CVGIP: GRAPHICAL MODELS AND IMAGE PROCESSING
Vol. 53, No. 6, November, pp. 601–605, 1991

NOTE

A Cost-Optimal Parallel Algorithm for B-Spline Surface Fitting

KUO-LIANG CHUNG AND FERNG-CHING LIN

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 10764, Republic of China

Received December 28, 1989; accepted May 6, 1991

We show how to transform the B-spline surface fitting problem into suffix computations of continued fractions. Then a parallel substitution scheme is used to compute the suffix values on a newly proposed mesh-of-unshuffle network. The derived parallel algorithm allows the surface interpolation at $m \times n$ points to be solved in $O(\log m \log n)$ time using $\Theta(mn/(\log m \log n))$ processors. The algorithm is cost-optimal in the sense that number of processors times execution time is minimized. The problem can be even more quickly solved in $O(\log m + \log n)$ time if $\Theta(mn)$ processors are used in the network.

1. INTRODUCTION

The task of surface fitting is to construct a smooth surface that fits a given set of $m \times n$ points in the space. Surface fitting is important in graphics, image processing, pattern recognition, and computer-aided geometric design [1, 3, 6]. B-spline surface interpolation is a good fitting tool because a local shape change of the surface affects only its vicinity. Some sequential methods for efficient surface interpolation in $O(mn)$ time using B-splines have been proposed [2]. Recently, on the basis of the cyclic reduction technique, Cheng and Goshtasby [4] presented a parallel algorithm to do B-spline surface fitting in $O(\log mn)$ time using $\Theta(mn)$ processors. However, their computation model is rather abstract; no configuration of processors' network was specified.

In this paper, we convert the B-spline surface fitting problem into suffix computations of continued fractions that correspond to some tridiagonal systems of linear equations. Then, a substitution scheme for computing the suffix values on a newly proposed mesh-of-unshuffle network is introduced. The parallel algorithm so derived allows the surface fitting problem to be solved in $O(\log m \log n)$ time using $\Theta(mn/(\log m \log n))$ processors. It achieves cost optimality in the sense that number of processors times execution time is minimized. Moreover, if $\Theta(mn)$ processors are used in the network, the problem can be more quickly solved in $O(\log m + \log n)$ time.

2. PRELIMINARIES

Suppose we are given the set of points $(i, j, T_{i,j})$, $1 \le i \le m$, $1 \le j \le n$. The uniform bicubic B-spline surface for interpolating the points consists of $(m-1) \times (n-1)$ patches $T_{i,j}(u, v)$, $1 \le i \le m-1$, $1 \le j \le n-1$. Each is defined by a bicubic polynomial,

$$T_{i,j}(u, v) = \tfrac{1}{36}[u^3, u^2, u, 1]NO_{i,j}N^t[v^3, v^2, v, 1]^t,$$

in which

$$N = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix}$$

and

$$O_{i,j} = \begin{pmatrix} W_{i-1,j-1} & W_{i-1,j} & W_{i-1,j+1} & W_{i-1,j+2} \\ W_{i,j-1} & W_{i,j} & W_{i,j+1} & W_{i,j+2} \\ W_{i+1,j-1} & W_{i+1,j} & W_{i+1,j+1} & W_{i+1,j+2} \\ W_{i+2,j-1} & W_{i+2,j} & W_{i+2,j+1} & W_{i+2,j+2} \end{pmatrix}.$$

$W_{i,j}$, $0 \le i \le m+1$, $0 \le j \le n+1$, are the control points of the surface to be determined.

According to [2], the corner points of the patches can be expressed by a weighted average of the control points:

$$\begin{aligned} T_{i,j} = \tfrac{1}{36}(&W_{i-1,j-1} + 4W_{i,j-1} + W_{i+1,j-1} \\ &+ 4W_{i-1,j} + 16W_{i,j} + 4W_{i+1,j} \\ &+ W_{i-1,j+1} + 4W_{i,j+1} + W_{i+1,j+1}) \\ &\text{for } 1 \le i \le m,\ 1 \le j \le n. \end{aligned}$$

They form a system of mn equations in $(m+2)(n+2)$ unknowns. In order to completely solve the system, we

1049-9652/91 $3.00

need the following $(m + 2)(n + 2) - mn = 2(m + n + 2)$ additional equations to specify how the boundary points are interpolated:

$$W_{0,j} = W_{1,j}; \quad W_{m+1,j} = W_{m,j}, \ 1 \le j \le n;$$
$$W_{i,0} = W_{i,1}; \quad W_{i,n+1} = W_{i,n}, \ 0 \le i \le m + 1.$$

By [3], the above system of equations can be equivalently transformed into

$$B_{m\times m}\begin{pmatrix} W_{1,i} \\ W_{2,i} \\ \vdots \\ W_{m,i} \end{pmatrix} = 6\begin{pmatrix} H_{1,i} \\ H_{2,i} \\ \vdots \\ H_{m,i} \end{pmatrix} \quad \text{for } i = 1, 2, \ldots, n, \quad (2.1)$$

$$B_{n\times n}\begin{pmatrix} H_{i,1} \\ H_{i,2} \\ \vdots \\ H_{i,n} \end{pmatrix} = 6\begin{pmatrix} T_{i,1} \\ T_{i,2} \\ \vdots \\ T_{i,n} \end{pmatrix} \quad \text{for } i = 1, 2, \ldots, m, \quad (2.2)$$

where the matrices $B_{m\times m}$ and $B_{n\times n}$ are of the form

$$\begin{pmatrix} 5 & 1 & 0 & \cdot & \cdot & 0 \\ 1 & 4 & 1 & 0 & \cdot & 0 \\ 0 & 1 & 4 & 1 & \cdot & 0 \\ \cdot & 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & 0 & 1 & 4 & 1 \\ 0 & \cdot & \cdot & 0 & 1 & 5 \end{pmatrix}.$$

Therefore, B-spline surface interpolation becomes a two-part process, namely, solving the m tridiagonal systems in (2.2) for $H_{m\times n}$ first and then solving the n tridiagonal systems in (2.1) for $W_{m\times n}$.

3. TRANSFORMATIONS TO CONTINUED FRACTIONS

Consider a tridiagonal system of linear equations:

$$d_1x_1 + e_1x_2 = b_1,$$
$$c_ix_{i-1} + d_ix_i + e_ix_{i+1} = b_i \quad \text{for } i = 2, 3, \ldots, n - 1,$$
$$c_nx_{n-1} + d_nx_n = b_n.$$

For $k = 2, 3, \ldots, n - 1$, we may eliminate x_{k-1} from the kth equation and get

$$d'_kx_k + e_kx_{k+1} = b'_k,$$

where

$$d'_k = d_k - c_ke_{k-1}/d'_{k-1} \quad (3.1)$$

and

$$b'_k = b_k - c_kb'_{k-1}/d'_{k-1}. \quad (3.2)$$

During the back-substitution, we first have

$$x_n = \frac{b'_n}{d'_n}$$

and then

$$x_k = \frac{b'_k - e_kx_{k+1}}{d'_k} \quad (3.3)$$

for $k = n - 1, n - 2, \ldots, 1$.

The three recurrences in (3.1), (3.2), and (3.3) can be further transformed into suffix computations of continued fractions (CFs). Let us represent the CF

$$p_1 + \cfrac{q_2}{p_2 + \cfrac{q_3}{\cdots \cfrac{}{p_{n-1} + \cfrac{q_n}{p_n}}}}$$

in a more compact form:

$$p_1 + \frac{q_2}{p_2+}\frac{q_3}{p_3+}\cdots\frac{q_n}{p_n}.$$

From (3.1), d'_n can be expanded into a CF:

$$d'_n = d_n + \frac{-c_ne_{n-1}}{d_{n-1}+}\frac{-c_{n-1}e_{n-2}}{d_{n-2}+}\cdots\frac{-c_3e_2}{d_2+}\frac{-c_2e_1}{d_1}.$$

We may assume that n is a power of 2; otherwise, pad d'_n with a special CF,

$$\frac{0}{1+}\frac{0}{1+}\cdots\frac{0}{1},$$

to enlarge n without altering the computation results. Our goal is to speed up the computing of all d'_k, $k = 1, 2, \ldots, n$.

Let the rational form $(r_1 + r_2d'_k)/(r_3 + r_4d'_k)$ be represented by $(r_1, r_2, r_3, r_4, d'_k)$. The CF of, say, d'_8 can be decomposed into eight sub-CFs:

$$d'_8 = d_8 + \frac{-c_8e_7}{d'_7} = (-c_8e_7, d_8, 0, 1, d'_7),$$

$$d'_7 = d_7 + \frac{-c_7e_6}{d'_6} = (-c_7e_6, d_7, 0, 1, d'_6),$$

$$\vdots$$

$$d_2' = d_2 + \frac{-c_2e_1}{d_1'} = (-c_2e_1,\ d_2,\ 0,\ 1,\ d_1'),$$

$$d_1' = \frac{d_1}{1 + d_0'} = (d_1,\ 0,\ 1,\ 1,\ d_0').$$

The variable d_0' will be replaced by zero after all the substitutions are completed.

Since substituting $d_i' = (s_1 + s_2d_j')/(s_3 + s_4d_j')$ into $(r_1 + r_2d_i')/(r_3 + r_4d_i')$ gives $((r_1s_3 + r_2s_1) + (r_1s_4 + r_2s_2)d_j')/((r_3s_3 + r_4s_1) + (r_3s_4 + r_4s_2)d_j')$, the substitution operation should be defined as

$$(r_1,\ r_2,\ r_3,\ r_4,\ d_i') \circ (s_1,\ s_2,\ s_3,\ s_4,\ d_j')$$
$$= (r_1s_3 + r_2s_1,\ r_1s_4 + r_2s_2,\ r_3s_3 + r_4s_1,\ r_3s_4 + r_4s_2,\ d_j').$$

This operation can be verified to be associative.

Due to the associativity of the substitution operation, we can use the overlaid tree network shown in Fig. 3.1 to compute all the suffix values of d_8'. The variables are not real data and hence are omitted there. The white nodes are used for transmitting data only. The black nodes perform the substitution operation. There are $\log n + 2$ stages in this network, meaning that the computation time is only $O(\log n)$. If each node is implemented by a processor, the number of processors is as large as $\Theta(n \log n)$, which is to be ultimately reduced to $\Theta(n/\log n)$ later.

Now for the computation of b_k' in (3.2), the CF for, say, b_4'/b_3', is given by

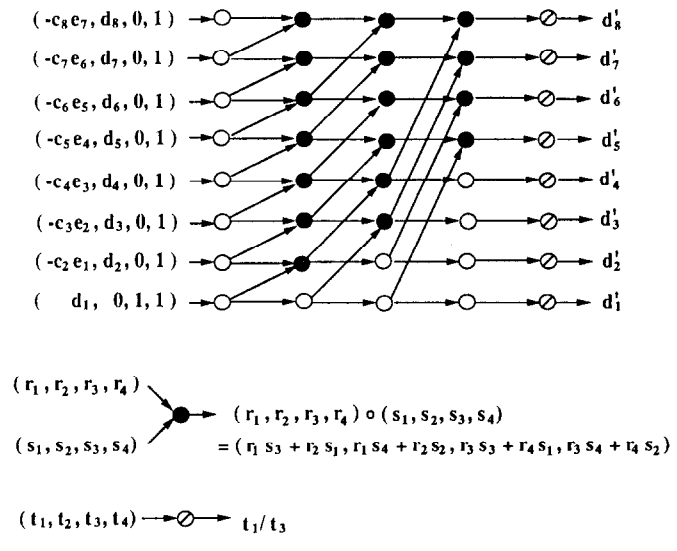

FIG. 3.1. The overlaid tree network for computing suffix values of d_8'.

$$X_4 = \frac{b_4'}{b_3'} = \frac{b_4 - (c_4/d_3')b_3'}{b_3'} = (-c_4/d_3') + \frac{b_4}{b_3'}.$$

When x is of the form $r_1 + r_2/r_3$, we define $N(x)$ to be the value $r_1r_3 + r_2$. The sub-CFs for the above CF can be written as

$$X_4 = -c_4/d_3' + \frac{b_4}{N(X_3)},$$

$$X_3 = -c_3/d_2' + \frac{b_3}{N(X_2)},$$

$$X_2 = -c_2/d_1' + \frac{b_2}{N(X_1)},$$

$$X_1 = \frac{b_1}{1 + N(X_0)},$$

where $N(X_0) = 0$, $N(X_1) = b_1$, and $b_i' = -(c_i/d_{i-1})b_{i-1}' + b_i = N(X_i)$ for all i. The substitution operation for this case is defined as

$$(r_1,\ r_2,\ r_3,\ r_4,\ N(X_i)) \circ (s_1,\ s_2,\ s_3,\ s_4,\ N(X_j))$$
$$= (r_1 + r_2s_1,\ r_2s_2,\ r_3 + r_4s_1,\ r_4s_2,\ N(X_j)).$$

This operation is associative, too. The problem of solving the recurrence equation in (3.2) becomes the suffix computations of X_n and the network in Fig. 3.1 can be used again.

Similarly, from (3.3), the sub-CFs for, say, x_{n-3}/x_{n-2}, are given by

$$Y_{n-3} = -e_{n-3}/d_{n-3}' + \frac{b_{n-3}'/d_{n-3}'}{N(Y_{n-2})},$$

$$Y_{n-2} = -e_{n-2}/d_{n-2}' + \frac{b_{n-2}'/d_{n-2}'}{N(Y_{n-1})},$$

$$Y_{n-1} = -e_{n-1}/d_{n-1}' + \frac{b_{n-1}'/d_{n-1}'}{N(Y_n)},$$

$$Y_n = \frac{b_n'/d_n'}{1 + N(Y_{n+1})},$$

where $N(Y_{n+1}) = 0$, and $x_i = N(Y_i)$ for all i. So the problem of solving the recurrence equation in (3.3) becomes the suffix computations of Y_1.

4. THREE-PHASE ALGORITHM ON UNSHUFFLE NETWORK

It is well known that an unshuffle network of n processors can simulate the overlaid tree network with n inputs with the same time complexity [7, 9]. In the following, a modified three-phase algorithm is proposed to reduce the

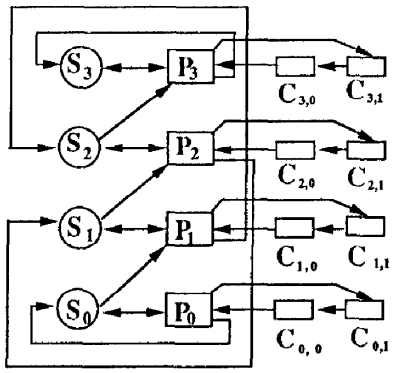

FIG. 4.1. The unshuffle network with local memories.

number of processors further. To save space, we only discuss the suffix computations of d'_n. By adapting appropriate operations in the processors, the suffix computations of X_n and Y_1 can also be performed on the same network with the same time complexity.

We start with an unshuffle network of k ($\leq n$) processors. Attach to each processor a serial memory [8], which consists of a linearly connected array of n/k memory cells, as depicted in Fig. 4.1 for $n = 8$ and $k = 4$. In each execution cycle, the attached memories rotate all their data one position. P_i executes on the data shifted in. Algorithm 4.1 describes how the network works. Initially, n input data (quadruples) are evenly divided into k pipes to be separately stored in the local memories.

Algorithm 4.1

Phase 1 (Local Suffix Computations). Each P_i sequentially computes n/k suffix values from its corresponding pipe of data and stores them in its local memory. Register Q_i in P_i has the final rational form computed from the pipe of data and transmits it to register R_i in S_i.

Phase 2 (Global Suffix Computations). By using the unshuffle routing mechanism, all the processors work together on the data in R_i's for suffix computations. After $\log k$ steps, register R_i holds the rational form of $d'_{(i+1)n/k}$, $0 \leq i \leq k - 1$.

Phase 3 (Final Adaptations). Each P_i except P_0 receives the rational form from S_{i-1} and sequentially modifies the suffix values calculated in phase 1 by substituting them into the received rational form.

Both phase 1 and phase 3 need n/k steps. Phase 2 needs $\log k$ steps. So the three-phase algorithm takes $O(n/k + \log k)$ time to finish computing all the suffix values of d'_n. For the subsequent parts of computation, namely, computing the suffix values of X_n and Y_1, the algorithm works equally well if we properly adjust the substitution operations in the processors. Preparing the input data from one part of computation to another takes $O(n/k)$ time, where n/k is the size of each data pipe. So we have the following lemma.

LEMMA 5.1. *A tridiagonal system can be solved in $O(n/k + \log k)$ time on an unshuffle network of k processors.*

5. COST OPTIMALITY

The performance of a parallel algorithm can be measured by Cost = Number of Processors × Execution Time. Given a problem, if the cost of a parallel algorithm matches the sequential time lower bound within a constant factor, the parallel algorithm is said to be cost optimal. In the case of solving a tridiagonal system, since there are n values to be computed, the sequential time lower bound is clearly $\Omega(n)$. Likewise, the sequential time lower bound for B-spline surface fitting is $\Omega(mn)$.

If we select $k = \Theta(n/\log n)$ in Lemma 5.1, we have the following theorem.

THEOREM 5.2. *A tridiagonal system can be solved in $O(\log n)$ time on an unshuffle network of $\Theta(n/\log n)$ processors.*

We now propose a mesh-of-unshuffle network to achieve the cost optimality for B-spline surface fitting. The network, as shown in Fig. 5.1, consists of a mesh of processors in which all the rows and columns are unshuffle networks. A tridiagonal system in (2.2) can be solved in $O(\log n)$ time on a row of processors. Since there are m tridiagonal systems, we need to use m rows in parallel. If we compact $\Theta(\log m)$ rows into one, each processor has a local memory of size $\Theta(\log n \log m)$. The time to solve (2.2) becomes $O(\log m \log n)$. We then use the $\Theta(n/\log n)$ columns of the mesh to solve the n tridiagonal systems in (2.1) in $O(\log n \log m)$ time.

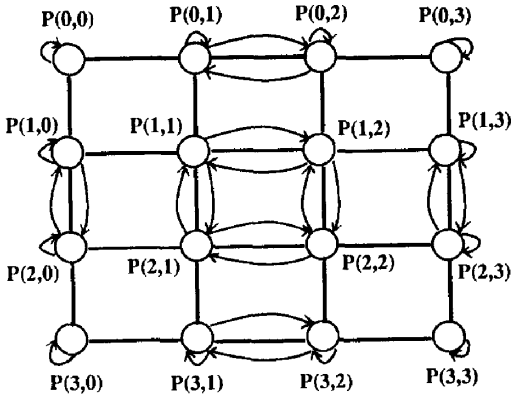

FIG. 5.1. The mesh-of-unshuffle network.

THEOREM 5.3. *The B-spline surface fitting problem can be cost-optimally solved in* $O(\log m \log n)$ *time on a mesh-of-unshuffle network of* $\Theta(mn/\log m \log n))$ *processors.*

Suppose we increase the number of processors from $\Theta(mn/(\log m \log n))$ to $\Theta(mn)$ and decrease the size of each local memory from $\Theta(\log m \log n)$ to $\Theta(1)$. By similar arguments, (2.2) can be solved in $O(\log n)$ time and (2.1) can be solved in $O(\log m)$ time.

THEOREM 5.4. *The B-spline surface fitting problem can be solved in* $O(\log m + \log n)$ *time on a mesh-of-unshuffle network with* $\Theta(mn)$ *processors.*

6. CONCLUDING REMARKS

The parallel substitution scheme was originally presented in [5] for the parallel computation of general CFs. In this paper, we first transform the B-spline surface fitting problem into tridiagonal systems of linear equations. Each tridiagonal system is then transformed into three recurrence equations. The recursive-doubling method [9] is an alternative approach to solving these equations through a rather tricky divide-and-conquer reformulation of the recurrences into ones with two indices. Our approach is simply to do the recurrences' CF expansions and apply the straightforward substitution concept. The associativity of the substitution operation not only brings out the parallelism for the computation naturally, but also supplies the idea of three-phase computation to reduce the number of processors. Finally, we conjecture that the B-spline surface fitting problem may be solved in $O(\log m + \log n)$ time using $\Theta(nm/(\log m + \log n))$ processors.

REFERENCES

1. R. E. Barnhill and R. F. Riesenfeld (Eds.), *Computer Aided Geometric Design*, Academic Press, New York, 1974.
2. B. A. Barsky and D. P. Greenberg, Determining a set of B-spline control vertices to generate an interpolating surface, *Comput. Graphics Image Process.* **14**(3), 1980, 203–226.
3. C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.
4. F. H. Cheng and A. Goshtasby, A parallel B-spline surface fitting algorithm, *ACM Trans. Graphics* **8**, 1, 1989, 41–50.
5. K. L. Chung, F. C. Lin, and W. C. Chen, Parallel computation of continued fractions, *J. Parallel Distrib. Comput.*, to appear.
6. I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Wiley, New York, 1979.
7. S. L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Statist. Comput.* **8**, 3, 1987, 354–392.
8. R. M. Owens and J. Ja'Ja', Parallel sorting with serial memories, *IEEE Trans. Comput.* **C-34**(4), 1985, 379–383.
9. H. S. Stone, Parallel tridiagonal equation solvers, *ACM Trans. Math. Software* **1**(4), 1975, 289–307.