Parallel Matrix Sign Iterations [*]

B. Bakkaloğlu and Ç. K. Koç
Department of Electrical & Computer Engineering
Oregon State University
Corvallis, Oregon 97331

Abstract

We have implemented three matrix sign function algorithms on a Meiko CS-2 multiprocessor using the Parallel Virtual Machine (PVM) software. These matrix sign function algorithms are the well-known Newton's method [3], the partial fraction expansion algoritm [12], and the continued fraction algorithm [9]. The parallelization of Newton's method is straightforward since it utilizes parallel LU decomposition. The parallel matrix sign function algorithm based on partial fraction expansion was given by Pandey, Kenney, and Laub in [12]. Our contribution in this paper was to introduce a new parallel iterative algorithm to compute the sign function of a complex matrix, based on the continued fraction expansion of the inverse of the principal square root function. This algorithm uses parallel matrix mutiplications and linear system solutions at each step, and is suitable for vector and parallel machines due to its multiplication-rich nature. We have written all three algorithms in PVM, obtaining a highly portable code which can be compiled and executed on a network of workstations as well as on sophisticated multiprocessor machines. We summarize the results of our experiments on an 8-processor Meiko CS-2.

1 Introduction

The matrix sign function has several applications in system theory, matrix analysis, and communications, e.g., solution of algebraic Riccati and matrix Lyapunov equations [10], system decomposition and model reduction [14, 11], separation of eigenpairs [4], condition theory [5], and recently the numerical solution of M/G/1 and G/M/1 type Markov chains [1]. The matrix sign function maps the stable and unstable eigenvalues of a given matrix to -1 and 1, respectively, while preserving the eigenvectors of the original matrix. This property of the matrix sign function is useful for studying the eigenstructures of matrices without explicitly computing the eigenvalues. The sign of a complex scalar λ is defined over $\mathrm{Re}(\lambda) \neq 0$ by

$$\mathrm{sign}(\lambda) = \begin{cases} 1 & \text{if } \mathrm{Re}(\lambda) > 0 \ , \\ -1 & \text{if } \mathrm{Re}(\lambda) < 0 \ . \end{cases}$$

This definition can be extended to a matrix $A \in \mathcal{C}^{n \times n}$ whose eigenvalues do not lie on the imaginary axis. Let M take A to its Jordan form J as

$$A = MJM^{-1}. \tag{1}$$

[*]This research was supported in part by the National Science Foundation under the grants ECS–9312240 and CDA–9216172.

Let J be defined as

$$J = \begin{bmatrix} J_+ & 0 \\ 0 & J_- \end{bmatrix} = J_+ \oplus J_-$$

where $J_+ \in \mathcal{C}^{n_1 \times n_1}$ and $J_- \in \mathcal{C}^{n_2 \times n_2}$ are the Jordan blocks with $\mathrm{Re}(\sigma(A)) > 0$ and $\mathrm{Re}(\sigma(A)) < 0$, respectively, and $n = n_1 + n_2$. Applying sign function to both sides of Equation (1) we obtain

$$\mathrm{sign}(A) = M \ \mathrm{sign}(J) \ M^{-1} \ .$$

The matrix sign of the Jordan blocks determine the sign of A as follows:

$$\mathrm{sign}(A) = M \begin{bmatrix} \mathrm{sign}(J_+) & 0 \\ 0 & \mathrm{sign}(J_-) \end{bmatrix} M^{-1} = M \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} M^{-1} \ . \tag{2}$$

Equation (2) shows that the Jordan blocks corresponding to positive (negative) eigenvalues are mapped to positive (negative) identity matrices, whose dimensions are the same as the number of positive (negative) eigenvalues. It follows that $S = \mathrm{sign}(A)$ is a diagonalizable matrix which commutes with A and is a square root of the identity, i.e.,

$$S^2 = I \ \text{ and } \quad AS = SA \ . \tag{3}$$

Equation (3) is a quadratic equation in S, and can be solved by Newton's method [13]. Another definition, which is based on integral representation, is given in [13]. It uses the integral formula

$$A^+ = \frac{1}{2\pi j} \int_C (zI - A)^{-1} dz \ ,$$

where C is a simple closed contour in $\mathcal{C}^+$, containing the eigenvalues of A with positive real part. Using the equality $A^+ = (\mathrm{sign}(A) + I)/2$, we obtain an integral expression for $\mathrm{sign}(A)$ as follows:

$$\mathrm{sign}(A) = \frac{2A}{\pi} \int_0^\infty (y^2 I + A^2)^{-1} dy \ .$$

The parallel computation of matrix sign function has recently received attention in order to deal with large matrices [3, 12]. In the following we give brief descriptions of the previously proposed parallel algorithms along with the new parallel algorithm. The analysis of these parallel algorithms are performed by counting the number of arithmetic operations and communication steps per iteration. The number of iterations needed for the convergence is a function of several factors, e.g., the size and condition of the matrix, certain properties of the algorithm, etc. The implementation results on an 8-processor Meiko CS-2 multiprocessor using the PVM software are summarized in Section 5.

2 The Parallel Newton Iteration

This algorithm is a slightly modified version of the symmetric pivoting algorithm of [3]. The original algorithm has been applied to solution of the algebraic Riccati equation, where the iteration is carried on a matrix pencil with Hamiltonian-like structure. The structure of the pencil has the property that multiplying with an anti-diagonal identity matrix would convert it to a symmetric matrix. In this paper we have implemented a modified version of this algorithm for finding the matrix sign function of a general nonsymmetric matrix. The algorithm is based on parallel factorization of the iteration matrix $S_k \in \mathcal{R}^{m\times m}$ at each step of the algorithm. We obtain the inverse of the symmetric matrix S_k in parallel and compute its determinant d_k in order to calculate the scaling factor γ_k. The algorithm starts with $S[0] = A$, and proceeds using the iteration

$$S[k+1] = \frac{1}{2}(\frac{1}{\gamma_k}S[k] + \gamma_k S^{-1}[k]) \ . \tag{4}$$

The determinantal scaling factor [7] is given as

$$\gamma_k = |\det S[k]|^{1/m} \ .$$

The inverse of the iteration matrix $S[k]$ is computed and the scaling factor γ_k is obtained during the parallel LU decomposition. The iteration matrix $S[k]$ is distributed among p processors in a column-wrapped fashion so that each processor works on an array of $n \times n/p$ elements. The parallel LU decomposition requires $O(n^3/p)$ arithmetic operations and $O(n)$ communication steps. The scaling and matrix addition requires $O(n^2/p)$ arithmetic steps. Finally, each processor sends its portion of the iteration matrix to the rest of the processors in order to calculate $S[k+1]$. This operation is called a multi-node broadcast operation. The details of the multi-node broadcast operation for various parallel architectures can be found in [2]. We denote the communication time of the multi-node broadcast operation of a single matrix element by B. Since n^2/p matrix elements are being broadcast, the update operation requires $O(n^2B/p)$ communication steps. Thus, a single Newton iteration requires $O(n^3/p)$ arithmetic operations with a communication penalty of $O(n^2B/p)$.

3 The Partial Fraction Expansion Algorithm

The most common method for finding sign(A) is Newton's method which is globally convergent for matrices with nonzero eigenvalues. This method can also be extended to globally convergent rational iterations of arbitrary order. Howland [4] derived a closed form formula for a desired degree rational iteration from the following error relation

$$\frac{s_{k+1}-1}{s_{k+1}+1} = \left(\frac{s_k-1}{s_k+1}\right)^p$$

from which we solve for s_{k+1} as

$$s_{k+1} = \frac{(s_k+1)^p + (s_k-1)^p}{(s_k+1)^p - (s_k-1)^p} \ .$$

This is the principal Padé iteration of order p [6]. Recently it is noted in [8] that this rational approximation can be represented as

$$s_{k+1} = \tanh(p \ \text{arctanh} \ s_k) \ ,$$

where

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \ .$$

It is also noted in [8] that the above function for even p can be expanded in partial fractions as

$$\tanh(p \operatorname{arctanh}(s)) = \frac{s}{p} \sum_{i=0}^{p-1} \frac{1}{\sin^2(\frac{(2i+1)\pi}{4n}) + \cos^2(\frac{(2i+1)\pi}{4n}) s^2} \tag{5}$$

Using the partial fraction expansion of (5), a parallel iteration [12] for the matrix sign function is obtained. The algorithm starts with $S[0] = A$, and uses the iteration formula

$$S[k+1] = S[k] \sum_{i=1}^{p} \frac{1}{p} (\alpha_i^2 I + \beta_i^2 S^2[k])^{-1} \ , \tag{6}$$

where p is the number of processors and

$$\alpha_i = \sin\left(\frac{(2i-1)\pi}{4n}\right) \quad \text{and} \quad \beta_i = \cos\left(\frac{(2i-1)\pi}{4n}\right) \ .$$

Each step of the parallel sign function iteration starts with parallel squaring of the iteration matrix $S[k]$ which requires $O(n^3/p)$ parallel arithmetic steps. After $S^2[k]$ is computed, it is distributed among the processors, and each processor sequentially computes $(\alpha_i^2 I + \beta_i^2 S^2[k])^{-1}$. This step requires $O(n^2/p)$ communication steps and $O(n^3)$ arithmetic operations. The summation of (6) is obtained using a binary tree which requires $O(n^2 \log p)$ communication and arithmetic steps. After the sum is obtained, it is multiplied by $S[k]$ in parallel to obtain $S[k+1]$. This step requires $O(n^3/p)$ arithmetic operations. Thus, a single iteration step requires approximately $O(n^3/p)$ arithmetic operations and $O(n^2 \log p)$ communication steps.

4 The Parallel Continued Fraction Algorithm

The proposed algorithm is based on parallelization of the continued fraction algorithm [9]. This algorithm employs the inverse square-root of a matrix using the continued fraction expansion. The iterative algorithm for computing the inverse of the principal square root of the complex matrix $A \in \mathcal{C}^{n \times n}$ is stated as follows:

$$\begin{aligned} \begin{bmatrix} P_1 \\ Q_1 \end{bmatrix} &= \begin{bmatrix} I \\ I \end{bmatrix} , \\ \begin{bmatrix} P_j \\ Q_j \end{bmatrix} &= \begin{bmatrix} I & I \\ A & I \end{bmatrix} \begin{bmatrix} P_{j-1} \\ Q_{j-1} \end{bmatrix} , \\ \lim_{j \to \infty} P_j Q_j^{-1} &= (\sqrt{A})^{-1} , \end{aligned} \tag{7}$$

where $P_j, Q_j \in \mathcal{C}^{n \times n}$. By replacing the block element A with A^2 in the iteration matrix of (7), we obtain an iterative algorithm for computing the inverse square root of the square of a complex matrix, which in turn can be used for the computation of the matrix sign function due to the following definition of the matrix sign function:

$$\operatorname{sign}(A) = A(\sqrt{A^2})^{-1} = A^{-1}(\sqrt{A^2}) \ .$$

The continued fraction based matrix sign function algorithm starts with $S[0] = A$, and uses the iteration:

$$S[k+1] = S[k] \left(\sqrt{S^2[k]}\right)^{-1} \ .$$

For each value of k, we compute an approximation for the inverse of the principal square root of $S^2[k]$. This is achieved by iterating the continued fraction algorithm r times for $j = 1, 2, \ldots, r$. The matrix sign function algorithm starts with $S[0] = A$ and iteratively computes the sign function of A by going through a series of baby-steps and giant-steps, corresponding to the computation of an r-step approximation for the inverse of the principal square root and the computation of the new value of S, respectively. Thus, the iteration for the inverse square root is modified for computing the matrix sign function as follows:

$$\begin{array}{lrcl}
\textsf{Start:} & S[0] & = & A \ , \\
\textsf{Baby-Step:} & P_1[k] & = & I \ , \\
 & Q_1[k] & = & I \ , \\
j = 2, 3, \ldots, r : & P_j[k] & = & P_{j-1}[k] + Q_{j-1}[k] \ , \\
 & Q_j[k] & = & S^2[k]P_{j-1}[k] + Q_{j-1}[k] \ , \\
\textsf{Giant-Step:} & S[k+1] & = & S[k]P_r[k]Q_r^{-1}[k] \ .
\end{array} \qquad (8)$$

A baby-step corresponds to the computation of $P_r[k]$ and $Q_r[k]$ using $S[k]$ and the initial values $P_1[k] = Q_1[k] = I$. There are no matrix inversions during this phase. The number of iterations during the baby-step phase is equal to r, which is predetermined. We start with $P_1[k] = I$ and $Q_1[k] = I$ and compute $P_j[k]$ and $Q_j[k]$ for $j = 2, 3, \ldots, r$, using the continued fraction based iterative algorithm for the inverse square root. A giant-step, on the other hand, corresponds to the computation of $S[k+1]$, using $S[k]$, $P_r[k]$, and $Q_r[k]$. There is a single matrix inversion during the giant-step. The number of giant-step iterations is a function of the convergence properties of the algorithm, the input matrix, as well as the baby-step length r.

- Distribute $P_{j-1}[k]$ and $Q_{j-1}[k]$ among processors in a column wrapped fashion.
- Compute $S^2[k]$ using parallel matrix multiply.
- Compute $S^2[k]P_{j-1}[k]$ using parallel matrix multiply.
- Add the corresponding columns of $P_{j-1}[k]$ and $Q_{j-1}[k]$ to obtain $P_j[k]$.
- Add $S^2[k]P_{j-1}[k]$ and $Q_{j-1}[k]$ in a similar fashion to obtain $Q_j[k]$.
- Solve in parallel for $S[k+1]$ in $Q_r[k]S[k+1] = S[k]P_r[k]$.

The proposed algorithm achieves data parallelism by distributing the matrices among the processors and obtaining the LU decomposition and matrix products in parallel. With $p < n$ processors, computing $S^2[k]$ and $S^2[k]P_{j-1}[k]$ and updating the new iterate on each processor requires $O(n^3/p)$ arithmetic steps and $O(n^2B/p)$ communication steps, where B is the communication overhead of a multi-node broadcast operation. Adding the corresponding rectangular arrays $P_{j-1}[k]$, $S^2[k]P_{j-1}[k]$ and $Q_{j-1}[k]$ to obtain $P_j[k]$ and $Q_j[k]$ requires n^2/p arithmetic steps and $O(n^2B/p)$ communication steps. Finally parallel solution of the linear system requires $O(n^3/p)$ parallel arithmetic operations. Therefore each giant-step requires approximately $O(rn^3/p)$ arithmetic steps and $O(rn^2B/p)$ communication steps where r is the baby-step length.

5 Implementation Results and Conclusions

We have implemented these three parallel matrix sign function algorithms on an 8-processor partition of a Meiko CS-2 multiprocessor, in which each node is a Sparc processor equipped with 256

MBytes of memory. The algorithms are implemented using the PVM software. In our experiments, we have computed the sign functions of matrices of dimensions ranging from 128 to 1024. The matrices are generated randomly with geometrically distributed eigenvalues. In Table 1, we give the parallel times for the three algorithms as a function of time for $p = 2, 4$, and 8.

Table 1: The parallel times for the algorithms (in seconds).

	$p=2$			$p=4$			$p=8$		
Size	N	PFE	CF	N	PFE	CF	N	PFE	CF
128	0.221	0.070	0.052	0.184	0.058	0.043	0.153	0.047	0.037
256	0.504	0.186	0.161	0.387	0.143	0.124	0.297	0.109	0.094
384	0.985	0.422	0.312	0.703	0.301	0.230	0.502	0.212	0.168
512	1.463	0.700	0.601	0.975	0.464	0.401	0.650	0.314	0.273
640	2.242	1.192	1.024	1.401	0.745	0.640	0.875	0.481	0.407
768	3.924	2.086	1.792	2.451	1.301	1.120	1.529	0.843	0.712
896	7.065	3.754	3.224	4.420	2.340	2.017	2.750	1.517	1.288
1024	12.993	6.941	5.967	8.146	4.325	3.729	5.100	2.807	2.379

The number of iteration steps for the the partial fraction algorithm is a function of the number of processors (the order of the summation) and the size of the matrix. For the continued fraction algorithm the number of the giant-steps depends on r. A thorough analysis of the dependency of the number of giant-steps on r is given in [9]. For this implementation we selected r to be 4. In Table 2, we tabulate the number of iterations for the algorithms as a function of the matrix size and the number processors.

Table 2: The number of iterations for the algorithms.

	$p=2$			$p=4$			$p=8$		
Size	N	PFE	CF	N	PFE	CF	N	PFE	CF
128	13	9	8	13	6	8	13	5	8
256	13	9	7	13	6	7	13	5	7
384	13	9	7	13	6	7	13	5	7
512	14	9	7	14	6	7	14	5	7
640	14	8	7	14	6	7	14	4	7
768	14	8	7	14	5	7	14	4	7
896	14	8	7	14	5	7	14	4	7
1024	15	7	7	15	5	7	15	4	7

Our implementation results show that Newton's method is the slowest of all three algorithms, mainly because of the high number of iterations. For example, for matrix size 1024, Newton's method requires nearly twice the time required by the partial fraction expansion algorithm. Furthermore, comparing the other two algorithms to one another, we conclude that the continued fraction algorithm is slightly faster than the partial fraction expansion algorithm. Although the partial fraction algorithm requires the fewest number of iterations for $p = 4$ or 8, its total time is slightly larger than the continued fraction algorithm, This is mainly due to the fact that the partial fraction expansion algorithm requires the summation of n^3 matrix elements distributed over p processors, introducing a communication penalty of $O(n^3 \log p)$ at each step which considerably lengthens the total time.

References

[1] N. Akar and K. Sohraby. An invariant subspace approach in M/G/1 and G/M/1 type Markov chains. Manuscript, University of Missouri – Kansas City, March 1995.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[3] J. D. Gardiner and A. J. Laub. Parallel algorithms for algebraic Riccati equations. *International Journal of Control*, 44(6):1317–1333, December 1991.

[4] J. Howland. The sign matrix and the separation of eigenvalues. *Linear Algebra and its Applications*, 49:221–232, 1983.

[5] C. Kenney and A. J. Laub. Polar decomposition and matrix sign function condition estimates. *SIAM Journal on Scientific and Statistical Computing*, 12(3):488–504, 1991.

[6] C. Kenney and A. J. Laub. Rational iterative methods for the matrix sign function. *SIAM Journal on Matrix Analysis and Applications*, 12(2):273–291, April 1991.

[7] C. Kenney and A. J. Laub. On scaling Newton's method for polar decomposition and the matrix sign function. *SIAM Journal on Matrix Analysis and Applications*, 13(3):688–706, July 1992.

[8] C. Kenney and A. J. Laub. A hyperbolic tangent identity and geometry of Padé sign function iterations. *Numerical Algorithms*, 7:111–128, 1994.

[9] Ç. K. Koç, B. Bakkaloğlu, and L. S. Shieh. Computation of the matrix sign function using continued fraction expansion. *IEEE Transactions on Automatic Control*, 39(8):1644–1647, August 1994.

[10] A. J. Laub. Invariant subspace methods for the numerical solution of Riccati equations. In S. Bittanti, A. J. Laub, and J. C. Willems, editors, *The Riccati Equation*, pages 163–196. New York, NY: Springer-Verlag, 1991.

[11] R. L. Mattheys. Stability analysis via the extended matrix sign function. *IEE Proceedings: Control Theory and Applications*, 125(3):1057–1078, 1978.

[12] P. Pandey, C. Kenney, and A. J. Laub. A parallel algorithm for the matrix sign function. *International Journal of High-Speed Computing*, 2(2):181–191, 1990.

[13] J. D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *International Journal of Control*, 32(4):677–687, 1980.

[14] L. S. Shieh, H. M. Dib, and R. E. Yates. Separation of matrix eigenvalues and structural decomposition of large-scale systems. *IEE Proceedings: Control Theory and Applications*, 133(2):90–96, 1986.