Parallel Computation of Pseudospectra using Transfer Functions on a MATLAB-MPI Cluster Platform

C. Bekas[1], E. Kokiopoulou[1], E. Gallopoulos[1], and V. Simoncini[2]

[1] Computer Engineering and Informatics Department
University of Patras, Greece
[knb,exk,statis]@hpclab.ceid.upatras.gr
[2] Mathematics Department
University of Bologna, Italy
val@dragon.ian.pv.cnr.it

Abstract. One of the most expensive problems in numerical linear algebra is the computation of the ϵ-pseudospectrum of matrices, that is, the locus of eigenvalues of all matrices of the form $A + E$, where the norm of E is bounded by ϵ. Several research efforts have been attempting to make the problem tractable by means of better algorithms and utilization of all possible computational resources. One common goal of these efforts is to bring to users the power to extract pseudospectrum information from their applications, on the computational environments they generally use, at a cost that is sufficiently low to render these computations routine. To this end, we investigate a scheme based on $i)$ iterative methods for computing pseudospectra via approximations of the resolvent norm, with $ii)$ a computational platform based on a cluster of PCs and $iii)$ a programming environment based on MATLAB enhanced with MPI functionality and show that it can achieve high performance for problems of significant size.

1 Introduction and motivation

Let $A \in \mathbb{C}^{n\times n}$ have singular value decomposition (SVD) $A = U\Sigma V^*$, and let $\Lambda(A)$ be the set of eigenvalues of A. The $\epsilon-$pseudospectrum $\Lambda_\epsilon(A)$ (pseudospectrum for short) of a matrix is the locus of eigenvalues of $\Lambda(A+E)$, for all possible E such that $\|E\| \leq \epsilon$ for given ϵ and norm. When A is normal (i.e. satisfies the relation $AA^* = A^*A$), the pseudospectral regions are readily computed as the union of the disks of radius ϵ surrounding each eigenvalue of A. The pseudospectrum becomes of interest when A is nonnormal see [1], references therein and [2] for a package for pseudospectra computation using PVM. An important barrier in providing pseudospectral information routinely is the expense involved in their calculation. In the sequel we use the 2-norm for matrices and vectors, $R(z) = (A - zI)^{-1}$ is the resolvent of A and e_k the kth standard unit vector. Also $s(x, y) := \sigma_{\min}(xI + iyI - A)$, herein written compactly as $s(z)$ for $z = x + iy$, is the minimum singular value of matrix $zI - A$. Two equivalent definitions of the $\epsilon-$pseudospectrum are

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} : \sigma_{\min}(A - zI) \leq \epsilon\} = \{z \in \mathbb{C} : \|R(z)\| \geq \frac{1}{\epsilon}\}. \qquad (1)$$

Relations (1) immediately suggest an algorithm, referred to as GRID, for the computation of $\Lambda_\epsilon(A)$ that consists of the following steps: $i)$ Define grid Ω_h on a region of the complex plane that contains $\Lambda_\epsilon(A)$. $ii)$ Compute $s(z_h) := \sigma_{\min}(z_h I - A)$ or $R(z_h)$, $\forall z_h \in \Omega_h$. $iii)$ Plot ϵ-contours of $s(z)$ or $R(z)$. GRID's cost is modeled as

$$C_{\texttt{GRID}} = |\Omega_h| \times C_{\sigma_{\min}}, \tag{2}$$

where $|\Omega_h|$ is the number of nodes of the mesh and $C_{\sigma_{\min}}$ is the average cost of computing $\sigma_{\min}$. GRID is simple, robust and embarassingly parallel, since the computations of $s(z_h)$ are completely decoupled between gridpoints. As cost formula (2) reveals, its cost increases rapidly with the size of the matrix and the number of gridpoints in Ω_h. For example, using MATLAB version 6.1, the LAPACK 3.0 based svd function takes $19\ \mathrm{sec}$ to compute the SVD of a 1000×1000 matrix on a Pentium-III at 866 MHz. Computing the pseudospectrum on a 50×50 mesh would take almost 13 hours or about 7, if A is real and one use the fact that then, the pseudospectrum is symmetric with respect to the real axis. Exploiting the embarassingly parallel nature of the algorithm, it would take over 150 such processors to bring the time down to $5\ \mathrm{min}$ (or 2.5, in case of real A). As an indication of the state of affairs when iterative methods are used, MATLAB's svds (based on ARPACK) on the same machine required almost $68\ \mathrm{sec}$ to compute $s(z)$ for Harwell-Boeing matrix e30r0100 (order $n = 9661$, $\mathtt{nnz} = 306002$ non-zero elements) and random z; furthermore, iterative methods over domains entail interesting load balancing issues; cf. [3, 4]. The above underline the fact that computing pseudospectra presents a formidable computational challenge since typical matrix sizes in applications can reach the order of hundreds of thousands. Cost formula (2) suggests that we can reduce the complexity of the calculation by *domain-based* methods, that seek to reduce the number of gridpoints where $s(z)$ must be computed, and *matrix-based* methods, that target at reducing the cost of evaluating $s(z)$.

Given the complexity of the problem, it also becomes necessary that we use any advances in high performance computing at our disposal. All of the above approaches are the subject of current research.[3] Therefore, if the pseudospectrum is to become a practical tool, we must bring it to the users on the computational environments they generally use, at an overall cost that is sufficiently low to render these computations routine. Here we present our efforts towards this target and contribute a strategy that combines: Parallel computing on clusters of PCs for high performance at low cost, communication using MPI-like calls, and programming using MATLAB, for rapid prototyping over high quality libraries with a very popular system. Central to our strategy is the Cornell Multitasking Toolbox for MATLAB (CMTM) ([5]) that enhances MATLAB with MPI functionality [6]. In the context of our efforts so far, we follow here the mathematical foundations of the "transfer function framework" described in [7]. Some first results, relating the performance of the transfer function framework on an 8 processor Origin-2000 ccNUMA system were presented in [8]. In [3], we described the first use of the CMTM system in the context of domain-based algorithms developed by our group as well as some load balancing issues that arise in the context of domain and matrix methods. In this paper, we extend our efforts related to the iterative computation of pseudospectra using the *transfer function framework*.

[3] See also the site URL http://web.comlab.ox.ac.uk/projects/pseudospectra.

Table 1. Available CMTM commands (version 0.82) (actual commands prefixed with MMPI_).

Abort, Barrier, Bcast, Comm_dup, Comm_free, Comm_rank
Comm_size, Comm_split, Gather, Iprobe, Irecv, Recv
Reduce, Scatter, Scatterv, Send, Testany, Wtime

1.1 Computational environment

As a representative of the kind of computational cluster that many users will not have much difficulty in utilizing, in this paper we employ up to 8 uniprocessor Pentium-III @ 933MHz PCs, running Windows 2000, with 256KB cache and 256MB memory, connected using a fast Ethernet switch. We use MATLAB (v. 6.1) due to the high quality of its numerics and interface characteristics as well as its widespread use in the scientific community. The question arises, then, how to use MATLAB to solve problems such as the pseudospectrum computation in parallel? We can, for example translate MATLAB programs into source code for high performance compilers e.g. [9], or use MATLAB as interface for calls to distributed numerical libraries e.g. [10]. Another approach is to use multiple, independent MATLAB sessions with software to coordinate data exchange. This latter approach is followed in the "Cornell Multitasking Toolbox for MATLAB" [5], a descendant of the MultiMATLAB package [11]. CMTM enables multiple copies of MATLAB to run simultaneously and to exchange data arrays via MPI-like calls. CMTM provides easy-to-use MPI-like calls of the form `A = MMPI_Bcast(A);` Each MMPI_ function is a MATLAB mex file, linked as a DLL with a commercial version of MPI, MPI/Pro v. 1.5, see [12]. Like MATLAB, CMTM functions take as arguments arrays. As the above broadcast command shows, CMTM calls are simpler than their MPI counterparts. The particular command is executed by all nodes; the system automatically distinguishes whether a node is a sender or a receiver and performs the communication. Similar are the calls to other communication procedures such as blocking send-receive and reduction clauses. For example, in order to compute a dot product of two vectors x and y that are distributed among the processors, all nodes execute the call `MMPI_Reduce(x'*y, MPI_ADD);` Each node performs the local dot product and then the system accumulates all local sub-products to a root node. Table 1 lists the commands available in the current version of CMTM. Notice the absence of the MPI ALL_ type commands (e.g. Allreduce, Allgather). Whenever these commands are needed, we simulate them, e.g. using reduce or gather followed by broadcast.

Baseline measurements In order to better appreciate the results for our main algorithm, we provide some baseline measurements for a few indicative collective communication operations on our computational cluster. We note that for all experiments in this work the cluster was in single user mode. Table 2 contains the measured timings with varying number of processors and problem sizes when using CMTM to broadcast vectors and also to reduce by summation. As expected, MMPI_Reduce is always more time consuming than MMPI_Bcast, while both take longer as the cluster size increases.

Table 2. Timings (in sec) for broadcasting and reducing summation of vectors of size 40000:20000:100000 using `CMTM` on $p = 2, 4, 8$ processors.

	`MMPI_Bcast`				`MMPI_Reduce`			
p	40000	60000	80000	100000	40000	60000	80000	100000
2	0.03	0.05	0.06	0.08	0.05	0.05	0.06	0.081
4	0.07	0.09	0.12	0.15	0.09	0.13	0.13	0.16
8	0.09	0.13	0.17	0.22	0.13	0.15	0.19	0.24

2 The transfer function framework

An early, interesting idea for the economical approximation of the pseudospectrum, presented in [13], was to use Krylov methods and obtain approximations to $\sigma_{\min}(zI - A)$, from $\sigma_{\min}(zI - H)$, where H was either the square or the rectangular upper Hessenberg matrix that is obtained during the Arnoldi iteration and I is the identity matrix or a section thereof to conform in size with H. A practical advantage of this approach is that, whatever the number of gridpoints, it requires only one (expensive) reduction of A to upper Hessenberg form. It still requires, however, the (independent) computations of $\sigma_{\min}(z_h I - H)$, for every gridpoint z_h, since, unlike eigenvalues singular values do not respect the shift property, i.e. in general, $\sigma(zI - A) \neq z - \sigma(A)$. The above approach was further refined in [14]. Let now D^*, E be full rank - typically long and thin - matrices, of row dimension n. Matrix $G_z(A, E, D) := DR(z)E$ is called in the literature *transfer function*. It was shown in [7] that the transfer function provides a useful framework for describing existing and defining new methods for computing pseudospectra. The central method in [7] was based on the selection $D = W_m^*$ and $E = W_{m+1}$, where $W_m = [w_1, \ldots, w_m]$ denotes the orthonormal basis for the Krylov subspace $\mathcal{K}_m(A, w_1)$ constructed by the Arnoldi iteration, so that

$$AW_m = W_m H_{m,m} + h_{m+1,m} w_{m+1} e_m^\top, \tag{3}$$

where $H_{m,m}$ is the square upper Hessenberg matrix consisting of the first m rows of $H_{m+1,m}$. We note that $W_{m+1} = [W_m, w_{m+1}]$ and $H_{m+1,m} = [H_{m,m}; h_{m+1,m} e_m^\top]$. We will be referring to m as the "transfer function dimension" and will be writing $G_{z,m}(A)$ for $G_z(A, W_{m+1}, W_m^*)$. With the afore-mentioned selection for D and E, for large enough m we have $\|R(z)\| \approx \|G_z(A, W_{m+1}, W_m^*)\|$, which provides an approximation to $\|R(z)\|$ that improves monotonically with m and would be used in the sequel to construct the pseudospectrum based on relation (1); cf. [7]. Even though computation of $G_z(A, W_{m+1}, W_m^*) = W_m^*(A - zI)^{-1} W_{m+1}$ appears to require solving $m + 1$ linear systems of size n, a trick allows us to reduce the number of (large) linear systems to one because

$$G_{z,m}(A) = [(\tilde{I} - h_{m+1,m} \phi_z e_m^\top)(H_{m,m} - zI)^{-1}, \phi_z)], \tag{4}$$

where $\phi_z = W_m^*(A - zI)^{-1} w_{m+1}$; cf. [7]. Therefore, in order to compute $\|G_{z,m}(A)\|$ we have to solve a single (large) linear system of size n for each shift z; furthermore,

Table 3. Arnoldi performance (in sec) for $p = 1$ to 8 processors using CGS, MGS and CGS with reorthogonalization on random sparse matrices of size n.

Arnoldi	$p \setminus n$	40000	60000	80000	100000	$p \setminus n$	40000	60000	80000	100000
CGS	1	328	512	1716	5885	2	150	241	340	440
MGS		362	560	1453	3126		172	265	365	476
CGSo		389	606	1780	6962		181	289	405	522
CGS	4	80	126	179	235	8	53	80	113	143
MGS		129	158	209	268		92	125	188	201
CGSo		101	150	211	275		64	97	135	164

the right hand side, w_{m+1}, remains the same for each shift, something that will be exploited by the iterative solver (cf. the next section). We also need to solve m Hessenberg systems of size m, and to compute the norm of $G_{z,m}(A)$. We discuss the actual parallel implementation of this approach for our cluster environment in Section 2.1.

The Arnoldi iteration As with most modern iterative schemes, the effective implementation of the transfer function methodology uses the Arnoldi iteration, via an implementation of relation (3), as a computational kernel, to create an orthogonal basis for the Krylov subspace [15]. We organize the iteration around a row-wise partition of the data: Each processor is assigned a number of rows of A and V. At step j, we store the entire $V(:, j)$ that must be orthogonalized into each processor. This requires a broadcast, at some point during the iteration, but allows thereafter the matrix-vector multiplication to take place locally on each processor. Due to lack of space we do not go into further details but tabulate, in Table 3, results from experiments on our cluster and programming environment with three different types of Arnoldi iteration: CGS-Arnoldi, MGS-Arnoldi and CGSo-Arnoldi, where the first is based on classical Gram-Schmidt (GS) orthogonalization, the next on modified GS and the last on classical GS with selective reorthogonalization. Notice the superlinear speedup for the larger matrices due to the disribution of the large Krylov bases. The random sparse matrices were created using the `MATLAB`'s built-in function `sprand(n,n,p)` with p equal to $3e - 4$.

2.1 `TRGRID`: `GRID` with transfer functions

The property that renders Krylov type linear solvers particularly suitable in the context of the transfer function framework for pseudospectra, in particular formula (4), is the shift invariance of Krylov subspaces, i.e. the fact that $\mathcal{K}_d(A, w_1) = \mathcal{K}_d(A - zI, w_1)$ for any z. This means that we can reuse the same basis that we build for $\mathcal{K}_d(A, w_1)$, say, to solve linear systems with "shifted" matrices, e.g. $A - zI$; see for example [16]. Since our interest is in nonnormal matrices, we used `GMRES` as the linear solver. The pseudocode for our parallel algorithm is listed in Table 4.

Remember that after completing the (parallel) Arnoldi iterations (Table 4, line 1), with the row distribution described earlier, the rows of the bases W and $\hat{W}$ are distributed among the cluster nodes. For the sake of economy in notation we use the same

Table 4. Pseudocode for the parallel `TRGRID` algorithm for the `CMTM` environment

```
Parallel TRGRID
(* Input *) Points z_i, i = 1, ..., M, vector w_1
      with ||w_1|| = 1, scalars m, d.
1.  All nodes work on two parallel Arnoldi iterations
      [W_{m+1}, H_{m+1,m}] ← arnoldi (A, w_1, m)
      [Ŵ_{d+1}, F_{d+1,d}] ← arnoldi(A, w_{m+1}, d)
      Each node has some rows of W_{m+1}, Ŵ_{d+1}
      Node zero distributes the M gridpoints
      so that processor pr_id holds points in the M/p-sized set I(pr_id)
2.  for i ∈ I(pr_id)
3.        Y(:, i) = argmin_y{||(F_{d+1,d} − z_i Ĩ_d)y − e_1||}
4.  end
5.  Node zero gathers matrix Y and broadcasts it back
6.  Each node performs local multiplication φ_{z,i} = W*_m Ŵ_d Y
7.  Node zero collects Φ_z =Reduce(φ_z,SUM)
8.  Node zero distributes back the columns of Φ_z
9.  for i ∈ I(pr_id)
10.     D_i = (Ĩ − h_{m+1,m} Φ_z(:, i) e_m^⊤)(H_{m,m} − z_i I)^{−1}
11.     ||G_{z_i}(A)|| = ||[D_i, φ_{z_i}]||
12. end
13. Node zero gathers approximations of 1/s(z_i)
```

symbolism when we deal with local computations too, while in reality we are referring only to a subset of the rows of W and $\hat{W}$. Each processor (see lines 2-4, 9-12) works on M/p gridpoints (we have assumed that p exactly divides M). In line 3, each node computes its share of the columns of matrix Y. For the next steps, in order for the local ϕ_z to be computed, all processors must access the whole Y, hence all-to-all type communication is needed. In lines 9-12 each processor works on its share of gridpoints and columns of matrix Φ_z. Finally, processor zero gathers the approximations of $\{1/s(z_i)\}_{i=1}^{M}$. What remains to clarify is the computation of $\phi_{z,i}$ and Φ_z (lines 6-7). We have $\phi_{z,i} = W_i^* \hat{W}_i Y$, where W_i and $\hat{W}_i, i = 0..., P-1$ are the subsets of contiguous rows of the basis matrices W and $\hat{W}$ that each node privately owns. Then, $\Phi_z = \sum_{i=0}^{P-1} \phi_{z,i}$ and we use reducing summation to compute Φ_z in parallel, since `CMTM` allows reduction processes to work on matrices as well as scalars. It is worth noting that the preferred order used when applying to implement the two matrix-matrix multiplications of line 6 depends on the dimensions of the local matrices $W_i, \hat{W}_i$ and on the column dimension of Y and could be decided at run time. In our experiments, we assumed that the Krylov dimensions m, d were much smaller than the size of the matrix, n, and the grid, M. `TRGRID` consists of two parts. The first part (line 1) consists of the Arnoldi iterations used to obtain orthogonal bases for the Krylov subspaces used for the transfer function and for the solution of the size n system. In forthcoming work we detail techniques that exploit the common information present in the two bases ([8]). An important advantage of the method is that after the first part is completed and basis ma-

Table 5. Performance of parallel `TRGRID`

	`gre_1107`				`pores_2`				`sprand(2000)`				`sprand(4000)`			
p	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
Time (sec)	174	76.2	36.2	18.7	340	155	76.3	38.1	244	118.8	59.3	31	501	245	123	63.9
Speedup	-	2.3	4.8	9.3	-	2.2	4.5	8.9	-	2.1	4.1	7.87	-	2.04	4.1	7.84

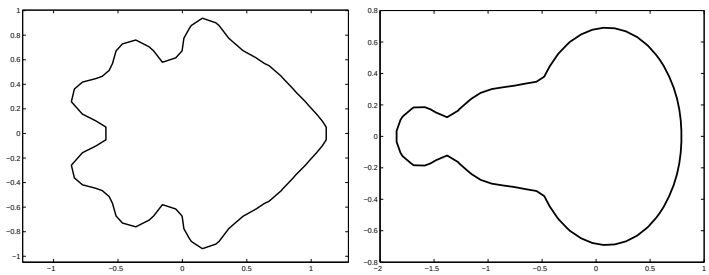

Fig. 1. Pseudospectrum contour $\partial\Lambda_\epsilon(A)$, $\epsilon = 1e-1$ for `gre_1107` (left) and `pores_2` (right).

trices W and $\hat{W}$ become available, we only need to repeat the second part, from line 2 onwards to compute the pseudospectrum. Based on this property and given the previous results for Arnoldi we measure the performance of only the second part of `TRGRID`. A second observation is that the second `TRGRID` involves only dense linear algebra. Remember also that at the end of the first part, each processor has a copy of the upper Hessenberg matrices. Hence, we can use `BLAS-3` to exploit the memory hierarchy of each processor. Furthermore, assuming that $M \geq p$, the workload would be evenly divided amongst processors. Of course, the method is always subject to load imbalances due to systemic effects. However, since the second part of parallel `TRGRID` requires only limited communication, increased network traffic is not expected to have a significant effect on performance. Let us now investigate the performance of the main body of parallel `TRGRID` on our cluster's nodes. We experimented with the HB-matrices `gre_1107` ($n = 1107, \texttt{nnz} = 5664$) and `pores_2` ($n = 1124, \texttt{nnz} = 9613$) scaled by $1e-7$, as well as with two random sparse matrices of size $n = 2000$ and 4000 with density $p = 1e-2$. On the domain $[-1.8, 1.8] \times [-1.8, 1.8]$, we employed a 50×50 grid for the HB matrices and a 30×40 grid for the random matrices. Figure 1 illustrates the $\epsilon = 0.1$ contours for the two HB matrices. The Krylov dimensions were $m = d = 75, 100, 120$ and 150 respectively. Table 5 illustrates the performance results. In all cases, except for $p = 8$ for the random matrices we observe excellent, superlinear in most cases, (absolute) speedups due to the fact that the Krylov bases are distributed.

3 Conclusions

The widespread availability of the high quality numerics and interface of `MATLAB` together with an `MPI`-based parallel processing toolbox and the results shown in this

paper for matrix-based methods and in [3] for domain based methods, indicate that the computation of pseudospectra of large matrices is rapidly becoming accessible to everyday users who prefer to program in MATLAB and whose only hardware infrastructure is a PC cluster. Further work in progress includes the incorporation of these methods into a toolbox as well as the exploitation of multithreading, testing different forms of parallelism for MATLAB as well as the enhancement of the numerical infrastructure with lower level parallel computational kernels, e.g. SCALAPACK-like. We are also working on several numerical issues that arise during the combination of the transfer function approach with domain based methods so as to permit even more effective and robust implementations [8].

References

1. Trefethen, L.: Computation of pseudospectra. In: Acta Numerica 1999. Volume 8. Cambridge University Press (1999) 247–295
2. Braconnier, T.: Fvpspack: A Fortran and PVM package to compute the field of values and pseudospectra of large matrices. Numerical Analysis Report No. 293, Manchester Centre for Computational Mathematics, Manchester, England (1996)
3. Bekas, C., Kokiopoulou, E., Koutis, I., Gallopoulos, E.: Towards the effective parallel computation of matrix pseudospectra. In: Proc. 15th ACM Int'l. Conf. Supercomputing (ICS'01), Sorrento, Italy. (2001) 260–269
4. Mezher, D., Philippe, B.: Parallel computation of the pseudospectrum of large matrices. Parallel Computing **28** (2002) 199–221
5. Zollweg, J., Verma, A.: The Cornell Multitask Toolbox. Directory Services/Software/CMTM at URL http://www.tc.cornell.edu (2002)
6. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message Programming Interface. second edn. MIT Press, Cambridge, MA (1999)
7. Simoncini, V., Gallopoulos, E.: Transfer functions and resolvent norm approximation of large matrices. Electronic Transactions on Numerical Analysis (ETNA) **7** (1998) 190–201
8. Bekas, C., Gallopoulos, E., Simoncini, V.: On the computational effectiveness of transfer function approximations to the matrix pseudospectrum. In: Proc. Copper Mountain Conf. Iterative Methods. (Apr. 2000) Full paper in preparation.
9. DeRose, L., et al.: FALCON: A MATLAB Interactive Restructuring Compiler. In C.-H. Huang, et al., ed.: Lecture Notes in Computer Science: Languages and Compilers for Parallel Computing. Springer-Verlag, New York (1995) 269–288
10. Casanova, H., Dongarra, J.: NetSolve: A network server for solving computational science problems. Int'l. J. Supercomput. Appl. and High Perf. Comput. **11** (1997) 212–223
11. Menon, V., Trefethen, A.: MultiMATLAB: Integrating MATLAB with high-performance parallel computing. In: Proc. Supercomputing'97. (1997)
12. MPI Software Technology: (MPI/Pro) http://www.mpi-softtech.com.
13. Toh, K.C., Trefethen, L.: Calculation of pseudospectra by the Arnoldi iteration. SIAM J. Sci. Comput. **17** (1996) 1–15
14. Wright, T., Trefethen, L.N.: Large-scale computation of pseudospectra using ARPACK and Eigs. SIAM J. Sci. Stat. Comput. **23** (2001) 591:605
15. Dongarra, J., Duff, I., Sorensen, D., van der Vorst, H.: Numerical Linear Algebra for High-Performance Computers. SIAM (1998)
16. Frommer, A., Glässner, U.: Restarted GMRES for shifted linear systems. SIAM J. Sci. Comput. **19** (1998) 15–26