# A Parallel Algorithm for Principal $n$th Roots of Matrices *

Ç. K. Koç    and    M. İnceoğlu

## Abstract

An iterative algorithm for computing the principal $n$th root of a positive definite matrix is presented. The algorithm is based on the Gauss-Legendre approximation of a definite integral. We present a parallelization in which we use as many processors as the order of the approximation. An analysis of the error introduced at each step of the iteration indicates that the algorithm converges more rapidly as the order of the approximation (thus, the number of processors) increases. We describe the results of our implementation on an 8-processor Meiko CS-2, comparing the parallel algorithm to the fastest sequential algorithm, which is the Hoskins-Walton method.

## Key Words

Matrix methods, Iterative methods, Parallel algorithms.

## Authors' Affiliations

Ç. K. Koç   — Author to whom all correspondence should be sent
Department of Electrical & Computer Engineering
Oregon State University
Corvallis, OR 97331, USA


M. İnceoğlu
Department of Computer Engineering
Ege University
Izmir, Turkey

# 1  Introduction

Several methods for finding the $n$th roots of matrices have been developed [Hoskins and Walton, 1979] [Denman, 1981] [Bjork and Hammarling, 1983] [Higham, 1986] [Higham, 1987] [Tsai *et al*, 1988]. A fast and stable method for computing the square-root and cube-root of a matrix, based on the Schur factorization $A = QSQ^H$ and a fast recursion, is described in [Bjork and Hammarling, 1983]. An extension of this algorithm to compute the real square-root of a real matrix in given in [Higham, 1987]. An accelerated iterative method for computing the $n$th root of a positive definite matrix is introduced in [Hoskins and Walton, 1979]. This method is later extended for general real matrices [Denman, 1981]. Furthermore, methods for computing the principal $n$th roots of complex matrices have also been developed [Shieh *et al*, 1986] [Tsai *et al*, 1988].

The computation of the $n$th roots of matrices has several applications in control system design and analysis. For example, the matrix root functions and the associated matrix sign and sector functions are used to solve the matrix Lyapunov and Riccati equations in stability and optimal control problems [Gardiner and Laub, 1986] [Shieh *et al*, 1987] [Shieh *et al*, 1990]. Other related applications can be found in [Shieh *et al*, 1987].

In this paper, we describe a new parallel iterative algorithm to compute the principal $n$th root of a positive definite matrix, without explicitly computing its eigenvalues. The algorithm, based on the Gauss-Legendre approximation of a definite integral, is similar in flavor to the approach used in [Pandey *et al*, 1990] to approximate the matrix sign function. We derive the summation expression for the iterative $n$th root algorithm in Section 2, and analyze the error properties of this iteration in Section 3. Finally, we summarize the results of our implementation on an 8-processor Meiko CS-2, and compare the speed of the new algorithm to that of the Hoskins-Walton algorithm [Hoskins and Walton, 1979], which seems to be the fastest sequential algorithm.

# 2  The Derivation of the Iteration

The principal $n$th root of a positive definite matrix can be defined in terms of its eigenvalue decomposition. Let $A \in \mathcal{R}^{q \times q}$ be a positive definite matrix and $\sigma(A) = \{\lambda_i ,\ i = 1, \ldots, q\}$ be its spectrum with eigenvalues $\lambda_i \neq 0$ and $\arg(\lambda_i) \neq \pi$. Let $M \in \mathcal{C}^{q \times q}$ be the modal matrix that takes $A$ to its Jordan form as

$$A = M \ \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_q) \ M^{-1} \ .$$

Applying the matrix function definition of Giorgi [Rinehart, 1955], the principal $n$th root of a complex matrix can be defined as

$$\sqrt[n]{A} = M \ \mathrm{diag}(\sqrt[n]{\lambda_1}, \sqrt[n]{\lambda_2}, \ldots, \sqrt[n]{\lambda_q}) \ M^{-1},$$

where $n$ is a positive integer, and $\arg(\sqrt[n]{\lambda_i}) \in (-\pi/n, \pi/n)$.

In order to derive the algorithm, we express the $n$th root function in terms of an hypergeometric function, and then by expanding this hypergeometric function in power series, we find a definite integral giving the $n$th root function. We follow the method described in [Pandey *et al*, 1990] [Kenney and Laub, 1991], and use the hypergeometric function

$$_2F_1(\alpha, \beta, \gamma, z) = \sum_{i=0}^{\infty} \frac{(\alpha)_i (\beta)_i}{i!(\gamma)_i} \ z^i \ ,$$

where $\alpha, \gamma, z \in \mathcal{R}$, and

$$(\alpha)_i = \alpha(\alpha + 1) \cdots (\alpha + i - 1)$$

with $(\alpha)_0 = 1$. More specifically, in order to approximate the $n$th root, we use the hypergeometric function

$$\frac{1}{\sqrt[n]{1-z}} = {}_2F_1(1/n, 1, 1, z) = \sum_{i=0}^{\infty} \frac{(1/n)_i}{i!} \, z^i \qquad (1)$$

The coefficients in the power series expansion are calculated using the following integral

$$\frac{(\alpha)_i}{(\gamma)_i} = \int_0^1 x^i w(\alpha, \gamma, x) dx \ ,$$

where the weight function $w(\alpha, \gamma, x)$ is given in [Kenney and Laub, 1989]. For this particular case, the weight function is

$$w(1/n, 1, x) = \frac{x^{(1/n)-1}(1-x)^{-1/n}}{\pi \csc(\pi/n)} \ .$$

Substituting $w(\alpha, \gamma, x)$ in the power series expansion of (1), we obtain

$$\begin{aligned}
\frac{1}{\sqrt[n]{1-z}} &= \sum_{i=0}^{\infty} \left( \int_0^1 x^i w(1/n, 1, x) dx \right) z^i \\
&= \frac{1}{\pi \csc(\pi/n)} \int_0^1 \frac{x^{(1/n)-1}}{(1-x)^{1/n}(1-xz)} dx \ .
\end{aligned} \qquad (2)$$

We derive the parallel $n$th root algorithm using the Gauss-Legendre quadrature approximation for the integral formula (2). To apply the Gauss-Legendre quadrature formula, we shift the interval of integration to $(-1, 1)$. By a change of variable as $x = (\tilde{x} + 1)/2$, we obtain the integral as

$$\frac{1}{\sqrt[n]{1-z}} = \frac{1}{\pi \csc(\pi/n)} \int_{-1}^1 \frac{h(\tilde{x})}{(2 - (\tilde{x} + 1)z)} d\tilde{x} \ ,$$

where

$$h(\tilde{x}) = \frac{2(1 + \tilde{x})^{(1/n)}}{(1 + \tilde{x})(1 - \tilde{x})^{1/n}} \ .$$

This integral can be approximated [Stroud and Secrest, 1966] using the following summation

$$\int_{-1}^1 \frac{h(\tilde{x}) \, d\tilde{x}}{(2 - (\tilde{x} + 1)z)} \approx \sum_{i=1}^m \frac{w_i h(x_i)}{(2 - (x_i + 1)z)} \ , \qquad (3)$$

where the nodes $\{x_i\}$ are the zeroes of the degree $m$ Legendre polynomial $P_m(x)$ on $(-1, 1)$, and the weights $w_i$ for $i = 1, 2, \ldots, m$ are given as

$$w_i = \frac{-2}{(m + 1)P_m'(x_i)P_{m+1}(x_i)}$$

## 3   The Parallel Iteration

The parallel iteration uses the summation formula (3) in order to compute the $n$th root. The iteration starts with $s_0 = 1$, and proceeds with $s_{k+1} = s_k / \sqrt[n]{s_k^n/\lambda}$ so that $\lim_{k \to \infty} s_k = \sqrt[n]{\lambda}$. By substituting $z_k = 1 - s_k^n/\lambda$ in the quadrature formula, we introduce an auxiliary variable $z_k$ as

$$\begin{aligned}
z_k &= 1 - \frac{s_k^n}{\lambda} \ , \\
s_{k+1} &= \frac{s_k}{\pi \csc(\pi/n)} \sum_{i=1}^m \frac{w_i h(x_i)}{(2 - (x_i + 1)z_k)} \ ,
\end{aligned}$$

In the matrix case, the iteration starts with $S_0 = I$, and proceeds with

$$Z_k = I - A^{-1} S_k^n \; ,$$

$$S_{k+1} = \frac{1}{\pi \csc(\pi/n)} \; S_k \sum_{i=1}^{m} w_i h(x_i)(2I - (x_i + 1)Z_k)^{-1} \; .$$

Since $\lim_{k\to\infty} S_k = \sqrt[n]{A}$, the auxiliary iteration matrix $Z_k$ satisfies $\lim_{k\to\infty} Z_k = 0_q$, where $0_q$ is a $q \times q$ zero matrix. Therefore, the convergence can be monitored by checking $\|Z_k\|$ at each step, and terminating the iteration when $\|Z_k\| < \epsilon$.

The propsed iteration is suitable for parallelism in a natural way. Assuming we have $m$ processors available, at each step of the iteration, each processor computes one part of the above summation, and broadcasts this result among all processors. These results are then summed by all processors using the associative binary tree algorithm. The details of the parallel algorithm are given in Table 1.

$$\boxed{\text{Table 1 goes here}}$$

The parallel algorithm assumes that the order of approximation $m$ and the actual number of processors $P$ is equal. In practice, this need not be the case. In order to achieve a reasonable convergence rate, it may be necessary to use high order approximation even if there are fewer processors. If $P < m$, then the structure of the proposed parallel algorithm changes only slightly: A processor computes more than one matrix $T_i$. If $P > m$, then more than one processor is involved in the computation of a matrix $T_i$. The computation of $\Sigma$, $S_{k+1}$, and $Z_{k+1}$ are performed by all processors in both cases. Another factor in the parallelization of the proposed algorithm is the value of $n$. When $n$ is large (e.g., $n > 100$), then the computation of $S_{k+1}^n$ in the last step will take significant amount of time if it is not performed in parallel. We have considered only small values of $n$ in our numerical experiments (parallel or sequential). However, for larger values of $n$, this step needs to be parallelized as well.

We now give a brief analysis of the proposed algorithm in terms of the number of arithmetic operations required in a single step of the iteration. This analysis is further helpful in determining the amount of parallelization required for different values of $m$, $n$, and the matrix size $q$. In this analysis, it is assumed that the number of processors $P$ is less than the order of approximation $m$. We also assume that all scalar approximation parameters, e.g., the zeroes $\{x_i\}$ of the Legendre polynomial, the weights $w_i$, etc., are precomputed and readily available. We calculate only the number of arithmetic operations required by the matrix operations: scalar-matrix multiply, matrix add and subtract, matrix multiply, matrix inversion, and matrix powering. The number of arithmetic operations required for these matrix operations are well-known, for example, see [Golub and van Loan, 1989].

$$\boxed{\text{Table 2 goes here}}$$

Using the values tabulated in Table 2, we calculate the number of arithmetic operations required by a single iteration of the algorithm below:

- Computation of a single $T_i$ requires $\frac{8}{3}q^3 + O(q^2)$ arithmetic operations by one processor. All $T_i$ from 1 to $m$ are computed using $\frac{8m}{3P}q^3$ arithmetic operations with $P < m$ processors, ignoring the lower order terms.

- Computation of $\Sigma$ requires $O(mq^2)$ arithmetic operations with one processor, and $O((\frac{m}{P} + \log_2 P)q^2)$ arithmetic operations with $P$ processors.

- Computation of $S_{k+1}$ requires $2q^3 + O(q^2)$ arithmetic operations by a single processor. Assuming $P < q$, this can be accomplished using $\frac{2q^3}{P}$ arithmetic operations with $P$ processors, ignoring the lower order terms.

- Computation $S_{k+1}^n$ can accomplished by performing parallel matrix multiply operations using $P$ processors. This requires $(1.5 \log_2 n)(2q^3 - q^2)/P$ arithmetic operations. We then compute $Z_{k+1}$ using an additional $\frac{2q^3}{P}$ arithmetic operations.

Keeping only the higher order terms (i.e., terms containing $q^3$), we calculate the total number of arithmetic operations required in a single iteration step as

$$\frac{8mq^3}{3P} + \frac{2q^3}{P} + \frac{(2q^3)(1.5 \log_2 n)}{P} + \frac{2q^3}{P} = \left(\frac{8m}{3} + 3 \log_2 n + 4\right) \frac{q^3}{P} \ .$$

This analysis implies that the proposed parallel algorithm achieves nearly linear speedup per iteration step. The experiments performed on an 8-processor Meiko CS-2 are described in the last section.

# 4  Error Analysis

We analyze the errors in the quadrature approximation and the matrix iteration separately. The Gauss-Legendre integration including the error term can be given as [Davis and Rabinowitz, 1975]

$$\int_{-1}^{1} g(x)dx = \sum_{i=1}^{m} w_i g(x_i) + E_m(g) \ ,$$

where

$$E_m(g) = \frac{2^{2m+1}(m!)^4}{(2m + 1)((2m)!)^2} \frac{g^{(2m)}(t)}{(2m)!} \ . \tag{4}$$

This error term can be simplified by defining

$$A_m = \frac{2^{2m+1}(m!)^4}{(2m + 1)((2m)!)^2}$$

$$B_m = \max_{-1 < t < 1} \frac{|g^{(m)}(t)|}{m!} \ ,$$

which gives the error term (4) as

$$|E_m(g)| \leq A_m B_{2m} \ .$$

We use Stirling's formula $m! \approx e^{-m} m^m \sqrt{2\pi m}$ and obtain $A_m \approx \pi 4^{-m}$. This gives the error formula as

$$|E_m(g)| \leq \pi 4^{-m} B_{2m} \ . \tag{5}$$

In the scalar case of the definite integral for the matrix $n$th root, we have

$$f(x, z) = \frac{2(1 + x)^{(1/n)-1}}{(1 - x)^{(1/n)}(2 - (x + 1)z)} \ .$$

This integrand has an apparent singularity at the upper limit of the integration. However, this endpoint singularity can be ignored to approximate the error term [Davis and Rabinowitz, 1975]. Thus, the error formula (5) is applicable to the above integrand as well.

We now formulate error expression for the matrix iteration. Let $M \in \mathcal{R}^{q \times q}$ take the symmetric positive definite matrix $S_k$ to its diagonal form $D_k$ as

$$D_k = M^{-1} S_k M \ ,$$

where $D_k = \text{diag}(d_1[k], d_2[k], \ldots, d_q[k])$. In a similar fashion, $Z_k$ can be decomposed to its diagonal form using the same modal matrix $M$ as

$$Z_k = M(I - \Lambda^{-1} D_k^n)M^{-1} \ ,$$

where $\Lambda = M^{-1}AM$, $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_q)$, and

$$I - \Lambda^{-1}D_k^n = \text{diag}\left(1 - \frac{d_1^n[k]}{\lambda_1}, 1 - \frac{d_2^n[k]}{\lambda_2}, \ldots, 1 - \frac{d_q^n[k]}{\lambda_q}\right) .$$

Decomposing the matrix iteration with the same modal matrix $M$, we obtain the matrix iteration on the singular values of $S_k$ as

$$\begin{aligned} d_j[k+1] &= \frac{d_j[k]}{\pi \csc(\pi/n)} \sum_{i=1}^{m} \frac{w_i h(x_i)}{2 - (x_i + 1)(1 - d_j^n[k]/\lambda_j)} \\ &= \sqrt[n]{\lambda_j} + \mathcal{E}_{j,m}[k] \end{aligned}$$

for $j = 1, 2, \ldots, q$. The analysis of the scalar case, as given by the error formula (5), suggests that

$$|\mathcal{E}_{j,m}[k]| \leq \pi 4^{-m} B_{j,2m}[k] \tag{6}$$

for $j = 1, 2, \ldots, q$, where

$$B_{j,2m}[k] = \max_{-1 < t < 1} \frac{f^{(2m)}(t, 1 - d_j^n[k]/\lambda_j)}{(2m)!} \tag{7}$$

for $j = 1, 2, \ldots, q$. As seen from Equations (6) and (7), the error term at each step of the iteration decreases in magnitude as the order of summation $m$ grows. As the order of summation increases, the error at each step decreases, and thus, the number of iteration steps decreases and the algorithm converges more rapidly.

# 5  Implementation Results and Conclusions

We implemented the parallel $n$th root algorithm 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. In our experiments, we compute the $n$th roots of matrices for $n = 2, 3, 4, 5$ with dimensions ranging from 128 to 1024. These matrices are generated randomly with condition numbers $\kappa_2(A)$ up to $10^3$. The number of iteration steps performed by the parallel algorithm is tabulated in Table 3 for $m = 2, 4, 8$ processors, where the stopping criteria $\epsilon$ is $10^{-6}$. As can be seen from Table 3, increasing the number of processors (the order of summation) decreases the number of iteration steps for the parallel algorithm.

| Table 3 goes here |

We also implemented the three most commonly used sequential algorithms: the Hoskins-Walton iteration [Hoskins and Walton, 1979], the continued fraction method [Tsai *et al*, 1988], and Newton's method [Hoskins and Walton, 1979] [Higham, 1986]. It turns out that the Hoskins-Walton algorithm is the fastest of these three algorithms. Therefore, we compare the speedup of the proposed parallel algorithm to the fastest sequential algorithm. The speedup of the parallel algorithm for with respect to the sequential Hoskins-Walton algorithm is tabulated in Table 4.

| Table 4 goes here |

The implementation results indicate that the speedup of the parallel algorithm is greater than $P/2$ even for small matrices, where $P$ is the number of processors. As the matrix dimension grows, the speedup increases, e.g., the principal 4th root of a positive definite matrix of dimension 768 is computed with a speedup greater than 5 on 8 processors. Unfortunately, the proposed parallel algorithm is not highly scalable; the efficiency (the speedup divided by the number of processors) slightly decreases as

the number of processors increases when the matrix root and size are fixed. This was illustrated in Table 5 for computing the 5th of matrices. The rate of decrease becomes somewhat smaller as the size of the matrix increases: When the number of processors is increased from 2 to 4, the efficiency drops by 14.7 % for $q = 128$ and 11.9 % for $q = 1024$.

$$\boxed{\text{Table 5 goes here}}$$

A shortcoming of the proposed algorithm is it numerical stability problems for large values of $n$ since it requires $n$th power of the matrix $S_{k+1}$ at each iteration step. The algorithm works well and converges quickly for small $n$ (less than 10). Thus, its scope is somewhat limited. In the near future, we plan to compare of the proposed algorithm to the QR method from the point of numerical stability and computational cost. Furthermore, for large values of $n$, several approaches to compute $A^{1/n}$ are possible, e.g., the prime factorization of $n$ or the binary expansion of $n$. For example, $A^{1/6}$ can be computed using $A^{1/2}A^{1/3}$. These approaches need to be studied as well.

# 6    Acknowledgements

# References

[Bjork and Hammarling, 1983] Bjork, A. and S. Hammarling. A Schur method for the square root of a matrix. *Linear Algebra and its Applications*, 53:127–140, 1983.

[Davis and Rabinowitz, 1975] Davis, P. J. and P. Rabinowitz. *Methods of Numerical Integration*. New York, NY: Academic Press, 1975.

[Denman, 1981] Denman, E. D. Roots of real matrices. *Linear Algebra and its Applications*, 36:133–139, 1981.

[Gardiner and Laub, 1986] Gardiner, J. D. and A. J. Laub. A generalization of the matrix-sign-function solution for algebraic Riccati equations. *International Journal of Control*, 44(3):823–832, September 1986.

[Golub and van Loan, 1989] Golub, G. H. and C. F. van Loan. *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press, 2nd edition, 1989.

[Higham, 1986] Higham, N. J. Newton's method for the matrix square root. *Mathematics of Computation*, 46(174):537–549, April 1986.

[Higham, 1987] Higham, N. J. Computing real square roots of a real matrix. *Linear Algebra and its Applications*, 88:405–430, 1987.

[Hoskins and Walton, 1979] Hoskins, W. D. and D. J. Walton. A faster, more stable method for computing the $p$th roots of positive definite matrices. *Linear Algebra and its Applications*, 26:139–163, 1979.

[Kenney and Laub, 1989] Kenney, C. and A. J. Laub. Padé error estimates for the logarithm of a matrix. *International Journal of Control*, 50:707–730, 1989.

[Kenney and Laub, 1991] Kenney, C. 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.

[Pandey et al, 1990] Pandey, P., 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.

[Rinehart, 1955] Rinehart, R. F. The equivalence of definitions of a matric function. *The American Mathematical Monthly*, 3(62):395–414, 1955.

[Shieh et al, 1987] Shieh, L. S., S. R. Lian, and B. C. McInnis. Fast and stable algorithms for computing the principle square root of a complex matrix. *IEEE Transactions on Automatic Control*, 32(9):819–822, 1987.

[Shieh et al, 1990] Shieh, L. S., J. S. H. Tsai, and R. E. Yates. The generalized matrix sector functions and their applications to systems theory. *IMA Journal of Mathematical Control and Information*, 2:251–258, 1990.

[Shieh et al, 1986] Shieh, L. S., Y. T. Tsay, and R. E. Yates. Computation of the principal $n$th roots of complex matrices. *IEE Proceedings: Control Theory and Applications*, 130:111–118, 1986.

[Stroud and Secrest, 1966] Stroud, A. and D. Secrest. *Gaussian Quadrature Formulas*. Englewood Cliffs, NJ: Prentice-Hall, 1966.

[Tsai et al, 1988] Tsai, J. S. H., L. S. Shieh, and R. E. Yates. Fast and stable algorithms for computing the principal $n$th root of a complex matrix and the matrix sector functions. *Computers and Mathematics with Applications*, 15(11):903–913, 1988.

# Table Captions

**Table 1:** The parallel iteration.

**Table 2:** The complexity of the matrix operations.

**Table 3:** The number of iteration steps for the parallel algorithm.

**Table 4:** The speedup of the parallel algorithm.

**Table 5:** The efficiency of the parallel algorithm for computing the 5th root.

**Table 1:** The parallel iteration.

|  | |
|---|---|
| | $i = $ processor id |
| | $S_0 = I$ |
| | $Z_0 = I - A^{-1}$ |
| | `while` $\|Z_k\| < \epsilon$ |
| all: | `broadcast` $S_k$ and $Z_k$ |
| $i$: | `compute` $T_i = w_i h(x_i)(2I - (x_i + 1)Z_k)^{-1}$ |
| all: | `compute` $\Sigma = \sum_{i=1}^{m} T_i$ |
| all: | `compute` $S_{k+1} = (1/\pi) \csc(\pi/n) S_k \Sigma$ |
| all: | `compute` $Z_{k+1} = I - A^{-1} S_{k+1}^n$ |
| | `end` |

**Table 2:** The complexity of the matrix operations.

| Matrix Operation | Arithmetic Operations |
|---|---|
| scalar-matrix multiply | $q^2$ |
| matrix add & subtract | $q^2$ |
| matrix multiply | $2q^3 - q^2$ |
| matrix inversion | $\frac{8}{3}q^3 + O(q^2)$ |
| $n$th power of matrix | $(1.5 \log_2 n)(2q^3 - q^2)$ |

**Table 3:** The number of iteration steps for the parallel algorithm.

| | | Matrix Size | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $P$ | Root | 128 | 256 | 384 | 512 | 640 | 768 | 896 | 1024 |
| 2 | 2 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 |
| | 3 | 5 | 5 | 8 | 9 | 9 | 9 | 10 | 10 |
| | 4 | 6 | 12 | 14 | 15 | 15 | 16 | 16 | 16 |
| | 5 | 6 | 13 | 14 | 15 | 16 | 16 | 16 | 16 |
| 4 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| | 3 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 |
| | 4 | 6 | 7 | 7 | 8 | 8 | 8 | 9 | 10 |
| | 5 | 6 | 7 | 7 | 8 | 8 | 8 | 9 | 10 |
| 8 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 |
| | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 5 |
| | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 6 |
| | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 7 |

**Table 4:** The speedup of the parallel algorithm.

| $P$ | Root | Matrix Size | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 128 | 256 | 384 | 512 | 640 | 768 | 896 | 1024 |
| 2 | 2 | 1.14 | 1.26 | 1.34 | 1.40 | 1.46 | 1.56 | 1.62 | 1.62 |
| | 3 | 1.24 | 1.36 | 1.44 | 1.50 | 1.58 | 1.66 | 1.70 | 1.72 |
| | 4 | 1.30 | 1.42 | 1.48 | 1.54 | 1.62 | 1.70 | 1.76 | 1.82 |
| | 5 | 1.36 | 1.46 | 1.54 | 1.58 | 1.66 | 1.72 | 1.78 | 1.84 |
| 4 | 2 | 1.92 | 2.24 | 2.32 | 2.44 | 2.52 | 2.60 | 2.68 | 2.76 |
| | 3 | 2.04 | 2.32 | 2.48 | 2.56 | 2.68 | 2.80 | 2.88 | 2.96 |
| | 4 | 2.16 | 2.44 | 2.56 | 2.68 | 2.80 | 2.92 | 3.00 | 3.12 |
| | 5 | 2.32 | 2.52 | 2.64 | 2.76 | 2.88 | 2.96 | 3.08 | 3.24 |
| 8 | 2 | 3.36 | 3.68 | 4.00 | 4.16 | 4.40 | 4.64 | 4.96 | 5.20 |
| | 3 | 3.60 | 4.00 | 4.16 | 4.32 | 4.56 | 5.04 | 5.20 | 5.36 |
| | 4 | 3.84 | 4.24 | 4.40 | 4.64 | 4.88 | 5.28 | 5.44 | 5.68 |
| | 5 | 4.00 | 4.56 | 4.80 | 4.88 | 5.04 | 5.20 | 5.52 | 5.84 |

**Table 5:** The efficiency of the parallel algorithm for computing the 5th root.

| $P$ | Matrix Size | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 128 | 256 | 384 | 512 | 640 | 768 | 896 | 1024 |
| 2 | 0.68 | 0.73 | 0.77 | 0.79 | 0.83 | 0.86 | 0.89 | 0.92 |
| 4 | 0.58 | 0.63 | 0.66 | 0.69 | 0.72 | 0.74 | 0.77 | 0.81 |
| 8 | 0.50 | 0.57 | 0.60 | 0.61 | 0.63 | 0.65 | 0.69 | 0.73 |