A Parallel Iterative Method for Exponential Propagation

Chuck Baldwin* Roland W. Freund† Efstratios Gallopoulos‡

Abstract

We consider the computation of $x := \exp(-A)b$, where b is a vector and A is a matrix of order N, possibly nonsymmetric, by means of iterative methods. The algorithm is based on a transpose-free quasi-minimal residual algorithm to efficiently compute the solution of systems $(A - z_j I)x_j = b$ for several distinct values of the shift z_j, and on the partial-fraction representation of rational approximations of the matrix exponential. We outline mappings of the algorithm on a parallel architecture, present numerical experiments, and discuss the computational performance of the algorithm.

1 Problem Specification

The computation of

$$x := \exp(-A)b, \tag{1}$$

where b is a vector and A is a real matrix of order N, possibly nonsymmetric, is an important kernel operation in several application areas; see, e.g., [1, 2, 5, 14] and the references given therein. For large A it becomes prohibitive to compute $\exp(-A)$ directly (e.g., based on a spectral decomposition of A), and alternative methods have to be used.

One typical class of methods approximates x by

$$\hat{x} := R(A)b, \quad \text{where} \quad R(z) := \frac{q(z)}{p(z)} \quad \text{with polynomials} \quad p,\ q. \tag{2}$$

Here, the function $R(z)$ is a rational approximation to $\exp(-z)$ in a region of the complex plane enclosing the spectrum of A, and we always assume that $n := \deg p > 0$. The computation of the vector $\hat{x}$ is performed using the product form $p(z) := \gamma_0 \prod_{j=1}^{n}(z - z_j)$ of the denominator polynomial in (2). This has the advantage of better conditioning over the power-form representation. If we assume that the roots of p are all distinct and that $\deg p \geq \deg q$, then the sequential bottleneck that seems to arise because of the product-form representation is resolved by using the partial-fraction representation

$$\frac{q(z)}{p(z)} = \alpha_0 + \sum_{j=1}^{n} \frac{\alpha_j}{z - z_j}, \qquad \alpha_0 = \lim_{z \to \infty} \frac{q(z)}{p(z)}, \qquad \alpha_j = \frac{q(z_j)}{p'(z_j)}, \quad j = 1, 2, \ldots, n. \tag{3}$$

*Center for Computational Sciences & Engineering, Lawrence Livermore National Laboratory, P.O. Box 808, L–316, Livermore, CA 94551 (baldwin@xanthus.llnl.gov). This work was conducted while the author was with CSRD, University of Illinois at Urbana-Champaign, and it was supported in part by the U.S. National Science Foundation grant No. NSF CCR-91-20105.

†AT&T Bell Laboratories, Room 2C–420, 600 Mountain Avenue, Murray Hill, NJ 07974–0636 (freund@research.att.com).

‡CSRD, University of Illinois at Urbana-Champaign, 465 CSRL, 1308 West Main Street, Urbana, IL 61801 (stratis@csrd.uiuc.edu). Supported in part by the U.S. National Science Foundation grant No. NSF CCR-91-20105.

The formula (3) gives rise to the following Algorithm PF (**P**artial **F**raction) for computing an approximation $\hat{x}$ to the sought vector x; see [1, 9].

ALGORITHM PF (Computation of approximation $\hat{x}$ to (1) using formulas (2) and (3)).

1) Compute

$$\alpha_0 = \lim_{z \to \infty} \frac{q(z)}{p(z)} \qquad \text{and} \qquad \alpha_j = \frac{q(z_j)}{/} p'(z_j), \quad j = 1, 2, \ldots, n;$$

2) For $j = 1, 2, \ldots, n$, compute x_j by solving

$$(A - z_j I) x_j = b; \tag{4}$$

3) Set

$$\hat{x} = \alpha_0 b + \sum_{j=1}^{n} \alpha_j x_j.$$

Of course, in Algorithm PF, we still need to specify the rational approximation R, as well as the technique used to solve the linear systems (4) in step 2). Possible choices for R include rational Chebyshev approximations to $\exp(-z)$ on some compact set in the complex plane, and Padé approximations to $\exp(-z)$ about the origin $z = 0$; see, e.g., [2, 14]. In this paper, we will consider only Padé approximations. Recall that, by their very definition, Padé approximants provide a very good approximation in the neighborhood of the origin, while the approximation worsens as $|z|$ increases. Therefore, to obtain an acceptable approximation $\hat{x}$ to x we have to scale A by some factor $\delta < 1$ that depends on the spectrum of the matrix A and on the Padé approximant R, and then apply the algorithm for the computation of $\exp(-\delta A)$ times a vector repeatedly. For problems arising from parabolic partial differential equations, the scaling represents a time-step constraint, and the application of the exponential amounts to time-stepping. Clearly, the larger δ is, the fewer steps need to be taken to compute $\hat{x}$. The use of higher-order approximations is motivated by the desire to increase δ without sacrificing accuracy. Experiments in [1, 9] show that using higher-order approximations can be very effective on parallel architectures.

We now turn to the solution of the linear systems (4) in step 2). Note that this is the computationally dominating part of Algorithm PF. First, we observe that step 2) involves the independent solution of n linear systems. Hence Algorithm PF offers high-level parallelism and can be mapped effectively on parallel architectures. In addition, there is the parallelism originating from the solution of each linear system. If the order N of the linear systems (4) is large enough, then it becomes attractive to employ iterative methods. In this paper, we use a particular Krylov-subspace iteration, namely the transpose-free quasi-minimal residual (TFQMR) method [6], for the solution of the systems (4) in step 2) of Algorithm PF.

2 TFQMR and Exponential Propagation

Krylov-subspace iterations generate approximations to the solution x_j of the linear system (4) using the subspaces $K_m(A - z_j I, b)$, and possibly $K_m(A^T - z_j I, b)$. Here, $m = 1, 2, \ldots$ is the iteration index, and the notation $K_m(M, b) := \text{span}\{b, Mb, \ldots, M^{m-1}b\}$ is used for the mth Krylov subspace generated by the matrix M and the vector b. The quasi-minimal residual (QMR) method and its transpose-free variant TFQMR are Krylov-subspace iterations based on the Lanczos process and a squared Lanczos process, respectively. For a recent survey of Krylov-subspace iterations, we refer the reader to [7].

Note that the coefficient matrices of the n linear systems (4) differ only in their shifts z_j, and that Krylov subspaces are invariant under these shifts, i.e., $K_m(A - z_jI, b) = K_m(A, b)$ for all $j = 1, 2, \ldots, n$. It is well known that this property can be exploited when systems (4) are solved by Krylov-subspace iterations, and in fact, the n systems (4) can be solved with essentially the same sequential work that is required to solve a single system, plus a few extra vector updates For the case of Krylov-subspace iterations based on the Arnoldi process, this was observed in [3, 10]. In [5], it was shown that the special structure can also be exploited when the Lanczos-based QMR and TFQMR algorithms are used for the solution of multiply shifted systems of the type (4). In particular, detailed descriptions of modifications of QMR and TFQMR for multiply shifted systems were presented in [5]. Here, we will use the resulting multiply shifted variant of TFQMR [5, Algorithm 4,1], and in the sequel, we refer to this algorithm as MSTFQMR.

Each iteration of MSTFQMR consists of two types of calculations: computations that build common Krylov information to be shared among all n systems (4), and "private" computations that build or update data specific to each of the n systems. More precisely, computations of the first type are used to advance the dimension of the underlying Krylov subspaces $K_m(A - z_jI, b) = K_m(A, b)$ by 2 at each iteration step. These computations are independent of the number n of systems, and they consist of 2 matrix-vector multiplications $A \cdot v$, 4 dot products, and 6 vector updates. Computation of the second type require 9 vector updates for each of the n systems, plus updates of a few scalar quantities. Since TFQMR involves only short recurrences (in contrast to Arnoldi-based Krylov-subspace iterations), the storage requirements are independent of the iteration index m. More precisely, the MFTFQMR algorithm requires storage for approximately $5 \cdot n$ complex and 6 real vectors of length N. Recall that $n = \deg p$ is the degree of the denominator polynomial of the Padé approximant R.

Finally, we note that MSTFQMR is stopped as soon as sufficiently accurate approximations to all the n linear systems (4) have been obtained. Then it remains to perform step 3) of Algorithm PF. This computation is done by means of a BLAS2-type operation $\bar{x} \leftarrow Xa + \alpha_0 b$, where X is a complex dense matrix of dimension $N \times \deg p$.

One feature of the approach we described for the Algorithm PF is the high-level parallelism and its combination of shared and private computations to produce several partial vector results that are then reduced into one; cf. the comments in [13]. Shared computations originate from our desire to exploit the multiply shifted structure of the systems. We could, instead, dispense of the shared computations altogether and solve for each shift without any exchange of information between them. In order to accommodate these options, we rewrite (3) in the form

$$\frac{q(z)}{p(z)} = \alpha_0 + \sum_{l=1}^{k} \sum_{j \in I_l} \frac{\alpha_j}{z - z_j}. \tag{5}$$

Here, the collection of index sets $\{I_l\}_{l=1,2,\ldots,k}$ is a partition of $\{1, 2, \ldots, \deg p\}$. A modified version of Algorithm PF can then be written that treats the k components of the outer sum in (5) as independent systems, and solves them without exchanges of Krylov information, while it uses MSTFQMR to compute the components in each inner sum in (5). Thus when $k = \deg p$, the systems are solved as k independent processes; the resulting method was used in [8]. When $k = 1$ all systems are solved by a single instance of MSTFQMR. We also note that when A is symmetric and the shift ζ is complex, one can exploit the shifted structure to obtain economical CG-like methods [4] to solve each system $(A - \zeta I)x = b$.

3 Parallel Architectures with Hierarchical Resources

Some of the most interesting parallel architectures to date support some form of hierarchy in their processing and memory systems. For example, the CM-5 offers vector capabilities within each processor and parallelism across the network. One interesting recent example is the new Convex architecture. Two characteristics of the hierarchical organization are that access time increases according to the level of the memory hierarchy, and that there is support for a range of parallel task granularities.

The computer prototype that will be used for our experiments is the Cedar system at the Center for Supercomputing Research and Development of the University of Illinois. Cedar is a multicluster-based architecture (see Figure 1) with four clusters, where each cluster is a modified Alliant FX/8 machine with 8 computational elements (CEs). Three levels of parallelism can be applied. First, vectorization can be used within a CE. Second, small-grain parallelism can be exploited by using concurrency within the cluster. Finally, medium- and large-grain parallelism can be used across the clusters. There are four levels of memory-system hierarchy, namely registers for each CE, cache for each cluster, cluster memory, and global memory. As expected, the cost of access increases at each level. The relative performance degradation of the memory hierarchy is a factor of 2 from cache to cluster memory, a factor of 6 from cache to global memory, and a factor of 2 from cache to global memory, when a data-prefetching system is used. Each CE has its own set of scalar and vector registers. The CEs in each cluster share the cache and cluster memory, while the global memory is shared by the CEs of all clusters. Processors are connected to the global memory via a global multistage interconnection network. Cedar runs the Xylem operating system. Our programs were written in Cedar Fortran, a language with constructs similar to those of Fortran 90 and extensions for memory allocation, concurrency control, multitasking, and synchronization. It allows the specification of the location and visibility of the data (private, shared, cluster, or global). Concurrent execution of loops within a single cluster or across clusters are provided with CDOALL and SDOALL constructs, while parallel processing is also allowed by means of cluster tasks; see [11, 12] for detailed information.

Recall that k denotes the number of index sets in the representation (5). Setting k equal to the number of processor clusters, the modified PF algorithm outlined above lends itself to mapping on a hierarchical system such as Cedar. Each inner sum in (5) is mapped as a high-granularity task on each cluster, therefore k instances of MSTFQMR need to be invoked. We tried several data partitioning and placement strategies. The matrix A was stored in compressed-row sparse format. Under the first strategy, a copy of A was placed on each cluster's memory (**R**eplicated **C**luster organization). In the second strategy, we keep one copy of A in global memory, however each cluster performs matrix-vector multiplications using the entire matrix (**R**eplicated **G**lobal organization). The third strategy partitions the rows of A into k groups of approximately equal size and distributes the groups across the clusters (**B**locked **C**luster organization). The **B**locked **G**lobal strategy does a similar partition but keeps all groups (and hence A) in global memory. SDOALL loops were used to distribute the work across the clusters. In the blocked cases, the clusters collaborate to achieve the matrix-vector multiplications at the cost of greater synchronization per iteration. Instead, the replicated strategies require explicit synchronization in the formation of the global sums. In all cases, "private" vector data is partitioned $\deg p/k$ columns at a time and is stored in cluster memory. Therefore, except in matrix-vector multiplications, vectors are stored in the corresponding cluster memory. Since vectors are not spread across clusters, the corresponding dot products and updates can proceed without synchronization.

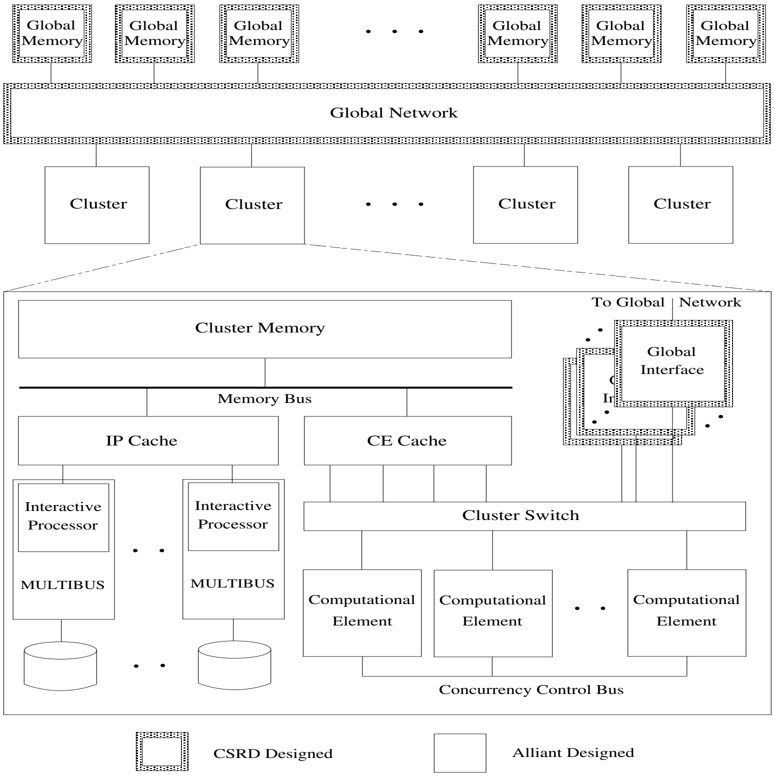

Fig. 1. *Cedar architecture.*

4 Numerical Experiments

We used the modified PF algorithm to compute (1) where A is a matrix resulting from a finite-difference discretization of a convection-diffusion operator on the unit square $[0,1] \times [0,1]$. A rectilinear finite-difference mesh is defined with N_x and N_y internal mesh points in the x and y directions, respectively. The corresponding mesh widths are $\Delta x = 1/(N_x+1)$ and $\Delta y = 1/(N_y+1)$, for a total of $N = N_x N_y$ points. The convection-diffusion operator is discretized by second-order centered finite differences, and then multiplied by a scaling factor δ. This results in a block tridiagonal matrix A of the form $A = T_x \otimes I_y + I_x \otimes T_y$. The matrices T_x and T_y are tridiagonal, with diagonal elements proportional to 2κ, and lower/upper off-diagonal elements proportional to $-\kappa \pm v_x \Delta_x/2$ and $-\kappa \pm v_y \Delta_y/2$, respectively. Here, κ is the diffusion coefficient, and v_x and v_y are the velocity components. As rational function R, we used diagonal (i.e., $\deg p = \deg q$) Padé approximations to $\exp(-z)$ about $z = 0$. Finally, the TFQMR iteration for each of the systems (4) was stopped as soon as the Euclidean norm of the initial residual was reduced by a factor of 10^{-7}. In Table 1, we list runtimes for different cluster sizes, date partitioning and placement strategies, and Padé approximations of order $\deg p = \deg q = 1, 4, 8, 12$.

We first observe that even though runtimes increase with $\deg p$ as the number of clusters stays fixed this increase is sublinear in the number of systems that are solved. On a single cluster, the application of the modified PF algorithm when $\deg p = 12$ is less than two times slower than the same algorithm with $\deg p = 1$. This shows the advantage of using the algorithm within each cluster. The replicated versions of the code appear to be superior, in general, to the blocked ones. This is partly due to the additional synchronization necessary to implement the distributed matrix-vector multiplications and despite the savings due to

TABLE 1
Runtimes in seconds for one step on Cedar for $N_x = N_y = 25$, $\kappa = 1.0$, $v_x = v_y = 1.0$.

		deg p			
partitioning	placement	1	4	8	12
One Cedar cluster (8 processors)					
R	C	0.96	1.18	1.52	1.83
R	G	0.94	1.18	1.53	1.86
B	C	0.98	1.19	1.57	1.89
B	G	1.01	1.23	1.60	1.90
Two Cedar clusters (16 processors)					
R	C	–	0.86	1.01	1.13
R	G	–	0.89	1.02	1.15
B	C	–	0.86	1.04	1.10
B	G	–	0.92	1.08	1.18
Four Cedar clusters (32 processors)					
R	C	–	0.75	0.81	0.86
R	G	–	0.73	0.84	0.86
B	C	–	0.76	0.84	0.88
B	G	–	0.74	0.91	0.92

the sharing of the work between the clusters. We also see that by increasing the number of clusters we obtain a reduction of the runtime of the higher-order methods. For instance, the runtimes when $\deg p = 1$ and one cluster is used are approximately the same as when $\deg p = 2$ and two clusters are used. This is expected, but we also observe that the times for four clusters become even smaller, e.g., the runtimes for $\deg p = 8$ on four clusters are smaller than the runtimes for $\deg p = 4$ on two clusters. They are also smaller than the runtimes for $\deg p = 1$ on one cluster. This behavior can be attributed to the additional bandwidth available as the number of clusters increases.

References

[1] D. Calvetti, E. Gallopoulos, and L. Reichel, *Incomplete partial fractions for parallel evaluation of rational matrix functions*, J. Comput. Appl. Math., to appear (1995).

[2] J. C. Cavendish, W. E. Culham, and R. S. Varga, *A comparison of Crank-Nicolson and Chebyshev rational methods for numerically solving linear parabolic equations*, J. Comput. Phys., 10 (1972), pp. 354–368.

[3] B. N. Datta and Y. Saad, *Arnoldi methods for large Sylvester-like observer matrix equations, and an associated algorithm for partial spectrum assignment*, Linear Algebra Appl., 154–156 (1991), pp. 225–244.

[4] R. W. Freund, *On conjugate gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices*, Numer. Math., 57 (1990), pp. 285–312.

[5] ———, *Solution of shifted linear systems by quasi-minimal residual iterations*, in Numerical Linear Algebra, L. Reichel, A. Ruttan, and R. Varga, eds., W. de Gruyter, Berlin, 1993, pp. 101–121.

[6] ———, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.

[7] R. W. Freund, G. H. Golub, and N. M. Nachtigal, *Iterative solution of linear systems*, Acta Numerica, 1 (1992), pp. 57–100.

[8] E. Gallopoulos and Y. Saad, *Efficient parallel solution of parabolic equations: implicit methods on the Cedar multicluster*, in Proc. Fourth SIAM Conf. Parallel Processing for Scientific Computing, J. Dongarra, P. Messina, D. C. Sorensen, and R. G. Voigt, eds., SIAM, Philadelphia, 1990, pp. 251–256.

[9] ———, *On the parallel solution of parabolic equations*, in Proc. 1989 ACM Internat. Conf. on Supercomputing, Herakleion, Greece, June 1989, pp. 17–28.

[10] C. W. Gear and Y. Saad, *Iterative solution of linear equations in ODE codes*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 583–601.

[11] J. Hoeflinger, *Cedar Fortran programmer's handbook*, Tech. Report 1157, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, October 1991.

[12] J. Konicek, T. Tilton, A. Veidenbaum, C. Zhu, E. Davidson, R. Downing, M. Haney, M. Sharma, P. Yew, P. Farmwald, D. Kuck, D. Lavery, R. Lindsey, D. Pointer, J. Andrews, T. Beck, T. Murphy, S. Turner, and N. Warter, *The organization of the Cedar system*, in Proc. 1991 Internat. Conf. on Parallel Processing, vol. I, St. Charles, Illinois, August 1991, pp. 49–56.

[13] D. M. Young and B. R. Vona, *On the use of rational iterative methods for solving large sparse linear systems*, Appl. Numer. Math., 10 (1992), pp. 261–278.

[14] R. S. Varga, *On higher order stable implicit methods for solving parabolic partial differential equations*, J. Math. Phys., 40 (1961), pp. 220–231.