ||(A - zI)^{-1}||_2 \geqslant \epsilon^{-1}\}.$$

When A is normal, the set $\Lambda_\epsilon(A)$ is the union of the discs of radius ϵ centered at every eigenvalue; when A is non-normal, the pseudospectrum may be much bigger. The determination of the pseudospectrum of a matrix A involves the evaluation of the function

$$s : z \to \sigma_{\min}(A - zI),$$

where $\sigma_{\min}$ denotes the smallest singular value. A common way to compute the pseudospectrum of a matrix A in a given region is to compute the function $s(z)$ on a discretization of the domain and feed the result to a contour plot routine. This is a highly expensive approach since it requires the computation of $s(z)$ for a large number of points. In [15], Heuveline et al. develop a parallel code to compute the pseudospectrum of matrices on a discretized domain. The main idea is to use different processors to compute $s(z)$ for different grid nodes and to parallelize the computation of $s(z)$.

It is now commonly accepted that path following algorithms which compute the level curve corresponding to a given value of ϵ are of much smaller complexity than methods based on a grid discretization [24]. The first attempt in this direction was done by Brühl [6]. Based on continuation with a predictor–corrector scheme, the process may fail in the case of angular discontinuities along the level curve [4,12–14,22]. In [26], Wright and Trefethen use the upper Hessenberg matrix constructed after successive iterations of the implicitly restarted Arnoldi algorithm to cheaply compute an approximation of the pseudospectrum in a region near the interesting eigenvalues. They show that for highly non-normal matrices the computed approximation does not agree with the pseudospectrum. In the sequel, we develop a method which computes the pseudospectrum at a higher cost than their method but it computes the exact pseudospectrum and not an approximation of it.

The Parallel Path following Algorithm using Triangles (PPAT) presented in this paper can handle singular points along the level without difficulty. Based on the Path following Algorithm using Triangles (PAT) it is guaranteed to terminate even in the presence of round-off errors [19] and provides large granularity for parallelism.

Section 2 introduces the PAT. In Section 3, the PPAT is derived. The numerical algorithms used to compute $\sigma_{\min}$ are presented in Section 4 and the corresponding parallel implementations are described in Section 5. Finally, Section 6 reports some numerical results showing the reliability and efficiency of the proposed algorithm.

2. PAT – a brief introduction

The PAT presented in [19] is a totally reliable path following algorithm. The PAT is proven to be numerically stable and offers a guarantee of termination even in the presence of round-off errors. Furthermore, the PAT is able to handle singular points along the level curve of interest without difficulty. The main idea of the PAT is to line

up a set of equilateral triangles along the level curve as presented in Fig. 1 and use a bisection algorithm to compute a numerical approximation of the pseudospectrum. A technique to compute multiple connected components of the same level curve having two sets of points $\{z_i : s(z_i) \leqslant \epsilon\}$ and $\{z_e : s(z_e) > \epsilon\}$ is presented in [19]. This is usually the case for small values of ϵ.

The choice of the bisection algorithm is mainly due to its reliability and numerical stability [25]. In terms of *worst case* analysis, it always yields a zero at the required precision. The bisection method only requires the continuity of the function $s(z)$ (but not its differentiability). As a trade-off, it converges only *linearly* with the convergence factor $1/2$. In our case, it is guaranteed to converge since the function values at the two starting points satisfy

$$s(z_0) \leqslant \epsilon < s(z_1).$$

If we were to consider another root finder, we would like to maintain the reliability of the bisection. Therefore, the only possible way would be to consider a bisection process combined with a fast root-finder. The procedure `ZEROIN` which is available on `Netlib` combines efficiently the bisection, the secant method and even a quadratic interpolation [5]. However, the tests performed for an acceptable precision equal to $10^{-2}\tau$ using `ZEROIN` and the bisection process proved that both algorithms are equivalent (requiring seven evaluations of $s(z)$ per segment).

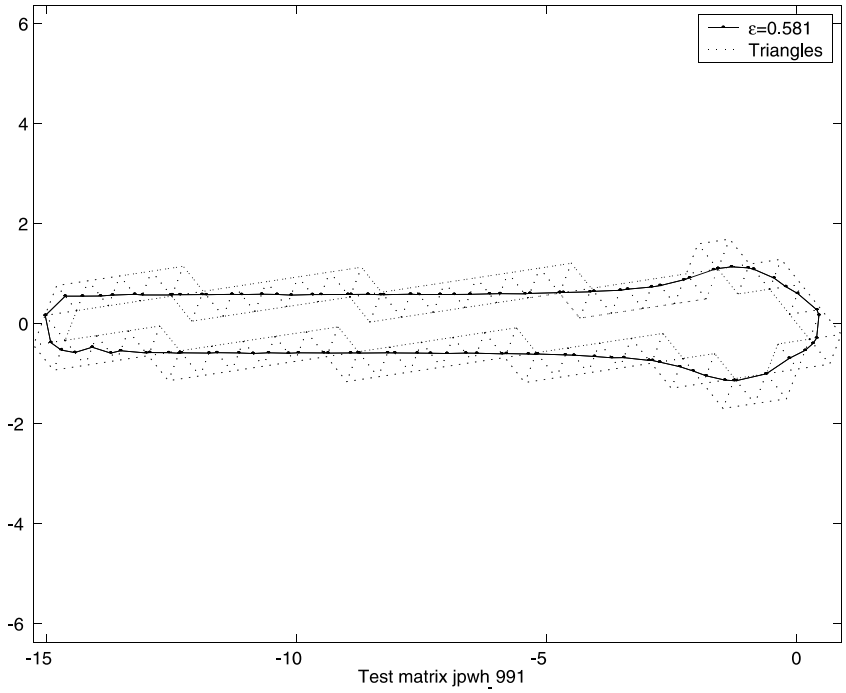

Fig. 1. A level curve and the corresponding triangles for `jpwh_991`.

Table 1
Values of $p(T)$ and $\theta(T)$

$s(x)$	$s(y)$	$s(z)$	$p(T_i)$	$\theta(T_i)$
$\leqslant \epsilon$	$> \epsilon$	$> \epsilon$	x	$\frac{\pi}{3}$
$\leqslant \epsilon$	$> \epsilon$	$\leqslant \epsilon$	y	$-\frac{\pi}{3}$

Given an equilateral triangle $T_i = \{x, y, z\}$ in the complex plane $\mathbb{C}$ such that

$$s(x) \leqslant \epsilon \quad \text{and} \quad s(y) > \epsilon \tag{1}$$

we compute the triangle T_{i+1}

$$T_{i+1} = F(T_i) = R(p(T_i), \theta(T_i))(T_i),$$

where $R(x, \alpha)$ is the rotation centered at x with angle α, $p(T)$ and $\theta(T)$ are chosen as shown in Table 1.

Fig. 2 displays the four different possibilities for the transformation F where

$$\Gamma_\epsilon(s) = \{z \in \mathbb{C} : s(z) = \epsilon\},$$

$$\Delta_\epsilon(s) = \{z \in \mathbb{C} : s(z) < \epsilon\}$$

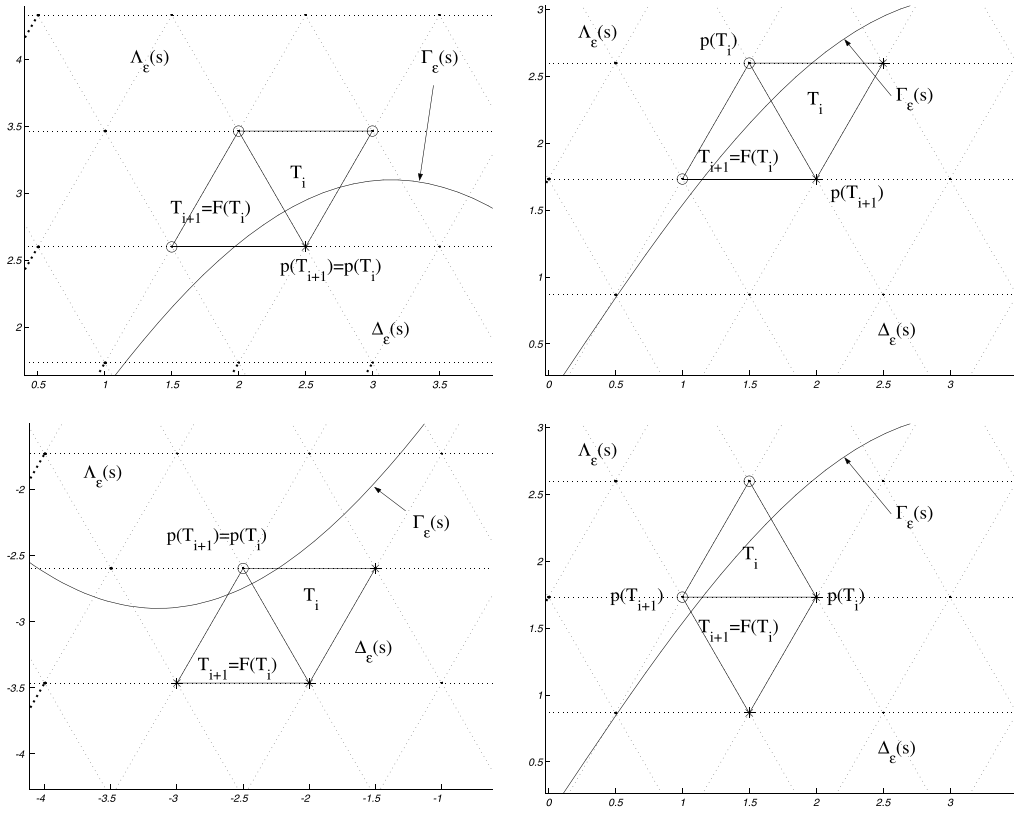

Fig. 2. The transformation F.

and

$$\Lambda_\epsilon(s) = \{z \in \mathbb{C} : s(z) > \epsilon\}.$$

In [19] is proven that for any given equilateral triangle $T_0 = \{x, y, z\}$ satisfying condition (1):

- F is a bijection, and the orbit $O(T_0) = \{T_0, T_1, \ldots, T_i, \ldots\}$, where $T_i = F^i(T_0)$, is a finite set having $F(T_{n-1}) = T_0$ for some n (we assume that n is the smallest integer which satisfies the condition).
- $\forall T_i = \{x_i, y_i, z_i\} \in O(T_0)$, such that $s(x_i) \leqslant s(y_i) \leqslant s(z_i)$, one of the two relations holds

$$s(x_i) \leqslant s(y_i) \leqslant \epsilon < s(z_i)$$

or

$$s(x_i) \leqslant \epsilon < s(y_i) \leqslant s(z_i).$$

- To compute a level curve of length l, we need n equilateral triangles of size τ, where n satisfies

$$\frac{l}{\tau} \leqslant n \leqslant \frac{10}{\sqrt{3}} \frac{l}{\tau}.$$

- The computed orbit is the exact orbit of a function $\tilde{s}$ close to s.
- The use of integer coordinates to identify the triangle vertices guarantees the termination of the process even in the presence of round-off errors.

Algorithm 1 displays the PAT and also presents the technique used to compute the starting triangle T_0. Given a complex value $\tilde{z}_0$ such that $s(\tilde{z}_0) \leqslant \epsilon$ ($\tilde{z}_0$ can be a numerical approximation of an eigenvalue), we build the complex sequence

$$\tilde{z}_k = \tilde{z}_0 + 2^{k-1}\tau e^{i\theta},$$

where θ is a random angle. We can easily prove that $\lim_{k\to\infty} s(\tilde{z}_k) = +\infty$. Therefore, there exists a $k_0 \in \mathbb{N}$ satisfying $s(\tilde{z}_{k_0}) > \epsilon$. The bisection algorithm run on the interval $[\tilde{z}_0, \tilde{z}_{k_0}]$ is used to compute two points z_i and z_e where

$$s(z_i) \leqslant \epsilon, s(z_e) > \epsilon \quad \text{and} \quad |z_i - z_e| = \tau.$$

The triangle $T_0 = \{z_i, z_e, z_i + (z_e - z_i)e^{(i\pi)/3}\}$ satisfies condition (1) and is used to generate the set $O(T_0)$.

Algorithm 1. *The Path following Algorithm using Triangles* (*PAT*).

a. *Startup*

Given ϵ, τ and $\tilde{z}_0$ such that $s(\tilde{z}_0) \leqslant \epsilon$
FOR $k = 1, 2, \ldots$ DO
$\tilde{z}_k = \tilde{z}_0 + 2^{k-1}\tau e^{i\theta}$
IF $s(\tilde{z}_k) > \epsilon$ **break**
END
Use a bisection algorithm over $\tilde{z}_0$ and $\tilde{z}_k$ to compute z_i and z_e where

$s(z_i) \leqslant \epsilon < s(z_e)$ and $|z_i - z_e| = \tau$
Let $T_0 = \{z_i, z_e, z_i + (z_e - z_i)e^{(i\pi)/3}\}$ be an initial triangle
b. *Compute the F-orbit*
FOR $i = 0, 1, 2, \ldots$ **DO**
$T_{i+1} = F(T_i)$
IF $T_{i+1} = T_0$ **break**
use a bisection algorithm to approximate the pseudospectrum
END

2.1. Pseudospectrum of real matrices

In the particular case of real matrices, the pseudospectrum is symmetric with respect to the real axis. Therefore, we compute only half of the pseudospectrum, with either positive or negative imaginary part. Given a starting z_0, such that $s(z_0) \leqslant \epsilon$, and a triangle edge size τ, then $T_0 = \{\alpha, \beta, \gamma\}$ denotes the starting equilateral triangle, where Table 2 can be used to determine α, β and γ, where $\Re(z)$ and $\Im(z)$ are the real part and imaginary part of z. If T_0 has two vertices satisfying condition (1), it is used to generate the set $O(T_0)$. In this case, we can prove that for any triangle $T \in O(T_0)$, there exists a triangle $\tilde{T} \in O(T_0)$ such that the vertices of $\tilde{T}$ are the complex conjugate of the vertices of T. Using this technique we can limit the computation to the positive or negative imaginary part of the complex plane. The rest of the curve is obtained by a symmetry with respect to the real axis. The process is terminated when the triangles T_{i+1} (or T_{i-1}) and T_i are symmetric with respect to the real axis. If T_0 does not satisfy condition (1), we define a smaller value of τ or look for another starting point z_0.

3. The parallel PAT (PPAT)

A clear difficulty in Algorithm 1 is that it requires for every triangle of $O(T_0)$:

- one evaluation of $\sigma_{\min}$ to choose the center and the angle of the rotation;
- q evaluations of $\sigma_{\min}$ for the bisection process. We can easily show that q is given by

$$q = \left\lceil \frac{\ln(\tau) - \ln(\eta)}{\ln(2)} \right\rceil, \tag{2}$$

Table 2
Starting triangle for real matrices

Vertex	Real part	Imaginary part
α	$\Re(z_0) - \frac{\tau}{2}$	$\left\lfloor \frac{\Im(z_0)}{\tau(\sqrt{3}/2)} \right\rfloor \times \tau \frac{\sqrt{3}}{2}$
β	$\Re(z_0) + \frac{\tau}{2}$	$\Im(\alpha)$
γ	$\Re(z_0)$	$\Im(\alpha) + \tau \frac{\sqrt{3}}{2}$

where τ is the triangle edge size and η is the convergence threshold of the bisection process.

Therefore, to compute a level curve of length l we need to evaluate $s(z)$ at $O(q\frac{l}{\tau})$ complex points. If the matrix is a dense matrix of order N, then the computation of $s(z)$ costs $O(N^3)$ floating point operations. If the matrix is sparse the cost will be lower and depending on the number of non-zero entries in the triangular factors of the matrix. Because of the high amount of computation, we consider parallel evaluations of $\sigma_{\min}$ and even, for large cases, parallelism within the computation of the singular value. The latter will only be considered in a special version of the code which will be described in Section 5.

The PPAT shown in Algorithms 2 and 3, exploits the fact that we can perform the following tasks in parallel:

1. compute multiple level curves (either different connected components for the same value of ϵ or different level curves for different values of ϵ),
2. compute multiple slices of the same connected component,
3. use F and F^{-1} to proceed in both directions on a single level curve slice,
4. run different bisections.

Algorithm 2. *The Master algorithm.*

Broadcast matrix A
Given z_0, τ and ϵ, find z_i and z_e such that $s(z_i) \leqslant \epsilon$, $s(z_e) > \epsilon$ and $|z_i - z_e| = \tau$
set $z_{-i} = z_i$ and $z_{-e} = z_e$
set $z_1^* = z_i + (z_e - z_i)e^{i(\pi/3)}$
set $z_{-1}^* = z_i + (z_e - z_i)e^{i(-\pi/3)}$
Add `Compute`$(s(z_1^*))$ and `Compute`$(s(z_{-1}^*))$ to **Task List**
WHILE not(Empty(**Task List**)) or a worker is busy **DO**
  Look for an available worker
  WHILE Found **DO**
    IF Task List contains a task **THEN**
     Send first task in **Task List** to worker
     Remove first task from **Task List**
     Look for an available worker
    ELSE
     break
    END
  END
  Wait for incoming messages
  IF received $s = s(z_1^*)$ **THEN**
    IF $s > \epsilon$ **THEN** $z_e = z_1^*$ **ELSE** $z_i = z_1^*$ **END**
    Add `Find`(z_i, z_e, ϵ) to **Task List**
    set $z_1^* = z_i + (z_e - z_i)e^{i(\pi/3)}$
    Add `Compute`$(s(z_1^*))$ to **Task List**
  ELSE
    IF received $s = s(z_{-1}^*)$ **THEN**

IF $s > \epsilon$ **THEN** $z_{-e} = z^*_{-1}$ **ELSE** $z_{-i} = z^*_{-1}$ **END**
Add `Find`$(z_{-i}, z_{-e}, \epsilon)$ to **Task List**
set $z^*_{-1} = z_{-i} + (z_{-e} - z_{-i})e^{i(-\pi/3)}$
Add `Compute`$(s(z^*_1))$ to **Task List**
END
END
END
Clean Up and exit

Algorithm 3. *The Worker algorithm.*

Receive Matrix A
FOR $k = 1, 2, 3 \ldots$ **DO**
Receive a message from Master
IF worker received z **THEN**
Compute $s = \sigma_{\min}(A - zI)$
Send s to Master
ELSE
WHILE$|z_i - z_e| > \eta$ **DO**
$z = 0.5 \times (z_i + z_e)$
$s = \sigma_{\min}(A - zI)$
IF $s \leqslant \epsilon$ **THEN**
$z_i = z$
ELSE
$z_e = z$
END
END
Send z to Master
END
END
Clean Up and exit

Therefore, we use a set of working nodes capable of computing $s(z)$ for a given complex value z (referred to as `Compute`$(s(z))$ task) and finding $z \in [z_l, z_g]$, where $\sigma_{\min}(A - zI) \simeq \epsilon$ given (z_l, z_g, ϵ), such that, $s(z_l) \leqslant \epsilon$ and $s(z_g) > \epsilon$ (referred to as `Find`(z_l, z_g, ϵ) task).

This can only be done if all workers have access to the matrix A. Therefore, A is broadcast to all workers.

The workers are controlled by a master process as presented in Fig. 3. The master generates computational tasks to be executed by idle workers. Additional tasks are queued in a task list managed by the master. Given a triangle $T_i = \{x, y, z\} \in O(T_0)$ such that $s(x) \leqslant \epsilon$ and $s(y) > \epsilon$, the master spawns a `Compute`$(s(z))$ task to determine $F(T_i)$, or $F^{-1}(T_i)$ if we are proceeding in the reverse direction, and a `Find`(x, y, ϵ) task to locate a new point on the level curve.

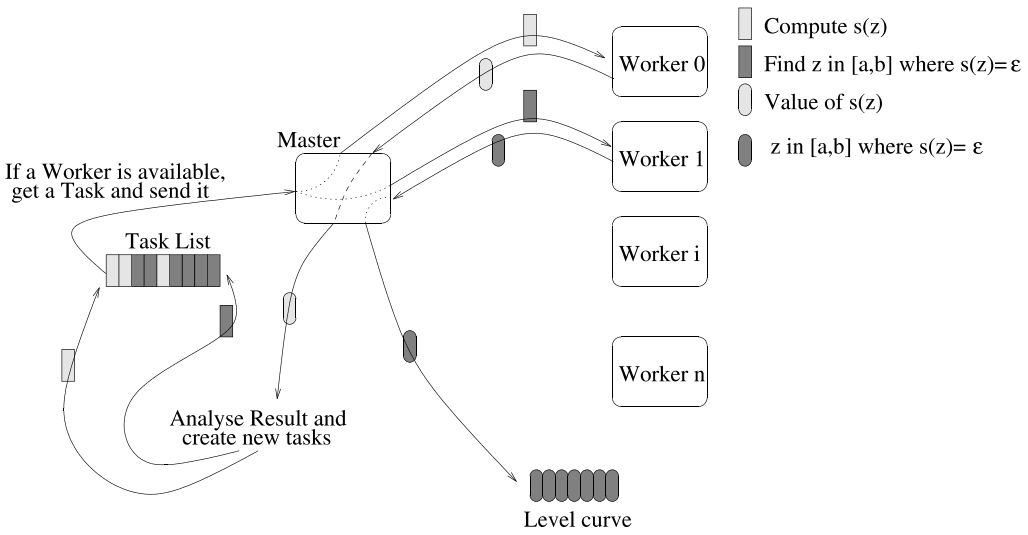

Fig. 3. Parallel architecture.

Upon reception of a message, the master analyses the result. If the message returns a complex z, where $s(z) \simeq \epsilon$, then z is stored in the level curve memory space, otherwise, the master determines $T_{i+1} = F(T_i)$ (or $T_{i-1} = F^{-1}(T_i)$) and generates the corresponding `Compute`$(s(\cdot))$ and `Find`$(\cdot,\cdot,\epsilon)$ tasks. Finally, the master retrieves a task from the task list and sends it to the idle worker. This dynamic task scheduling allows better load balancing among the different processors of a heterogeneous network of workstations.

`Compute`$(s(\cdot))$ tasks are considered as urgent tasks and are inserted at the top of the task-list while `Find`$(\cdot,\cdot,\cdot)$ tasks are appended to the end of the task list. This technique assigns the highest priority to the `Compute`$(s(\cdot))$ tasks. In case of a small number of workers, `Find`$(\cdot,\cdot,\cdot)$ tasks will be stored in the task list while the `Compute`$(s(\cdot))$ tasks are being served by the master leading to a too large task list. To limit the size of the task list, the high priority tasks are treated as low priority ones whenever the task list size exceeds a fixed limit. In this case, the `Compute`$(s(\cdot))$ tasks are added at the end of the task list reducing the task creation process.

The startup of the PPAT is identical to the startup of the PAT. The main difference is the use of `Compute`$(s(z))$ tasks to compute $s(z)$.

3.1. Speedup and efficiency

Let n be the number of equilateral triangles built to compute an approximation of a given level curve, p the number of processors, t_σ the time needed to compute $\sigma_{\min}$ and t_{com} the time to exchange a message. We can limit the number of generated tasks to n `Compute`$(s(\cdot))$ tasks to build the orbit and n `Find`$(\cdot,\cdot,\epsilon)$ tasks to compute a numerical approximation of the level curve.

Since t_σ is so large that any effect of t_{com} is negligible, therefore the analysis is done without concern for the communication costs. In the following, we consider t_σ to be the time unit; therefore, each $\texttt{Find}(\cdot,\cdot,\epsilon)$ task, requiring q evaluations of $\sigma_{\min}$, accounts for q time units where q is given by Eq. (2).

The time needed to compute a level curve approximation using a single processor is given by

$$t_1 = n + nq,$$

whereas, for large values of p ($p \geqslant q+1$), the time to compute the level curve approximation would be

$$t_\infty = n + q.$$

Therefore, the upper bound of the speedup, achieved whenever $p \geqslant q+1$, is given by

$$S_\infty = \frac{t_1}{t_\infty} = \frac{n+nq}{n+q} < \min(1+q, n).$$

On the other hand, when $p < q+1$, the computation of the level curve approximation is performed in three phases:

1. Reduced speedups are observed in the first $p-1$ time units where a new task is spawned for each time unit,
2. peak performance is observed whenever the number of generated tasks exceeds $p-1$,
3. the speedup drops in the final $p-1$ time units where no pending tasks are available to feed to idle workers.

In the startup and cleanup phases (steps 1 and 3), $p(p-1)$ tasks are computed; therefore, peak performance is observed for $t_{\text{peak}} = (n(q+1) - p(p-1))/p$ time units. The total time needed to compute the level curve using p processors is given by

$$t_p = 2(p-1) + t_{\text{peak}} = p - 1 + \frac{n(q+1)}{p}.$$

In general, t_p can be expressed as

$$t_p = \begin{cases} n+q & \text{if } p \geqslant q+1, \\ p-1+\frac{n(q+1)}{p} & \text{if } p < q+1. \end{cases} \tag{3}$$

Since we proceed in both directions, we can assume that $p/2$ processors are used to compute $n/2$ elements. Therefore, expression (3) yields

$$t_p = \begin{cases} \frac{n+2q}{2} & \text{if } p \geqslant 2q+2, \\ \frac{p-2}{2}+\frac{n(q+1)}{p} & \text{if } p < 2q+2. \end{cases} \tag{4}$$

In this case, the speedup is given by

$$S_p = \begin{cases} \frac{2n(q+1)}{n+2q} & \text{if } p \geqslant 2q+2, \\ \frac{2np(q+1)}{p^2-2p+2n(q+1)} & \text{if } p < 2q+2. \end{cases} \tag{5}$$

For large values of n, the speedup and efficiency can be expressed as

$$S_p = \min(p, 2q+2) + \mathrm{O}(1/n) \tag{6}$$

and

$$E_p = \min\left(1, \frac{2q+2}{p}\right) + \mathrm{O}(1/n). \tag{7}$$

The upper bound of the speedup is given by $S_{\max} = 2q+2$. This upper limit is achieved whenever $p \geqslant 2q+2$.

Higher speedups can be obtained when multiple slices of the same level curve are computed simultaneously where each slice requires $2q+2$ processors. Given a complex value $\tilde{z}_0$ such that $s(\tilde{z}_0) \leqslant \epsilon$ and a triangle edge size τ, we use the startup procedure shown in Algorithm 1 with different angles θ_j to compute multiple level curve points. Each point can be used to start a level curve slice. To get through successfully, the triangles of the different computed slices should align perfectly (see Fig. 4). Therefore, a prefixed lattice is used for all level curve slices. If p is the number of available processors, then $l = \lceil p/(2q+2) \rceil$ slices of the same level curve can be computed, where $l-1$ slices use $2q+2$ processors and the last slice uses the remaining processors.

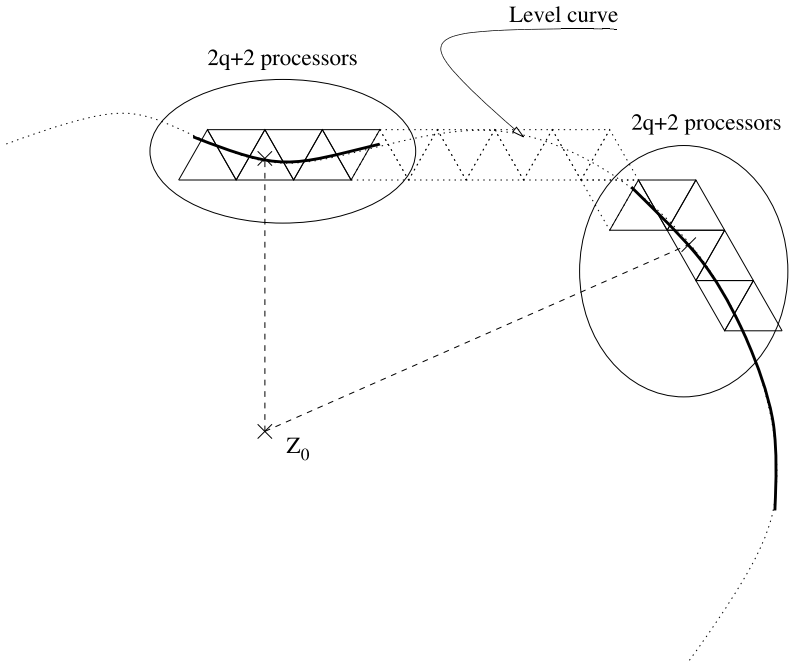

Fig. 4. Computing multiple slices of the same level curve.

4. Computing the smallest singular value

Given a large sparse matrix $A \in \mathbb{C}^{n\times n}$, we consider the matrix $B \in \mathbb{C}^{2n\times 2n}$

$$B = \begin{pmatrix} 0 & A^{-1} \\ A^{-H} & 0 \end{pmatrix}.$$

The eigenvalues of B are the singular values of A^{-1} and their negatives. Therefore the largest eigenvalue of B is equal to $1/\sigma_{\min}(A)$. The corresponding eigenvector is $x = \left(x_1^t\, x_2^t\right)^t$, where $x_1 \in \mathbb{C}^n$ and $x_2 \in \mathbb{C}^n$ are the right and left singular vectors of A, respectively. Therefore the evaluation of $\sigma_{\min}(A - zI)$ can be done through an evaluation of $1/\lambda_{\max}(B)$ with

$$B = \begin{pmatrix} 0 & (A - zI)^{-1} \\ (A - zI)^{-H} & 0 \end{pmatrix} = \begin{pmatrix} P & 0 \\ 0 & Q \end{pmatrix} \begin{pmatrix} 0 & R^{-1} \\ R^{-H} & 0 \end{pmatrix} \begin{pmatrix} P^t & 0 \\ 0 & Q^H \end{pmatrix}, \tag{8}$$

where $(A - zI)P = QR$ is a QR decomposition with column reordering of $A - zI$ and P is a permutation matrix to reduce the fill-in during the QR decomposition of the matrix $A - zI$. The matrix P is built using a minimum degree algorithm [1,9].

We can restrict ourselves to the computation of the largest eigenvalue of the matrix

$$\tilde{B} = \begin{pmatrix} 0 & R^{-1} \\ R^{-H} & 0 \end{pmatrix},$$

where a multifrontal QR decomposition is used to compute R [2,20]. The Lanczos algorithm is used to compute the largest eigenvalue [7,16].

Algorithm 4. *Multifrontal QR decomposition.*

Build the elimination tree to determine the dependencies between the columns of the matrix
FOR $k = 1, \ldots, n$ **DO**
  IF k is a leaf **THEN**
    F_k is the set of all the rows of the original matrix having their leading element in column k
  ELSE
    F_k is composed by appending the contribution blocks coming from the sons to the rows of the original matrix having their leading element in column k
  END
  Perform a Householder reflection on F_k
  The first row of F_k is the kth row of R
  The remaining sub-matrix is the contribution
  block to be sent to the father
END

We start by building a dependency graph, or *elimination tree* (see Fig. 6), that will express the connections between successive steps involved in the Householder reflec-

tions. The tree nodes in the elimination tree correspond to matrix columns where column a is a parent of column b if the application of the Householder reflection corresponding to column b alters column a. The multifrontal QR decomposition is shown in Algorithm 4. Notice that the computations of the Householder reflections are totally independent for separated leaves. Therefore, the reduction of independent subtrees can be done in parallel.

The Lanczos algorithm is a method dedicated to the particular case of Hermitian matrices [15,21]. Algorithm 5 shows an adapted version of the Lanczos algorithm to compute the largest eigenvalue of the matrix $\tilde{B}$.

Algorithm 5. *Lanczos algorithm.*

Choose $v_1 \in \mathbb{R}^{2n}$
$v_1 = v_1/||v_1||$
$\beta_1 = 0$
$v_0 = [0]$
FOR $j = 1, \ldots, m$ **DO**

$$\begin{bmatrix} v_{j,1} \\ v_{j,2} \end{bmatrix} = v_j \text{ where } v_{j*} \in \mathbb{R}^n$$

$w_{j,1} = R^{-1}v_{j,2} - \beta_j v_{j-1,1}$
$w_{j,2} = R^{-H}v_{j,1} - \beta_j v_{j-1,2}$

$$w_j = \begin{bmatrix} w_{j,1} \\ w_{j,2} \end{bmatrix}$$

$H_{j,j} = \alpha_j = w_j^t v_j$
$w_j = w_j - \alpha_j v_j$
$H_{j,j+1} = \beta_{j+1} = ||w_j||_2$
$v_{j+1} = w_j/\beta_{j+1}$

END
$(\lambda_{\max}, y) = eig(H_m)$ where $H_m y = \lambda_{\max} y$
$x = Vy$ with $V = [v_1, \ldots, v_m]$

5. Parallel computation of $\sigma_{\min}$

For larger matrix dimensions, parallel implementations of the multifrontal QR decomposition and the Lanczos algorithm are needed [2,8,11,17,20]. Furthermore, the matrix R can be too large to fit in the memory of a single machine. Therefore, we use a computational model based on a hierarchy of clusters where each worker node is replaced by a set of machines offering the same functionalities as shown in Fig. 5. The group master controls the set of slaves and responds to requests of the main master. The slave nodes execute the operations ordered by the group master.

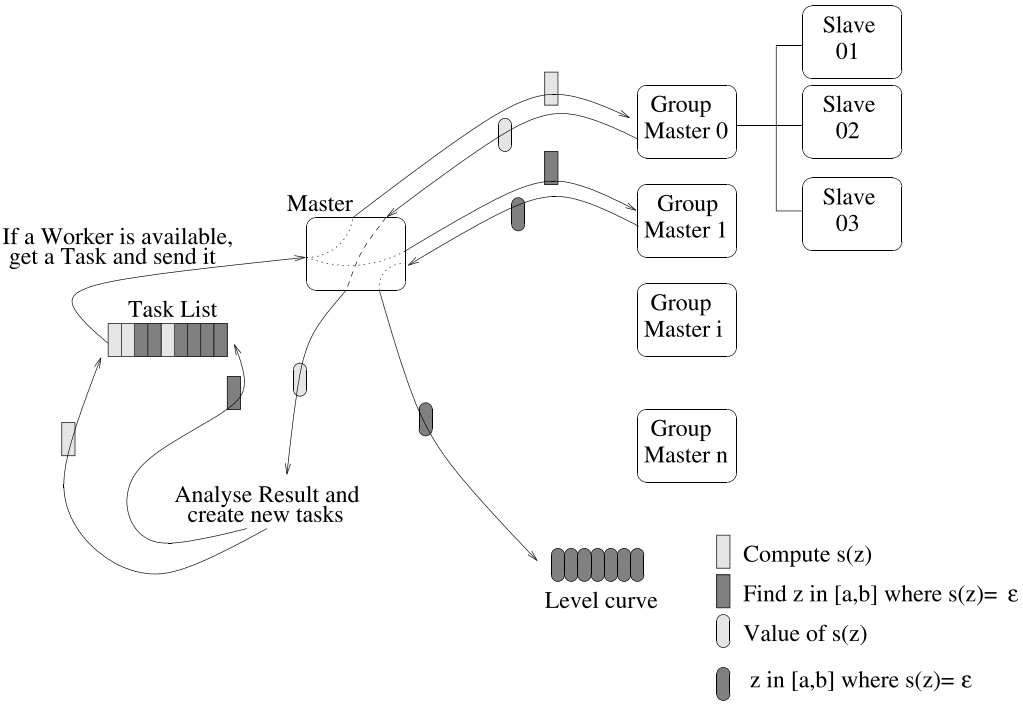

Fig. 5. Parallel structure for very large matrices.

The multifrontal QR decomposition allows a large granularity for parallelism. In [20], Puglisi developed a parallel multifrontal QR decomposition for shared memory machines. Therefore, she uses the fact that a processor can access at low cost the frontal matrix of any given leaf in the elimination tree. Unfortunately, this is not true in our case since we are using a distributed memory virtual machine. To cope with this difficulty we use the fact that the reduction of a given leaf only affects the parent node. Therefore, we isolate independent subtrees of the elimination tree which can be reduced in parallel (see Fig. 6). The group master needs to send the subtree nodes to a given slave and retrieves the final contribution block. Notice that in our implementation, the group master reorders the subtree nodes in their reduction order. The slave retrieves a set of nodes and operates the corresponding Householder reflections.

The group master builds the elimination tree and isolates the largest independent subtrees having fewer than q nodes. An estimation of the reduction cost is used to reorder the reduction tasks allowing better load balancing among the slaves. Each slave retrieves a subtree, reduces all subtree nodes and returns the contribution block to the group master. Upon reception of a contribution block for node i, the group master checks the possibility of reducing the ancestors of node i. This is feasible if and only if all the subtrees having i as a common ancestor have been reduced. Once all the subtrees are reduced, the group master completes the decomposition by reducing the remaining tree leaving the rows of R scattered among the group processors (master and slave processors).

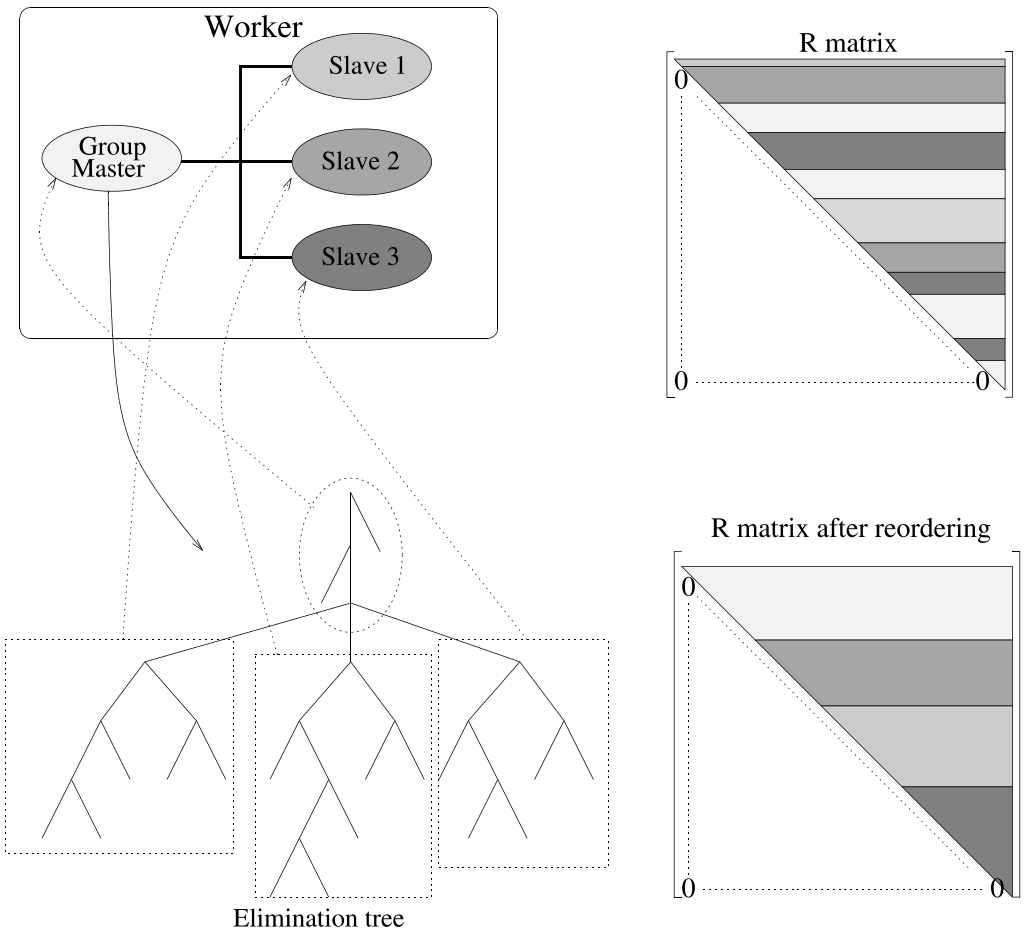

Fig. 6. Parallel multifrontal QR decomposition.

The most expensive operations in the Lanczos algorithm are the solutions of the two systems

$$w_{j,1} = R^{-1}v_{j,2} - \beta_j v_{j-1,1}$$

and

$$w_{j,2} = R^{-H}v_{j,1} - \beta_j v_{j-1,2}.$$

The systems are solved at each iteration of the Lanczos algorithm. Therefore, parallel solvers must be used to enhance the performance [3]. In the present implementation, we use a parallel block substitution routine to solve the triangular systems corresponding to $R^{-1}v_{j,2}$ and $R^{-H}v_{j,1}$ efficiently. The columns of R are reordered, as shown in Fig. 6, to allow larger block sizes and better performance.

Notice that for the Lanczos algorithm we have also considered the possibility of gathering the R matrix on two slaves, where a slave solves the linear systems corresponding to

$$w_{j,1} = R^{-1}v_{j,2} - \beta_j v_{j-1,1}$$

while the other performs

$$w_{j,2} = R^{-H}v_{j,1} - \beta_j v_{j-1,2}.$$

However this technique did not improve the performance of the code due to the lack of physical memory and the use of the swap disk space.

6. Test problems and numerical results

The master process is considered as a lightweight process since it involves a computational cost neglectible with respect to other processes. Therefore, the master process and the first worker run on the same physical processor. Hence the most interesting numerical tests are shown. All computations are performed using complex double precision arithmetic on a set of low cost general purpose heterogeneous network of workstations. The underlying hardware and network configurations are described in Table 3. The source code which is written in C, is available via anonymous ftp from the site `ftp.irisa.fr/local/aladin/philippe/PPAT`.

The investigation was limited to the worst scenario case, where a single level curve slice is computed. This is done in order to evaluate a lower bound of the speedup and efficiency of the PPAT. Additional speedups and better efficiencies are achieved when multiple level curves or level curve slices are computed simultaneously.

6.1. Standard test matrices

The test matrices shown in Table 4 were selected from Netlib's matrix market suite (`www.netlib.org`). The application uses up to 20 single processor workers. Figs. 7–9 show the wall-clock time, speedup and efficiency to compute 100 points on a given level curve for the PAT and PPAT. The numerical constants are shown in Table 5, where η is the bisection algorithm precision, η_σ is the error tolerance

Table 3
Description of the machine hardware configuration

Single processor	Pentium III	600 MHz
RAM	128 MB	
Internal cache	512 KB	
Swap space	600 MB	IDE Hard disk
Network card	3Com	10–100BaseT
HUB	3Com Super Stack	10BaseT
OS	Linux	Red Hat 6.0
Compiler	gcc	2.95.2
Communications	mpich	1.2
LAPACK	f2c translated	1.2
BLAS	lsblaspii1.2b_11.99.a	http://www.cs.utk.edu/~ghenry/distrib

Table 4
Some test matrices chosen from the non-Hermitian eigenvalue problem collection (`http://math.nist.gov/MatrixMarket/data/NEP/`)

Matrix	Collection	Dimension	N_z	$\|\|A\|\|_F$
Olm1000	NEP	1000	3996	1.3×10^6
Rdb3200L	NEP	3200	18 880	2.8×10^3
Dw8192	NEP	8192	41 746	1.6×10^3

for the Lanczos algorithm, q is the total number of $\sigma_{\min}$ evaluations per bisection, $S_{\max}$ is the speedup upper bound and $t_{\sigma_{\min}}$ is the average time needed to compute $\sigma_{\min}(A - zI)$.

The best performance was observed for the matrix `Dw8192`, that was the largest of the set. This is explained by the fact that this would have the smallest expected ratio of communication to computation. A speedup of 10.85 using 12 processors with a 90% efficiency was achieved. The best observed speedup was bound by 11 whereas the theoretical limit reaches 14.03. Some level curves of the pseudospectrum of `Dw8192` are presented in Fig. 11.

The worst results where for the matrix `Olm1000`. The speedup is only 8.31 for 13 processors with an efficiency of 63%. In a more realistic approach, we have used a graphical front-end, presented in [18], to drive the computation of the pseudospectrum with 20 processors used to compute simultaneously seven slices of the same level curve $\epsilon = 0.4$. A new slice was started whenever idle processors were observed.

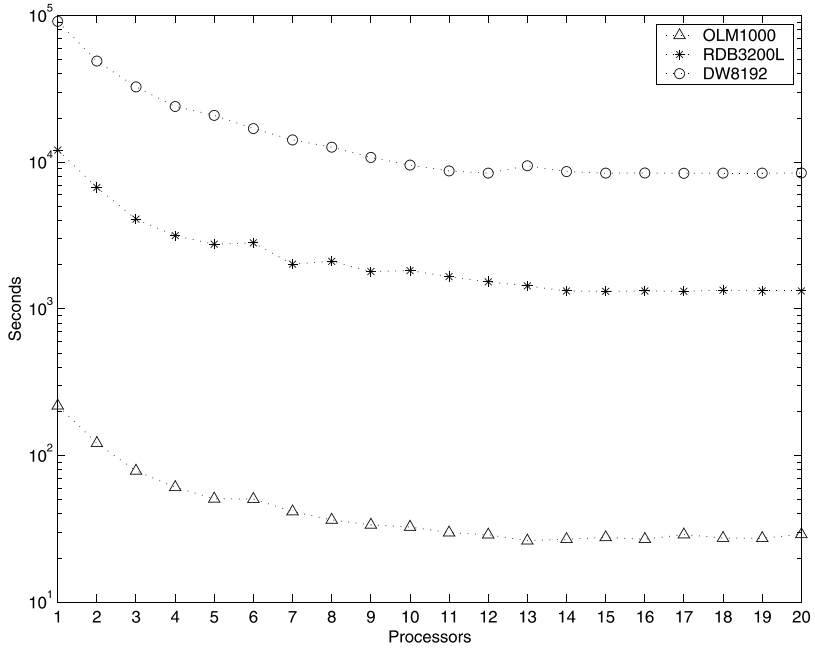

Fig. 7. Seconds to compute 100 points on the level curve.

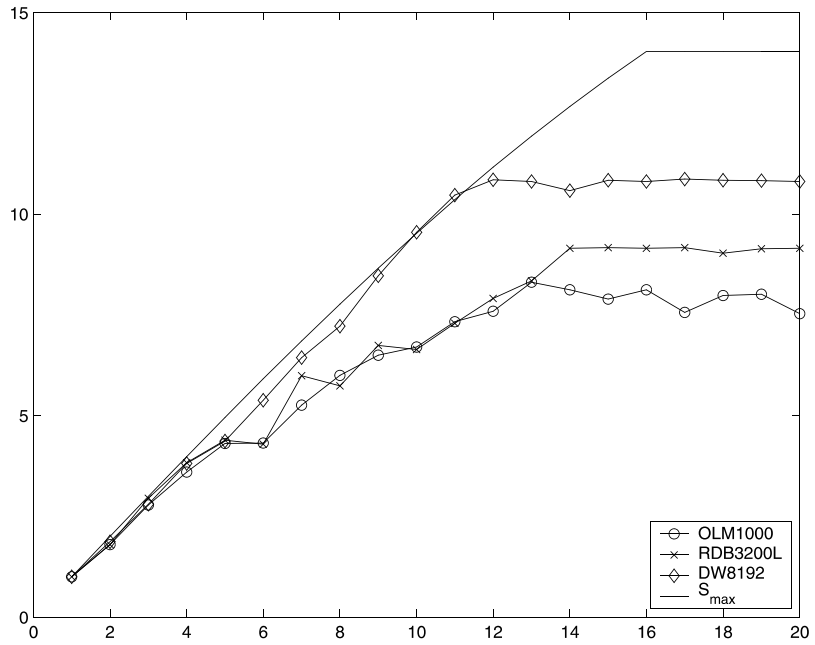

Fig. 8. Speedup of the PPAT.

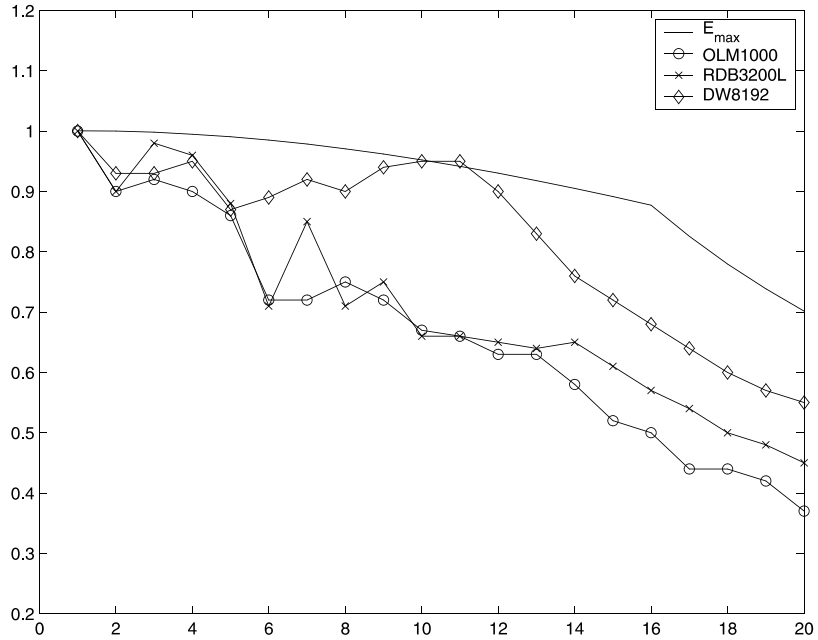

Fig. 9. Efficiency of the PPAT.

Table 5
Numerical constants

Matrix	Level curve	τ	η	η_σ	q	$S_{\max}$	$t_{\sigma_{\min}}$
Olm1000	6.1778×10^{-2}	0.01	10^{-4}	10^{-4}	7	16	0.27 s
Rdb3200L	2.0134	0.01	10^{-4}	10^{-4}	7	16	15.14 s
Dw8192	4.6352	0.01	10^{-4}	10^{-4}	7	16	114.36 s

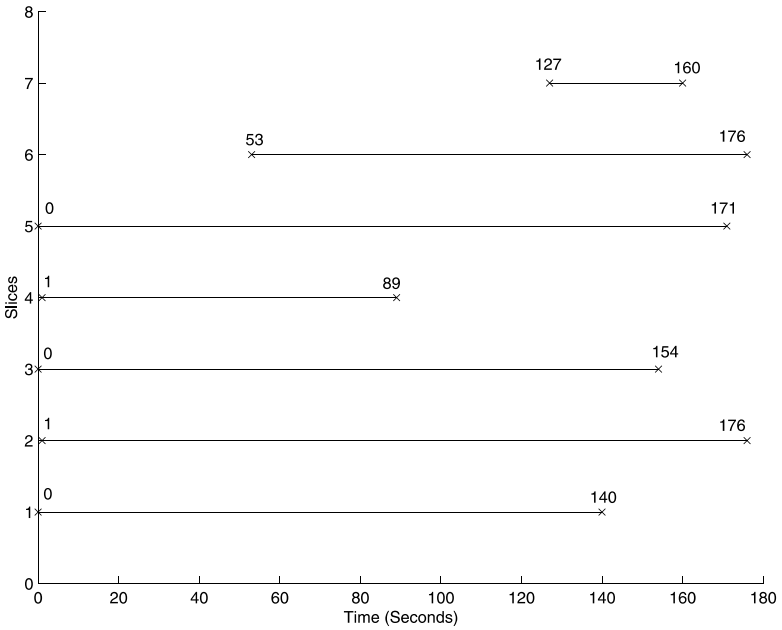

Fig. 10. Time diagram for the different level curve slices – Matrix `Olm1000`.

A total of 176 s were spent on the computation of the 4380 level points. A single processor required 2944 s to compute the same level curve. The corresponding speedup is 16.72 with an efficiency of 83%. Table 6 presents the numerical values and Fig. 10 shows the time diagram for each level curve slice.

6.2. Parallel computation of $\sigma_{\min}$

In this section, we describe a test of large dimension on which it is worth to consider a parallel computation of $\sigma_{\min}$. Although, the test is obtained from a practical situation, we do not discuss here the physical interpretation of the results. The problem has been defined and studied by Daube, Le Quéré and Tuckermann (LIMSI, Orsay). They consider the problem of a stability investigation of Navier–Stokes equations.

For a given perturbation x, the problem consists of considering the stability of the time dependent solution in a neighborhood of a steady solution. For this purpose,

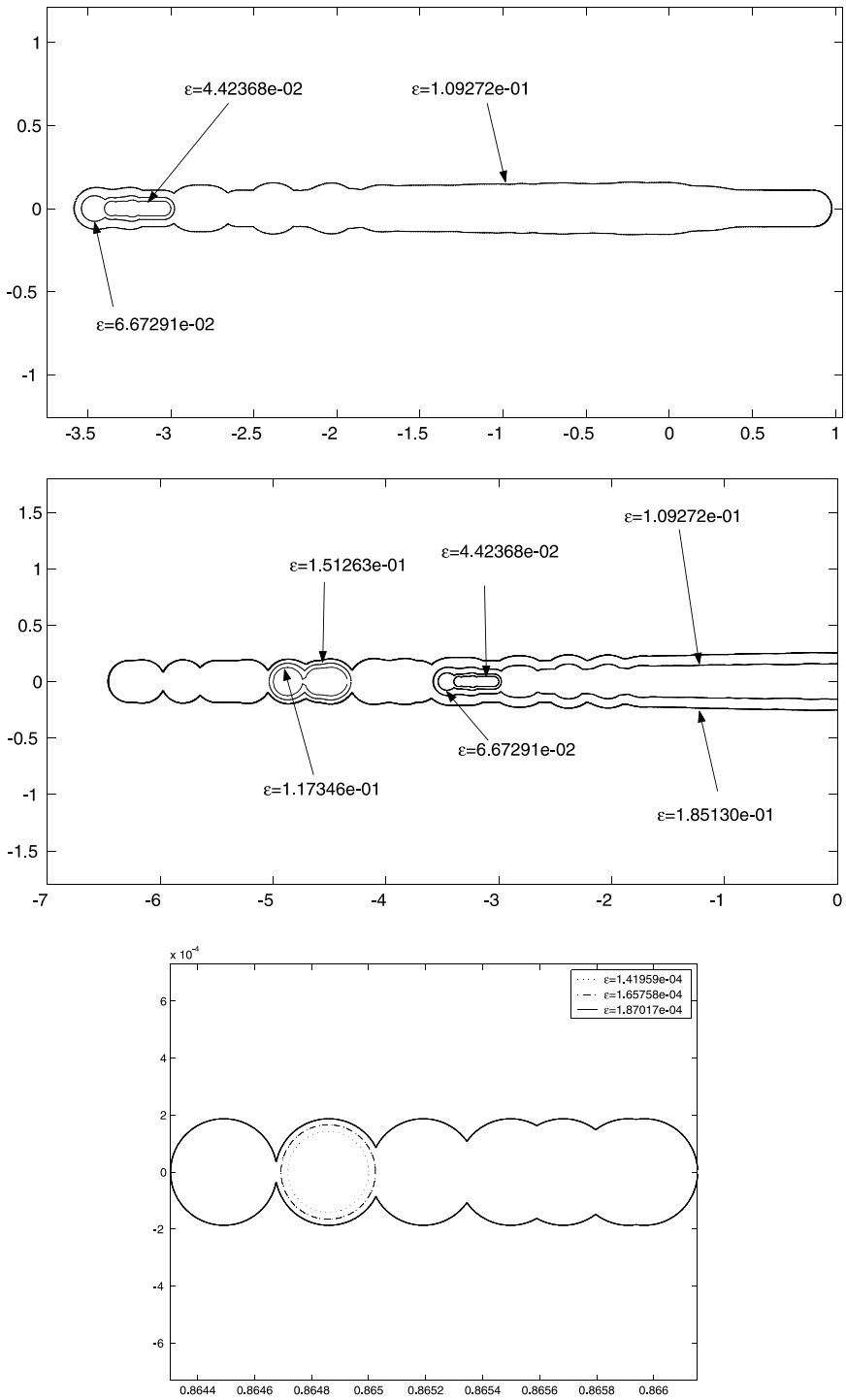

Fig. 11. Some level curves for `Dw8192`.

Table 6
Computing multiple slices of the same matrix – Test matrix Olm1000

Matrix	ϵ	τ	q	t_1	t_{20}	S_{20}	E_{20}
Olm1000	0.4	0.1	7	2944 s	176 s	16.72	0.83

Table 7
The test matrix set

Test	Dimension	$N_z(A)$	$N_z(L)$
S_1	1890×1890	23 078	5514
S_2	3906×3906	49 030	11 530
S_3	32 130×32 130	411 590	95 754

the eigenvalues of largest real part of some Jacobian matrix are studied. Therefore the pseudospectrum of the matrix in a the neighborhood of the origin is computed. As shown in Table 7, we conducted three tests with matrices of different sizes. For tests S_1 and S_2, each worker is assigned to a single processor while in S_3, because of the size of the matrix, three processors were assigned to the computation of each $\sigma_{\min}$, as shown in Fig. 5.

Table 8 shows the wall-clock time needed to compute $s(z)$ on a single processor. Test S_3 requires 3832 s to compute a single $s(z)$. Notice that for tests S_1 and S_2, the Lanczos algorithm accounts for 10% of the total time, whereas in the case of S_3, it accounts for 40% of the total time. This increase is due to swap disk effects. This is the main reason for considering parallelism within the singular value computation for large matrices. Therefore, parallel implementations of the multifrontal QR decomposition and the Lanczos algorithm should be considered.

Table 9 shows the computation time for the parallel versions of the multifrontal QR, Lanczos algorithm and $s(z)$ on a worker of three processors. The processors were distributed into one group master and two slaves. It can be observed that parallelizing the computation of $\sigma_{\min}$ is not beneficial for S_1 and S_2 since there is a slowdown. For S_3, a speedup of 1.66 and 2.29 were achieved in the QR decomposition and Lanczos algorithm, respectively. The total speed up for $s(z)$ is 1.85 and the corresponding efficiency is 61%.

We were able to compute 42 points of a level curve for the test S_3 in about 12 h at a progress rate of 1 point per 1040 s (17.3 min). Without the parallel implementations of the QR decomposition and the Lanczos algorithm, we would have been limited to a progress rate of 1 point per each 32 min.

Table 8
Seconds spent in the multifrontal QR decomposition and the Lanczos algorithm

Test	Multifrontal QR	Lanczos algorithm	$\sigma_{\min}(A - zI)$
S_1	3.7 (84%)	0.7 (16%)	4.4
S_2	41.8 (95%)	2.1 (05%)	43.9
S_3	2353 (61%)	1480 (39%)	3832

Table 9
Parallel implementation of the numerical algorithms

Test	1 Proc.	3 Procs.	Speedup	Efficiency
Multifrontal QR decomposition				
S_1	3.7	3.62	1.02	0.34
S_2	41.8	30.83	1.35	0.45
S_3	2353	1410	1.66	0.55
Lanczos algorithm				
Test S_1	0.7	5.24	0.13	0.04
Test S_2	2.1	15.62	0.14	0.05
Test S_3	1479	647	2.29	0.76
$\sigma_{\min}(A - zI)$				
Test S_1	4.4	8.86	0.49	0.16
Test S_2	43.9	46.45	0.94	0.32
Test S_3	3822	2057	1.85	0.61

7. Conclusion

The PPAT, just like its sequential predecessor, the PAT, offers total reliability. It is shown that PPAT offers large grain parallelism. Based on a master–slave model, the PPAT provides large speedups and high efficiency. Furthermore, the PPAT is capable of following multiple level curves simultaneously and exploits the symmetry of the level curve in the case of real matrices to reduce the total computational cost.

Parallel implementations of the Lanczos algorithm and the multifrontal QR decomposition are used to compute the smallest singular value of large matrices (up to $32\,130 \times 32\,130$ in this paper). Coupled with the PPAT, these implementations yield a highly parallel code. Although the observed performance numbers do not allow the computation of a pseudospectrum of large matrices (order higher than 3×10^4), the presented methods are promising.

References

[1] P.R. Amestoy, T.A. Davis, I.S. Duff, An approximate minimum degree ordering algorithm, SIMAX 17 (1996) 886–905.
[2] P.R. Amestoy, I.S. Duff, C. Puglisi, Multifrontal QR factorization in a multiprocessor environment, Int. J. Numer. Linear Algebra Appl. 3 (1996) 275–300.
[3] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, Comput. Methods Appl. Mech. Eng. (2000) 501–520.
[4] C. Bekas, E. Gallopoulos, Cobra: Parallel path following for computing the matrix pseudospectrum, Parallel Comput., to appear.
[5] R. Brent, Algorithms for Minimization Without Derivatives, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[6] M. Brühl, A curve tracing algorithm for computing the pseudospectrum, BIT 36 (3) (1996) 441–454.
[7] V. Fraysse, L. Giraud, V. Toumazou, Parallel computation of spectral portrait on the Meiko CS2, in: H. Liddell, A. Colbrook, B. HertzbergerP. Sloot (Eds.), High-Performance Computing and Networking, vol. 1067, Springer, Berlin, 1996, pp. 312–318.

[8] A. Geist, E. Ng, Task scheduling for parallel sparse Cholesky factorization, Int. J. Parallel Programming 18 (1989) 291–314.
[9] A. George, J.W.H. Liu, A fast implementation of the minimum degree ordering algorithm using quotient graphs, ACM Trans. Math. Software 6 (1980) 337–358.
[10] S.K. Godunov, Spectral portraits of matrices and criteria of spectrum dichotomy, in: L. Atanassova J. Hezberger (Eds.), 3rd International IMAGS-CAMM Symposium on Computer Arithmetic and Enclosure Methods, Amsterdam, 1992.
[11] M.T. Heath, E. Ng, P.W. Peyton, Parallel algorithms for sparse linear systems, SIAM Rev. 33 (1991) 420–460.
[12] J. Huilfieldt, A. Ruhe, A new algorithm for numerical path following applied to an example from hydro-dynamical flow, SIAM J. Sci. Statist. Comput. 11 (1990) 1181–1192.
[13] S.H. Lui, Computation of pseudospectra by continuation, SIAM J. Sci. Comput. 18 (2) (1997) 565–573.
[14] R. Mejia, CONKUB: A conversational path-follower for systems of nonlinear equations, J. Comput. Phys 63 (1986) 67–84.
[15] V. Heuveline, B. Philippe, M. Sadkane, Parallel computation of spectral portrait of large matrices by Davidson type methods, Numer. Algorithms 16 (1997) 55–75.
[16] V. Marques, V. Toumazou, Spectral portrait computation by a Lanczos method (Augmented matrix version), Tech. Rep. TR/PA/95/05, Cerfacs, Touluse, France, 1995.
[17] P. Matstoms, Parallel sparse *QR* factorization on shared memory architectures, Parallel Comput. 21 (1995) 473–486.
[18] D. Mezher, A graphical tool for driving the parallel computation of the Pseudo-Spectra, in: 15th ACM International Conference on Super Computing, Sorrento, Italy, June 2001.
[19] D. Mezher, B. Philippe, PAT – a reliable path following algorithm, Numer. Algorithms, forthcoming.
[20] C. Puglisi, *QR* factorization of large sparse over determined and square matrices using the multifrontal method in a multiprocessor environment, Cerfacs Report, Touluse, France, Ref: TH/PA/93/33, 1993.
[21] Y. Saad, Numerical methods for large eigenvalue problems, in: Manchester University Press Series in Algorithms and Architectures for Advanced Scientific Computing, 1991.
[22] H. Schwetlick, G. Timmermann, R. Lösche, Path following for large nonlinear equations by implicit block elimination based on recursive projections, in: J. Renegar, M. ShubS. Smale (Eds.), The mathematics of Numerical Analysis. Proceedings of the AMS-SIAM Summer Seminar Park City/Utah 1996, Lectures in Applied Mathematics, vol. 32, American Mathematical Society, Providence, RI, 1996, pp. 715–732.
[23] L.N. Trefethen, Pseudospectra of matrices, in: D.F. GriffithsG.A. Watson (Eds.), Numerical Analysis, Dundee 1991, Longman, Harlow, Essex, UK, 1992, pp. 234–266, MR 93e:65064.
[24] L.N. Trefethen, in: Computation of Pseudospectra, Acta Numerica, Cambridge University Press, Cambridge, 1999, pp. 247–295.
[25] C.W. Ueberhuber, Numerical Computation 2, Methods, Software and Analysis, Springer, Berlin, 1997.
[26] T. Wright, L.N. Trefethen, Large-scale computation of Pseudospectra using ARPACK and eigs, Oxford University Computing Laboratory, Numerical Analysis Group, Oxford, England, Report no. 00/11, June 2000.