Computers Math. Applic. Vol. 21, No. 2-3, pp. 187–195, 1991
Printed in Great Britain.

0097-4943/91 $3.00 + 0.00

A THREE-PHASE PARALLEL ALGORITHM FOR SOLVING LINEAR RECURRENCES

KUO-LIANG CHUNG
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan 10764, R. O. C.
FERNG-CHING LIN
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan 10764, R. O. C.
YEONG-NAN YEH
Institute of Mathematics, Academia Sinica, Taipei, Taiwan 11529, R. O. C.

(Received May, 1990)

Abstract—We present in this paper a three-phase parallel algorithm on the unshuffle network for solving linear recurrences. Through a detailed analysis on the special matrix multiplications involved in the computation we show that the first n terms of an mth order linear recurrence can be computed in $O(m^3 \log \frac{n}{m})$ time using $\Theta(\frac{n}{m \log \frac{n}{m}})$ processors. For the usual case when m is a small constant, the algorithm achieves cost optimality.

1. INTRODUCTION

Linear recurrences form an important class of computational problems which arise in many different application areas. Some parallel algorithms have been proposed before for solving linear recurrences. Stone and Kogge [5,6] first presented a recursive-doubling method to solve recurrences through a tricky divide-and-conquer reformulation of the recurrences into ones which use two indices. Based on a depth-width tradeoff in overlaid tree structure, Carlson and Sugla [1,2] presented a network containing multiple shuffle-exchange connections to compute the first n terms of a first-order linear recurrence in $O(n/p)$ time using $\Theta(p \log p)$ processors. Chung, Lin and Chen [3,7] presented a three-phase algorithm on the unshuffle network [8] for solving the first- and second-order linear recurrences, which reduces the $\Theta(n)$ processors used in [8] to $\Theta(n/\log n)$ while retaining the time complexity $O(\log n)$.

Consider the mth-order linear recurrence:

$$x_i = a_{i,1}x_{i-1} + a_{i,2}x_{i-2} + \cdots + a_{i,m}x_{i-m} + b_i. \tag{1.1}$$

Given $a_{i,j}$, b_i, $1 \le i \le n-1$, $1 \le j \le m$, and the initial values x_0, x_{-1}, $\ldots$, x_{i-m}, we want to compute all the values x_i, $1 \le i \le n-1$. For ease of exposition, we assume that n is a power of 2, $n > m$, and n/m is also a power of 2.

The recurrence in (1.1) can be converted into a first-order recurrence with the aid of companion matrices. Let $Z_i = [x_i, \ldots, x_{i-m+1}, 1]^t$ and

$$K_i = \begin{pmatrix} a_{i,1} & a_{i,2} & \cdots & a_{i,m-1} & a_{i,m} & b_i \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix},$$

which is an $(m+1)\times(m+1)$ matrix. We may write (1.1) as $Z_i = K_i Z_{i-1}$. Given Z_0 and K_i, $1 \le i \le n$, the problem becomes to compute Z_i, $1 \le i \le n-1$, because x_i equals the first component of Z_i. That is, our task is to compute all the prefix products of Z_{n-1} $(= K_{n-1}K_{n-2}\dots K_1 Z_0)$. Note that it takes $O(mn)$ time to compute all Z_i sequentially because the companion matrix-vector multiplication (CM-V) $K_i Z_{i-1}$ can be completed in $O(m)$ time, due to the special structure of the companion matrix K_i.

When parallel processing is involved, this good computation property of CM-Vs disappears. In the past, the complexity analysis was oversimplified by considering a matrix-matrix or matrix-vector multiplication as a one-time step and the parameter m was ignored [5,6]. Based on the three-phase algorithm on unshuffle networks [3,7], we present in this paper a parallel algorithm for solving general linear recurrences. We shall show sharper complexity results in terms of the three parameters n, m and the number of processors used. The performance of a parallel algorithm can be measured by Cost = Number of Processors × Execution Time. The minimum cost achievable by our algorithm is $\Theta(m^2 n)$ with $\Theta(m^3 \log \frac{n}{m})$ execution time and $\Theta(\frac{n}{m \log \frac{n}{m}})$ processors. Usually, m is a small constant in applications. In such a case, our algorithm achieves optimal cost $\Theta(n)$ with $O(\log n)$ time and $\Theta(n/\log n)$ processors.

2. THE UNSHUFFLE NETWORK

Due to the associativity of multiplication, the three multiplications in $Z_3 = K_3 K_2 K_1 K_0$, where $K_0 = Z_0$, can be performed in any order. Fig. 2.1 shows a possible computation tree for Z_3. The white nodes in the leftmost stage 0 are used for preparing input data only. The black nodes in stages 1 and 2 perform the multiplication.

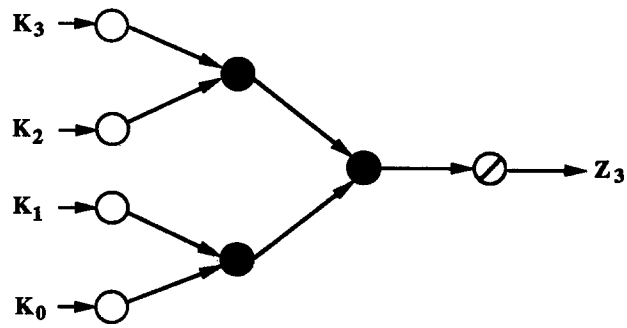

Figure 2.1. A computation tree of Z_3.

If we combine the computation trees for Z_i, $0 \le i \le 7$, an overlaid tree network, as shown in Fig. 2.2, is obtained for computing all the prefix products of Z_7. For general n, there are $\log n + 1$ stages of nodes in the network. If each node is implemented by a processor, the number of processors is as large as $n(\log n + 1)$, which is going to be reduced to $\Theta(\frac{n}{m \log \frac{n}{m}})$ later.

The multi-stage unshuffle network which can be used to simulate the overlaid tree network is depicted in Fig. 2.3, where $S_{i,j}$ and $P_{i,j}$, $0 \le i \le n-1$, $1 \le j \le \log n$, are simple devices and processors respectively. Let $q_0 q_1 \dots q_{l-1}$ $(l = \log n)$ be the binary representation of i. The output line of $P_{i,j}$ is connected to $S_{q_{l-1}q_0 q_1 \dots q_{l-2}, j+1}$ to provide the unshuffle data routing mechanism. The simple device $S_{i,j}$ is capable of producing two copies of its input and sending them to $P_{i,j}$ and $P_{i+1,j}$ except that the bottom $S_{n-1,j}$ has only one receiver $P_{n-1,j}$. For tracing the data passing in the network, each simple device is labelled by a number inside the parentheses to indicate where its data originate. The function of white nodes in a stage of Fig. 2.2 can be controlled by a proper masking mechanism in Fig. 2.3. For example, $P_{0,1}$ in stage 1, which is masked by a cross mark, merely transmits data to the next stage; $P_{0,2}$ and $P_{4,2}$ in stage 2 are masked also. Similarly, in stage 3, $P_{0,3}$, $P_{2,3}$, $P_{4,3}$, $P_{6,3}$ are masked.

Since the interconnection patterns between the stages of the multi-stage unshuffle network are all identical, we can compress all the stages together to obtain a single-stage unshuffle network as shown in Fig. 2.4, which has only n processors. After each iteration of execution, the processors feed the intermediate results back to the simple devices as inputs for the next iteration. The

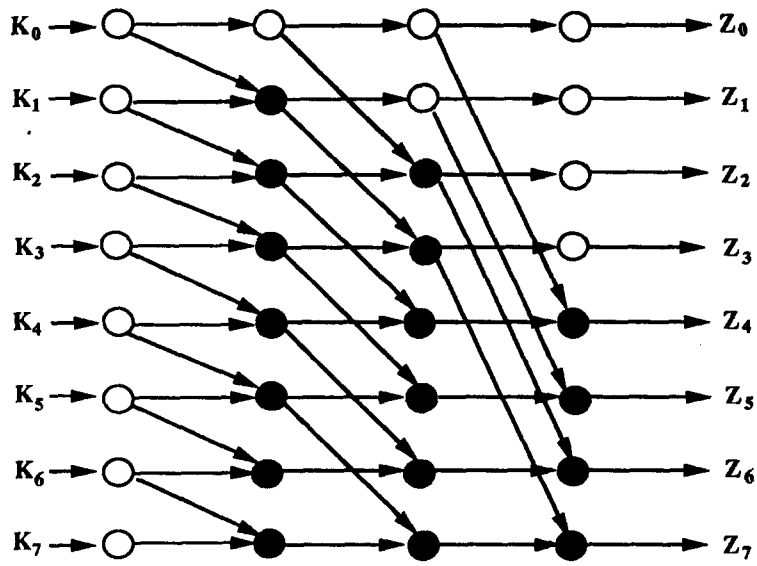

Figure 2.2. Overlaid tree network for computing all prefix values of Z_7.

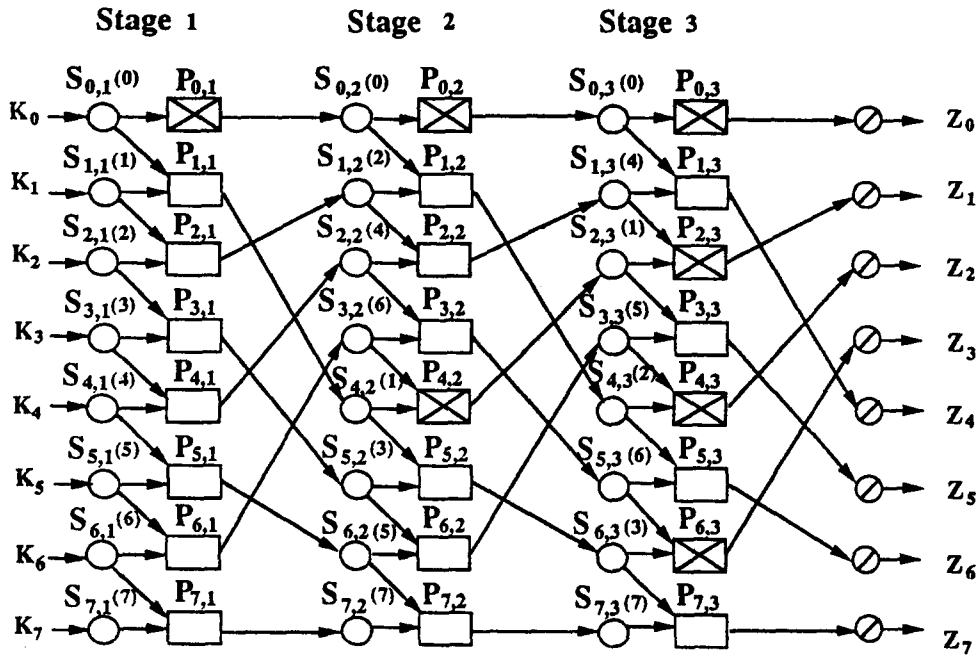

Figure 2.3. The multi-stage unshuffle network.

masking on the network at different iterations is exactly what we have described for different stages of the multi-stage network.

3. THE THREE-PHASE ALGORITHM

In the following, a modified three-phase algorithm is proposed to reduce the number of processors further. We start with an unshuffle network of $k(\leq n)$ processors with a bidirectional connection between each pair of P_i and S_i. To each processor P_i attach a local memory, which is a linear array of memory blocks $C_{i,j}$, $0 \leq j \leq n/k - 1$, as depicted in Fig. 3.1 for $n = 8$ and $k = 4$.

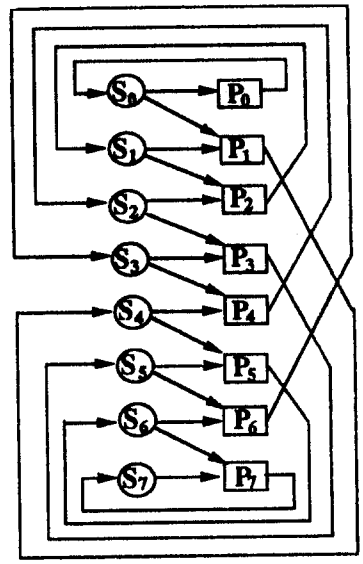

Figure 2.4. The single-stage unshuffle network.

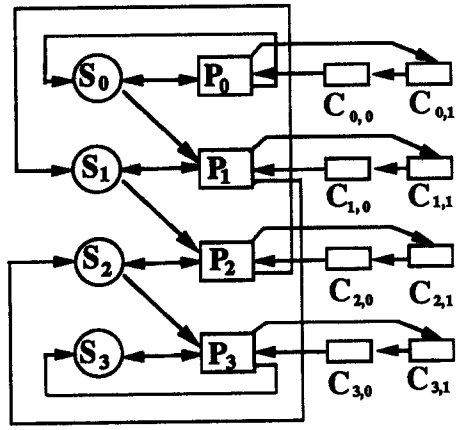

Figure 3.1. The reduced unshuffle network.

According to the value of mask register M_i resided in P_i, which is set by the masking mechanism, P_i takes one of the following actions:

$M_i = 0$: P_i sends the contents of its working storage Q_i to S_i. (Note that if only a one-way connection from S_i to P_i is allowed, sending data from P_i to S_i requires $O(\log k)$ communication steps.)

$M_i = 1$: P_i receives data from S_i and transmits them to $S_{q_{l-1}q_0q_1\cdots q_{l-2}}$, where $l = \log k$.

$M_i = 2$: P_i performs the multiplication on its two input data and sends the result to $S_{q_{l-1}q_0q_1q_{l-2}}$.

$M_i = 3$: Except P_0 which receives input data from S_0, P_i receives input data from S_{i-1}. The routing mechanism is forced to stop and the network goes to another phase of computation.

In the three-phase unshuffle network with k processors, the input data Z_0, K_1, ..., K_{n-2}, K_{n-1} are evenly divided into k pipes. Each local memory stores a pipe of data. The first pipe $Z_0, K_1, \ldots, K_{\frac{n}{k}-2}, K_{\frac{n}{k}-1}$ is stored in the memory associated with P_0. For $i \geq 2$, the ith pipe of data $K_{\frac{(i-1)n}{k}}, K_{\frac{(i-1)n}{k}+1}, \ldots, K_{\frac{in}{k}-1}$ is stored in the memory associated with P_{i-1}. Algorithm 3.1 describes how the network works. A computation step in the algorithm may contain a matrix-matrix or matrix-vector multiplication.

ALGORITHM 3.1.

PHASE 1. (Local prefix computations)

Each P_i sequentially computes n/k prefix products from its corresponding pipe of data and stores them in its local memory. The working storage Q_i has the final prefix product computed from the pipe of data and sends a copy to the working storage R_i in S_i.

PHASE 2. (Global prefix computations)

All the processors work together using the unshuffle routing mechanism for prefix product computations. After $\log k$ steps, the working storage R_i holds the value $Z_{(i+1)n/k}$, $0 \leq i \leq k-1$.

PHASE 3. (Final adaptations)

Each P_i except P_0 receives the prefix product from S_{i-1}. The n/k prefix products obtained in phase 1 are sequentially modified by performing a multiplication with the received prefix product.

We use $Z_7 = K_7K_6 \ldots K_0$ as an example to illustrate the algorithm. Let $n = 8$ and $k = 4$. Initially, the 8 input data (7 companion matrices and a vector) for Z_7 are evenly divided into 4 pipes, each containing 2 data. The cells $C_{0,0}$ and $C_{0,1}$ have the data K_0 and K_1 respectively. The cell $C_{i,j}$ has the data K_{2i+j} for $1 \leq i \leq 3$, $0 \leq j \leq 1$. We denote the product $K_i \ldots K_j$ by $F_{i,j}$ for $i \geq j \geq 0$.

Phase 1.

Each individual P_i computes $F_{2i,2i}$ and $F_{2i+1,2i}$ and stores them in its local memory sequentially. That is, P_0 computes all the prefix values of $F_{1,0}$; P_1 computes all the prefix values of $F_{3,2}$, and so on. The memory block $C_{i,j}$ saves $F_{2i+j,2i}$. After 2 steps in this phase, we have

$$\begin{aligned} C_{3,0} &= F_{6,6}, & C_{3,1} &= F_{7,6}, \\ C_{2,0} &= F_{4,4}, & C_{2,1} &= F_{5,4}, \\ C_{1,0} &= F_{2,2}, & C_{1,1} &= F_{3,2}, \\ C_{0,0} &= F_{0,0}, & C_{0,1} &= F_{1,0}, \end{aligned}$$

and

$$Q_3 = F_{7,6}, \qquad Q_2 = F_{5,4}, \qquad Q_1 = F_{3,2}, \qquad Q_0 = F_{1,0}.$$

Then, for all i, M_i becomes 0 and the contents of Q_i are sent to R_i.

Phase 2.

The global prefix computations are controlled by the unshuffle routing mechanism. During the first iteration, $M_0 = 1$ and $M_i = 2$ for $i = 1, 2, 3$. Once the P_i's complete their multiplications, the contents of R_i's become

$$\begin{aligned} R_3 &= F_{7,6} \bullet F_{5,4} = F_{7,4}, \\ R_2 &= F_{5,4} \bullet F_{3,2} = F_{5,2}, \\ R_1 &= F_{3,2} \bullet F_{1,0} = F_{3,0}, \\ R_0 &= F_{1,0}. \end{aligned}$$

During the second iteration, $M_i = 2$ for all i except $i \equiv 0 \bmod 2$ for which $M_i = 1$. Once the P_i's complete their multiplication operations again, the contents of R_i's become

$$\begin{aligned} R_3 &= F_{7,4} \bullet F_{3,0} = F_{7,0}, \\ R_2 &= F_{5,2} \bullet F_{1,0} = F_{5,0}, \\ R_1 &= F_{3,0}, \\ R_0 &= F_{1,0}. \end{aligned}$$

Let k be 2^h. In general, during the jth iteration, $1 \leq j \leq h$, $M_i = 2$ for all i, except $i \equiv 0 \bmod (2^{h+1-j})$ for which $M_i = 1$. After this phase of computation, R_i saves $F_{2i+1,0}$, a prefix value.

Phase 3.

During the first step of this phase, $M_i = 3$ and each P_i except P_0 receives input data from S_{i-1}. The contents of Q_i's become

$$Q_3 = F_{5,0}, \qquad Q_2 = F_{3,0}, \qquad Q_1 = F_{1,0}, \qquad Q_0 = F_{1,0}.$$

The unshuffle routing mechanism is forced to stop and each P_i sequentially modifies the contents of $C_{i,j}, j = 0, 1$, by performing the multiplication $C_{i,j} \bullet Q_i$. The final results are

$$\begin{aligned} C_{3,0} &= F_{6,6} \bullet F_{5,0} = F_{6,0}, & C_{3,1} &= F_{7,6} \bullet F_{5,0} = F_{7,0}, \\ C_{2,0} &= F_{4,4} \bullet F_{3,0} = F_{4,0}, & C_{2,1} &= F_{5,4} \bullet F_{3,0} = F_{5,0}, \\ C_{1,0} &= F_{2,2} \bullet F_{1,0} = F_{2,0}, & C_{1,1} &= F_{3,2} \bullet F_{1,0} = F_{3,0}, \\ C_{0,0} &= F_{0,0}, & C_{0,1} &= F_{1,0}. \end{aligned}$$

4. ANALYSIS OF THE ALGORITHM

In phase 1 of the three-phase algorithm, the good computation property of **CM-Vs** is reserved only in the first data pipe; the multiplications on the other pipes lose the property gradually. For the second pipe, it needs $(m+1)$ time to compute $K_{\frac{n}{k}+1}K_{\frac{n}{k}}(= K'_{\frac{n}{k}+1})$, needs $(2m+1)$ time to compute $K_{\frac{n}{k}+2}K'_{\frac{n}{k}+1}$ $(= K'_{\frac{n}{k}+2})$, and in general needs $i(m+1)$ time to compute $K_{\frac{n}{k}+i}K'_{\frac{n}{k}+i-1}$. Similar situations happen to the other pipes. One can also see the gradual change of time complexity in the steps of phases 2 and 3.

THEOREM 4.1. With the three-phase algorithm on k processors, the first n terms of an mth-order linear recurrence can be computed in $O(\frac{m^2 n}{k} + m^3 \log k)$ time when $n/k > m$ and in $O(\frac{mn^2}{k^2} + m^2 \log \frac{n}{m})$ time when $n/k \leq m$.

PROOF. For simplicity, we write $m+1$ as m without affecting the result of the analysis. There are two cases to be considered.

CASE 1. $n/k > m$.

Let $T_1(m,n,k)$ be the time to complete the local computations in phase 1, then

$$\begin{aligned} T_1(m,n,k) &= \left(\sum_{i=1}^{m-1} im\right) + (\frac{n}{k} - 1 - m + 1)m^2 \\ &= m^2(\frac{n}{k} - (\frac{m}{2} + \frac{1}{2})). \end{aligned} \tag{4.1}$$

The first term sums up the time slices for computing the companion matrix-unsaturated matrix multiplications (**CM-UMs**) until the resultant matrices are saturated. The time needed for the remaining computations is counted in the second term. During these steps, companion matrix-saturated matrix multiplications (**CM-SMs**) are performed.

Let $T_2(m,n,k)$ be the time to complete the global prefix computations in phase 2, then

$$\begin{aligned} T_2(m,n,k) &= m^3(\log k - 1) + m^2 \\ &= m^3 \log k - m^3 + m^2 \end{aligned} \tag{4.2}$$

The first term is the time needed to compute all the **SM-SMs**. The second term is the time to compute a **SM-V** at the last step.

Let $T_3(m,n,k)$ be the time to complete the adaptations in phase 3, then

$$\begin{aligned} T_3(m,n,k) &= \left(\sum_{i=1}^{m-1} im\right) + (\frac{n}{k} - m + 1)m^2 \\ &= m^2(\frac{1}{2} + \frac{n}{k} - \frac{m}{2}). \end{aligned} \tag{4.3}$$

The first term sums up the time to compute all the **UM-Vs**. The second term is the time for computing all the **SM-Vs**.

By combining (4.1), (4.2) and (4.3), we know that the total execution time is

$$\sum_{i=1}^{3} T_i(m,n,k) = m^2\left(\frac{n}{k} - \left(\frac{m}{2} + \frac{1}{2}\right)\right) + m^3 \log k - m^3 + m^2\left(\frac{1}{2} + \frac{n}{k} - \frac{m}{2}\right)$$
$$= O\left(\frac{m^2 n}{k} + m^3 \log k\right).$$

CASE 2. $n/k \leq m$.

We also try to calculate $T_i(m,n,k)$, $i = 1,2,3$. First, since all the computations in phase 1 are **CM-UMs**,

$$T_1(m,n,k) = \left(\sum_{i=1}^{n/k-1} im\right)$$
$$= \frac{mn}{2k}\left(\frac{n}{k} - 2\right). \tag{4.4}$$

For phase 2,

$$T_2(m,n,k) = \frac{mn^2}{k^2}\left(\sum_{i=0}^{(\log mk/n)-1} 4^i\right) + m^3\left(\log k - \log \frac{mk}{n} + 1 - 1\right) + m^2$$
$$= \frac{mn^2}{k^2}\left(\frac{\left(\frac{mk}{n}\right)^2 - 1}{3}\right) + m^3 \log \frac{n}{m} - m^3 + m^2. \tag{4.5}$$

The first term sums up the time slices for computing the **UM-UMs** until the resultant matrices are saturated. The second term is the time needed to compute the **SM-SMs**. The third term is the time to compute a **SM-V** at the last step. Finally, for phase 3,

$$T_3(m,n,k) = \left(\sum_{i=1}^{n/k} im\right)$$
$$= m\frac{n}{2k}\left(\frac{n}{k} + 1\right), \tag{4.6}$$

which is the time needed to compute all the **UM-Vs** in phase 3.

By combining (4.4), (4.5) and (4.6), we know that the total execution time is

$$\sum_{i=1}^{3} T_i(m,n,k) = \frac{mn}{2k}\left(\frac{n}{k} - 2\right) + \frac{mn^2}{k^2}\left(\frac{\left(\frac{mk}{n}\right)^2 - 1}{3}\right) + m^3 \log \frac{n}{m} - m^3 + m^2$$
$$= \frac{2mn^2}{3k^2} - \frac{2m^3}{3} + m^3 \log \frac{n}{m} + m^2 - \frac{mn}{2k}$$
$$= O\left(\frac{mn^2}{k^2} + m^3 \log \frac{n}{m}\right). \blacksquare$$

We may use $k = n$ processors in the network to achieve the minimum execution time $T_2(m,n,k) = O(m^3 \log n)$. However, this time complexity can also be met by using fewer processors.

THEOREM 4.2. For solving mth-order linear recurrences, the minimum cost achievable by the three-phase algorithm is $O(m^2 n)$ with $O(m^3 \log \frac{n}{m})$ execution time and $\Theta(\frac{n}{m \log \frac{n}{m}})$ processors.

PROOF.

From (4.1), (4.2) and (4.3), the computation cost for the first case is

$$\begin{aligned}\text{Cost}_1 &= k\left(\sum_{i=1}^{3} T_i(m,n,k)\right)\\ &= 2m^2n + km^3\log\frac{k}{4} + O(km^2).\end{aligned}$$

Let $k' = k/4$, then

$$\text{Cost}_1 = 2m^2n + 4k'm^3\log k' + O(k'm^2).$$

Now we consider the equation:

$$c_1m^2n = 4k'm^3\log k' \qquad \text{for some constant } c_1.$$

We can use a bootstrapping technique [4] to choose $k = 4k' = \Theta(\frac{n}{m\log\frac{n}{m}})$ to determine the minimum value of Cost_1 as $O(m^2n)$. In addition, the computing time is $O(m^3\log\frac{n}{m})$ for this case.

From (4.4), (4.5) and (4.6), the computation cost for the second case is

$$\begin{aligned}\text{Cost}_2 &= k\left(\sum_{i=1}^{3} T_i(m,n,k)\right)\\ &= \frac{2}{3}\frac{mn^2}{k} - \frac{2km^3}{3} + km^3\log\frac{n}{m} - \frac{mn}{2} + m^2k\\ &= \frac{2}{3}\frac{mn^2}{k} - k\left(\frac{2m^3}{3} - m^3\log\frac{n}{m} - m^2\right) - \frac{mn}{2}.\end{aligned}$$

Let $\text{Cost}_2 = h(m,n,k)$. Since $\frac{\partial^2 h(m,n,k)}{\partial k^2} = \frac{4mn^2}{3k^3} > 0$, $h(m,n,k)$ is a convex function in k. We might choose the critical point $k = \frac{n}{\sqrt{m(\frac{3}{2}m\log\frac{n}{m}+\frac{3}{2}-m)}}$ (see Fig. 4.1), which satisfies $\frac{\partial h(m,n,k)}{\partial k} = -\frac{2mn^2}{3k^2} - \frac{2m^3}{3} + m^3\log\frac{n}{m} + m^2 = 0$, to minimize the value of Cost_2. However, the value is illegal because the premise says that $k \geq \frac{n}{m}$. Therefore, the minimal value of Cost_2 is $O(nm^2\log\frac{n}{m})$ when we choose $k = \frac{n}{m}$. The time complexity is also $\Theta(m^3\log\frac{n}{m})$. Since the cost is higher than that for case 1, we have to abort this case. ∎

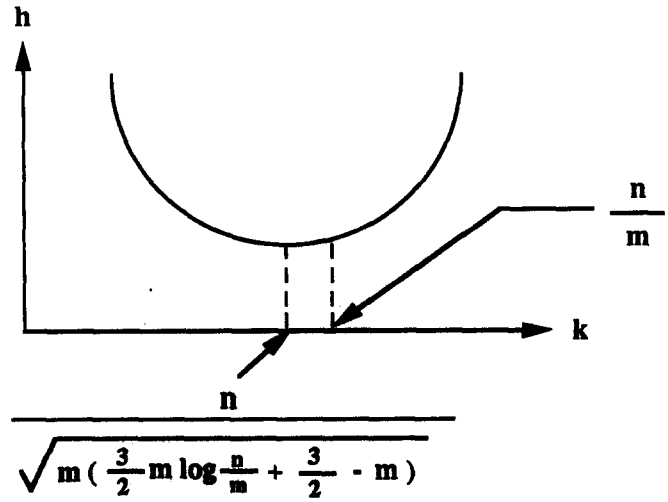

Figure 4.1. The convex function h.

5. CONCLUSION

We have proposed a three-phase algorithm on the unshuffle network to solve linear recurrences. Because the good computation property of CM-Vs in sequential computation cannot be retained in parallel computation, when m is large the optimal cost $O(mn)$ of the problem is not met by our algorithm. The minimum cost achievable by the algorithm is $O(m^2 n)$ with $O(m^2 \log \frac{n}{m})$ execution time and $\Theta(\frac{n}{m \log \frac{n}{m}})$ processors. However, for the usual case when m is a small constant, the algorithm does achieve the optimal cost $O(n)$ with $O(\log n)$ execution time and $\Theta(\frac{n}{\log n})$ processors.

REFERENCES

1. D. A. Carlson and B. Sugla, Time and processor efficient parallel algorithms for recurrence equations and related problems, *Proc. 1984 Int. Conf. on Parallel Processing, IEEE Computer Society Press, Washington D.C.*, 310–314 (1984).
2. D. A. Carlson and B. Sugla, Adapting shuffle-exchange like parallel processing organizations to work as systolic arrays, *Parallel Computing* **11** (1), 93–106 (1989).
3. K. L. Chung, F. C. Lin, and W. C. Chen, Parallel computation of continued fractions, under revision for *J. of Parallel and Distributed Computing*.
4. R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, (1989).
5. P. M. Kogge and H. S. Stone, A parallel algorithm for the efficient solution of a general class of recurrence equations, *IEEE Transactions on Computers* **C-22** (8), 786–793 (1973).
6. P. M. Kogge, Parallel solution of recurrence problems, *IBM J. Research and Development*, 138–148 (1974).
7. F. C. Lin and K. L. Chung, A cost-optimal parallel tridiagonal system solver, to appear in *Parallel Computing*.
8. H. S. Stone, *Introduction to Computer Architecture*, Science Research Associates, Chicago, IL, (1980).