Cost-Optimal Parallel B-Spline Interpolations*

KUO-LIANG CHUNG, FERNG-CHING LIN and WEN-CHIN CHEN

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 10764, R. O. C.

Abstract. We show how to transform the B-spline curve and surface fitting problems into suffix computations of continued fractions. Then a parallel substitution scheme is introduced to compute the suffix values on a newly proposed mesh-of-unshuffle network. The derived parallel algorithm allows the curve interpolation through n points to be solved in $O(\log n)$ time using $\Theta(n/\log n)$ processors and allows the surface interpolation through $m \times n$ points to be solved in $O(\log m \log n)$ time using $\Theta\left(mn/(\log m \log n)\right)$ processors. Both interpolation algorithms are cost-optimal for their respective problems. Besides, the surface fitting problem can be even faster solved in $O(\log m + \log n)$ time if $\Theta(mn)$ processors are used in the network.

1. INTRODUCTION

Curve and surface fittings are important in graphics, image processing, pattern recognition, and computer-aided geometric design [1,4,8,13,14]. The task of curve fitting is to construct a smooth curve that goes through a given set of n points in the plane. For surface fitting, it is to construct a smooth surface that fits a given set of $m \times n$ points in the space.

B-spline curve/surface interpolation is a good fitting tool because a local shape change of the curve/surface only affects its vicinity. Some sequential methods have been proposed for efficient curve (surface) interpolation in $O(n)$ $\left(O(mn)\right)$ time using B-splines [2,9,14]. Recently, based on a cyclic reduction technique, Cheng and Goshtasby [5] presented a parallel algorithm to do B-spline surface fitting in $O(\log mn)$ time using $\Theta(mn)$ concurrent processors. However, their computation model is rather abstract, no configuration of processors' network was specified.

In this paper, we first show that the B-spline curve and surface fitting problems can be converted into suffix computations of continued fractions which correspond to some tridiagonal systems of linear equations. Then, a parallel substitution scheme is introduced to compute the suffix values of the continued fractions on a newly proposed mesh-of-unshuffle network. The mesh-of-unshuffle network is a mesh in which the processors on each row or column communicate through an unshuffle routing mechanism. The parallel algorithm so derived allows the curve (surface) fitting problem to be solved in $O(\log n)$ $\left(O(\log m \log n)\right)$ time with $\Theta(n/\log n)$ $\left(\Theta(mn/(\log m \log n))\right)$ processors. Both interpolators achieve cost-optimality in the sense that the product of execution time and processor number is minimized. Besides, if $\Theta(mn)$ processors are used in the network, the surface fitting problem can be as fast solved in $O(\log m + \log n)$ time as in Cheng and Goshtasby's.

The rest of the paper is organized as follows. In Section 2, the curve and surface fitting problems are converted into tridiagonal systems of linear equations. Then, in Section 3, the solution of a tridiagonal system is transformed into suffix computations of some continued fractions. Section 4 introduces the parallel substitution scheme to compute the suffix values on the mesh-of-unshuffle network of processors. In Section 5, we justify the number of processors required in the network to achieve cost-optimality for both of the curve and surface fitting problems. We also point out how to use $\Theta(mn)$ processors to solve the surface fitting problem in $O(\log m + \log n)$ time. Concluding remarks are given in Section 6.

2. PRELIMINARIES

2.1 *B-spline Curve Fitting*

In B-spline curve fitting , we use a spline, which is a third-degree (or cubic) polynomial, between every two adjacent points (or knots). That is, the curve is composed of $n-1$ cubic polynomials:

* This research is supported in part by National Science Council of the Republic of China under contracts NSC78 − 0408−E002 − 14 and NSC79 − 0408−E002 − 07.

$$y = f_i(x - x_i)^3 + g_i(x - x_i)^2 + h_i(x - x_i) + k_i, \quad i = 1, 2, ..., n-1. \tag{2.1}$$

The ith cubic polynomial between (x_i, y_i) and (x_{i+1}, y_{i+1}) is obtained once the coefficients f_i, g_i, h_i, and k_i are determined. Because the spline must touch the knots, we have

$$k_i = y_i, \tag{2.2}$$

$$f_i l_i^3 + g_i l_i^2 + h_i l_i + k_i = y_{i+1}, \tag{2.3}$$

where $l_i = x_{i+1} - x_i$.

Let D_i and D_{i+1} represent the second derivatives of (2.1) at x_i and x_{i+1} respectively, then

$$D_i = 2g_i,$$

$$D_{i+1} = 6f_i l_i + 2g_i.$$

Therefore we can write

$$g_i = \frac{D_i}{2}, \tag{2.4}$$

$$f_i = \frac{D_{i+1} - D_i}{6l_i}. \tag{2.5}$$

Substituting (2.2), (2.4), and (2.5) into (2.3), we get

$$h_i = \frac{y_{i+1} - y_i}{l_i} - \frac{l_i D_{i+1} + 2l_i D_i}{6}. \tag{2.6}$$

The first derivatives of the $(i-1)$th and ith polynomials at x_i must be equal because the two splines surround (x_i, y_i) smoothly without a sharp turn. The first derivative of the ith polynomial is

$$\begin{aligned} y_i' &= 3f_i(x_i - x_i)^2 + 2g_i(x_i - x_i) + h_i \\ &= h_i. \end{aligned}$$

The first derivative of the $(i-1)$th polynomial is

$$\begin{aligned} y_{i-1}' &= 3f_{i-1}(x_i - x_{i-1})^2 + 2g_{i-1}(x_i - x_{i-1}) + h_{i-1} \\ &= 3f_{i-1}l_{i-1}^2 + 2g_{i-1}l_{i-1} + h_{i-1}. \end{aligned}$$

Equating the two values above and substituting f, g, h in terms of D and y, we have

$$\begin{aligned} \frac{y_{i+1} - y_i}{l_i} - \frac{l_i D_{i+1} + 2l_i D_i}{6} &= 3\left(\frac{D_i - D_{i-1}}{6l_{i-1}}\right) l_{i-1}^2 \\ +2\left(\frac{D_{i-1}}{2}\right) l_{i-1} + \frac{y_i - y_{i-1}}{l_{i-1}} - \frac{l_{i-1}D_i + 2l_{i-1}D_{i-1}}{6}&. \end{aligned}$$

Upon simplifying this equation, we get

$$l_{i-1}D_{i-1} + (2l_{i-1} + 2l_i)D_i + l_i D_{i+1} = m_i,$$

where $m_i = 6\left(\frac{y_{i+1} - y_i}{l_i} - \frac{y_i - y_{i-1}}{l_{i-1}}\right)$. This gives $n-1$ equations for the $n+1$ values D_i, $0 \le i \le n$. To uniquely solve the system, we need two boundary conditions $D_0 = 0$ and $D_n = 0$, meaning that the two end splines approach linearity at their utmost points. So, we have the following system of $n-1$ linear equations to be solved:

$$\begin{aligned} 2(l_0 + l_1)D_1 + l_1 D_2 &= m_1, \\ l_{i-1}D_{i-1} + 2(l_{i-1} + l_i)D_i + l_i D_{i+1} &= m_i, \ 2 \le i \le n-2, \\ l_{n-2}D_{n-2} + 2(l_{n-2} + l_{n-1})D_{n-1} &= m_{n-1}. \end{aligned} \tag{2.7}$$

The system is tridiagonal, only the elements directly on, directly above, and directly below the diagonal of the coefficient matrix are nonzero.

2.2 *B-spline Surface Fitting*

The surface fitting problem is to construct a surface which interpolates a given set of points $T_{i,j}$, $1 \le i \le m$, $1 \le j \le n$. The uniform bicubic B-spline surface for the points is a surface consisting of $(m-1) \times (n-1)$ patches $T_{i,j}(u, v)$, $1 \le i \le m-1$, $1 \le j \le n-1$ [2]. Each patch is a bicubic polynomial defined by

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

in which

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

and

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

where $W_{i,j}$, $0 \le i \le m+1$, $0 \le j \le n+1$, 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:

$$\begin{aligned} T_{i,j} = \frac{1}{36}&(W_{i-1,j-1} + 4W_{i,j-1} + W_{i+1,j-1} + 4W_{i-1,j} \\ &+ 16W_{i,j} + 4W_{i+1,j} + W_{i-1,j+1} + 4W_{i,j+1} \\ &+ W_{i+1,j+1}) \quad \text{for} \quad 1 \le i \le m, 1 \le j \le n. \end{aligned} \tag{2.8}$$

Equations in (2.8) form a system of mn equations in $(m+2)(n+2)$ unknowns. In order to completely solve the system, $(m+2)(n+2) - mn = 2(m+n+2)$ additional equations are required, each represents how a boundary point is

interpolated. Usually, the following double-boundary conditions are used:

$$W_{0,j} = W_{1,j}; \quad W_{m+1,j} = W_{m,j}, \quad 1 \le j \le n;$$
$$W_{i,0} = W_{i,1}; \quad W_{i,n+1} = W_{i,n}, 0 \le i \le m+1. \quad (2.9)$$

Putting (2.8) and (2.9) together in matrix form, we have

$$E_{mn\times mn} W^t_{1\times mn} = 36 T^t_{1\times mn}, \quad (2.10)$$

where $W_{1\times mn} = [W_{i,j}]_{1\times mn}$, $T_{1\times mn} = [T_{i,j}]_{1\times mn}$, and

$$E_{mn\times mn} = \begin{pmatrix} 5B & B & 0 & . & . & 0 \\ B & 4B & B & 0 & . & 0 \\ 0 & B & 4B & B & . & 0 \\ . & 0 & . & . & . & . \\ . & . & . & . & . & 0 \\ . & . & 0 & B & 4B & B \\ . & . & . & 0 & B & 5B \end{pmatrix}.$$

According to [Boo78], (2.10) can be rewritten as

$$B_{m\times m} W_{m\times n} B_{n\times n} = 36 T_{m\times n}, \quad (2.11)$$

where

$$B_{n\times n} = \begin{pmatrix} 5 & 1 & 0 & . & . & 0 \\ 1 & 4 & 1 & 0 & . & 0 \\ 0 & 1 & 4 & 1 & . & 0 \\ . & 0 & . & . & . & . \\ . & . & . & . & . & 0 \\ . & . & 0 & 1 & 4 & 1 \\ . & . & . & 0 & 1 & 5 \end{pmatrix}.$$

In the sequel, we attempt to convert (2.11) into tridiagonal systems. The inverse matrix of $B_{n\times n}$ exists because $B_{n\times n}$ is symmetric and positive definite. If we multiply both sides of (2.11) by $B^{-1}_{n\times n}$, we have

$$B_{m\times m} W_{m\times n} = 36 T_{m\times n} B^{-1}_{n\times n}.$$

Let $F_{m\times n} = 6 T_{m\times n} B^{-1}_{n\times n}$, we get

$$B_{m\times m} \begin{pmatrix} W_{1,i} \\ W_{2,i} \\ \vdots \\ W_{m,i} \end{pmatrix} = 6 \begin{pmatrix} H_{1,i} \\ H_{2,i} \\ \vdots \\ H_{m,i} \end{pmatrix} \quad \text{for } i = 1, 2, ..., n. \quad (2.12)$$

We need determine $H_{m\times n}$ before solving (2.12). If we take the transpose on both sides of the equation $H_{m\times n} B_{n\times n} = 6 T_{m\times n}$, we have

$$B_{n\times n} H^t_{m\times n} = 6 T^t_{m\times n},$$

that is,

$$B_{n\times n} \begin{pmatrix} H_{i,1} \\ H_{i,2} \\ \vdots \\ H_{i,n} \end{pmatrix} = 6 \begin{pmatrix} T_{i,1} \\ T_{i,2} \\ \vdots \\ T_{i,n} \end{pmatrix} \quad \text{for } i = 1, 2, ..., m. \quad (2.13)$$

Therefore, solving (2.11) for $W_{m\times n}$ becomes a two-step process, namely, solving the m tridiagonal systems in (2.13) for $H_{m\times n}$ first, and then solving the n tridiagonal systems in (2.12) for $W_{m\times n}$.

3. TRANSFORMATIONS TO CONTINUED FRACTIONS

From the discussions above, we see that solving tridiagonal systems is the major task for the B-spline curve and surface fitting problems.

Consider a tridiagonal system of linear equations:

$$\begin{aligned} d_1 x_1 + e_1 x_2 &= b_1, \\ c_i x_{i-1} + d_i x_i + e_i x_{i+1} &= b_i \quad \text{for } i = 2, 3, \ldots, n-1, \\ c_n x_{n-1} + d_n x_n &= b_n. \end{aligned} \quad (3.1)$$

Assuming $d_1 \neq 0$ in (3.1), we can eliminate x_1 from the second equation to get the new equation

$$d'_2 x_2 + e_2 x_3 = b'_2,$$

where $d'_2 = d_2 - c_2 e_1/d_1$ and $b'_2 = b_2 - c_2 b_1/d_1$. Next, we use the above equation to eliminate x_2 from the third equation (assuming $d'_2 \neq 0$), and get

$$d'_3 x_3 + e_3 x_4 = b'_3,$$

where $d'_3 = d_3 - c_3 e_2/d'_2$ and $b'_3 = b_3 - c_3 b'_2/d'_2$. In general, at stage $(k-1)$, $k = 2, 3, ..., n$, we eliminate x_{k-1} from the kth equation (assuming $d'_{k-1} \neq 0$), and get

$$d'_k x_k + e_k x_{k+1} = b'_k,$$

with

$$d'_k = d_k - c_k e_{k-1}/d'_{k-1}, \quad (3.2)$$

and

$$b'_k = b_k - c_k b'_{k-1}/d'_{k-1}. \quad (3.3)$$

This procedure is the commonly used Gaussian elimination.

During the back-substitution for obtaining x_i, $i = n, n-1, \ldots, 1$, we first have (assuming $d'_n \neq 0$)

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

and then

$$x_k = \frac{b'_k - e_k x_{k+1}}{d'_k} \quad (3.4)$$

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

The solution of the tridiagonal system has now been transformed into three recurrence equations, namely, (3.2), (3.3) and (3.4), for d'_k, b'_k and x_k respectively. In the remainder of this section, these recurrences will be further transformed into suffix computations of continued fractions.

The following form represents a finite continued fraction (CF):

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

Such a representation for CF is typographically cumbersome and can be replaced by the more compact form:

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

Let us define a second-order linear recurrence:

$$\begin{aligned} &f_0 = 1, \quad f_1 = d_1, \\ &f_k = d_k f_{k-1} - c_k e_{k-1} f_{k-2} \quad \text{for } k \geq 2. \end{aligned} \tag{3.5}$$

It is easily seen that $d'_k = f_k / f_{k-1}$ for $k \geq 1$. The computation of d'_k breaks down when $f_{k-1} = 0$. This may occur even if the original coefficient matrix is nonsingular and has nonzero diagonal entries. However, this rarely happens in practice although there is no theoretical guarantee for it [3].

By (3.5), $d'_n = f_n / f_{n-1} = (d_n f_{n-1} - c_n e_{n-1} f_{n-2}) / f_{n-1}$ $= d_n + (-c_n e_{n-1}) / (f_{n-1}/f_{n-2})$. So, d'_n can be expressed as the following CF:

$$d'_n = d_n + \frac{-c_n e_{n-1}}{d_{n-1}+}\frac{-c_{n-1}e_{n-2}}{d_{n-2}+}\cdots\frac{-c_3 e_2}{d_2+}\frac{-c_2 e_1}{d_1}.$$

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 to be a power of 2 without affecting the computation results. The kth suffix value of d'_n is d'_k, i.e.,

$$d'_k = d_k + \frac{-c_k e_{k-1}}{d_{k-1}+}\frac{-c_{k-1}e_{k-2}}{d_{k-2}+}\cdots\frac{-c_3 e_2}{d_2+}\frac{-c_2 e_1}{d_1}.$$

Obviously, d'_k can be converted into a rational form in $O(k)$ time sequentially. Our goal is to speed up the computation for obtaining the rational form of d'_k for all $k = 1, 2, \ldots, n$.

Let the rational form $(r_1 + r_2 d'_k)/(r_3 + r_4 d'_k)$ be represented by $(r_1, r_2, r_3, r_4, d'_k)$. Here d'_k can be thought as a pseudo variable for substitution. Following the substitution scheme we used in [6], d'_8, say, can be decomposed into 8 sub_CFs:

$$\begin{aligned} d'_8 &= d_8 + \frac{-c_8 e_7}{d'_7} = (-c_8 e_7, d_8, 0, 1, d'_7), \\ d'_7 &= d_7 + \frac{-c_7 e_6}{d'_6} = (-c_7 e_6, d_7, 0, 1, d'_6), \\ &\vdots \\ d'_3 &= d_3 + \frac{-c_3 e_2}{d'_2} = (-c_3 e_2, d_3, 0, 1, d'_2), \\ d'_2 &= d_2 + \frac{-c_2 e_1}{d'_1} = (-c_2 e_1, d_2, 0, 1, d'_1), \\ d'_1 &= \frac{d_1}{1 + d'_0} = (d_1, 0, 1, 1, d'_0). \end{aligned}$$

The pseudo variable d'_0 will be replaced by zero after all the substitutions are completed.

Since substituting $d'_i = (s_1 + s_2 d'_j)/(s_3 + s_4 d'_j)$ into $(r_1 + r_2 d'_i)/(r_3 + r_4 d'_i)$ gives $((r_1 s_3 + r_2 s_1) + (r_1 s_4 + r_2 s_2) d'_j)/((r_3 s_3 + r_4 s_1) + (r_3 s_4 + r_4 s_2) d'_j)$, the substitution operation, which uses 8 multiplications and 4 additions, can be defined as:

$$\begin{aligned} &(r_1, r_2, r_3, r_4, d'_i) \circ (s_1, s_2, s_3, s_4, d'_j) \\ =&(r_1 s_3 + r_2 s_1, r_1 s_4 + r_2 s_2, r_3 s_3 + r_4 s_1, \\ &r_3 s_4 + r_4 s_2, d'_j). \end{aligned}$$

Lemma 3.1. *The substitution operation $\circ$ is associative.*

Proof. For any $(r_1, r_2, r_3, r_4, d'_i)$, $(s_1, s_2, s_3, s_4, d'_j)$ and $(t_1, t_2, t_3, t_4, d'_k)$, we have

$$\begin{aligned} &\left[(r_1, r_2, r_3, r_4, d'_i) \circ (s_1, s_2, s_3, s_4, d'_j)\right] \circ (t_1, t_2, t_3, t_4, d'_k) \\ =&(r_1 s_3 + r_2 s_1, r_1 s_4 + r_2 s_2, r_3 s_3 + r_4 s_1, r_3 s_4 + r_4 s_2, d'_j) \\ &\circ (t_1, t_2, t_3, t_4, d'_k) \\ =&(r_1 s_3 t_3 + r_2 s_1 t_3 + r_1 s_4 t_1 + r_2 s_2 t_1, r_1 s_3 t_4 + r_2 s_1 t_4 \\ &+ r_1 s_4 t_2 + r_2 s_2 t_2, r_3 s_3 t_3 + r_4 s_1 t_3 + r_3 s_4 t_1 + r_4 s_2 t_1, \\ &r_3 s_3 t_4 + r_4 s_1 t_4 + r_3 s_4 t_2 + r_4 s_2 t_2, d'_k) \end{aligned}$$

and

$$\begin{aligned} &(r_1, r_2, r_3, r_4, d'_i) \circ \\ &\left[((s_1, s_2, s_3, s_4, d'_j) \circ (t_1, t_2, t_3, t_4, d'_k)\right] \\ =&(r_1, r_2, r_3, r_4, d'_i) \circ \\ &(s_1 t_3 + s_2 t_1, s_1 t_4 + s_2 t_2, s_3 t_3 + s_4 t_1, s_3 t_4 + s_4 t_2, d'_k) \\ =&(r_1 s_3 t_3 + r_2 s_1 t_3 + r_1 s_4 t_1 + r_2 s_2 t_1, r_1 s_3 t_4 + r_2 s_1 t_4 \\ &+ r_1 s_4 t_2 + r_2 s_2 t_2, r_3 s_3 t_3 + r_4 s_1 t_3 + r_3 s_4 t_1 \\ &+ r_4 s_2 t_1, r_3 s_3 t_4 + r_4 s_1 t_4 + r_3 s_4 t_2 + r_4 s_2 t_2, d'_k). \end{aligned}$$

Since both computation sequences give the same rational form, the associativity holds. ∎

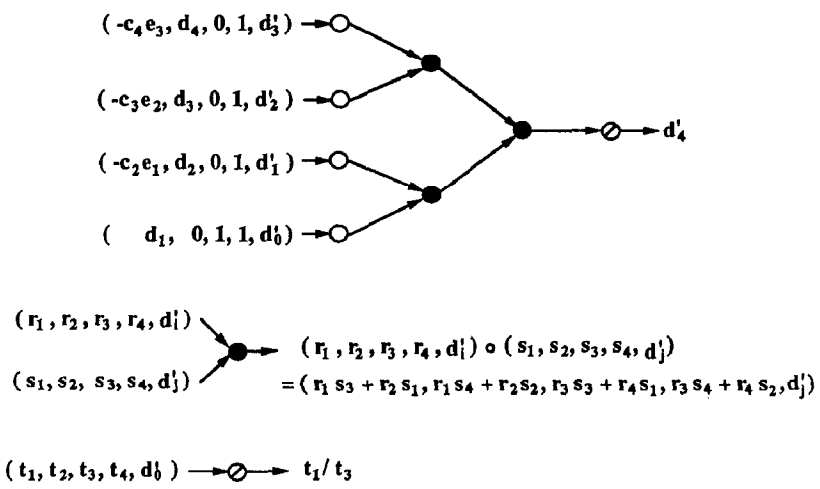

Figure 3.1. A computation tree of d'_4.

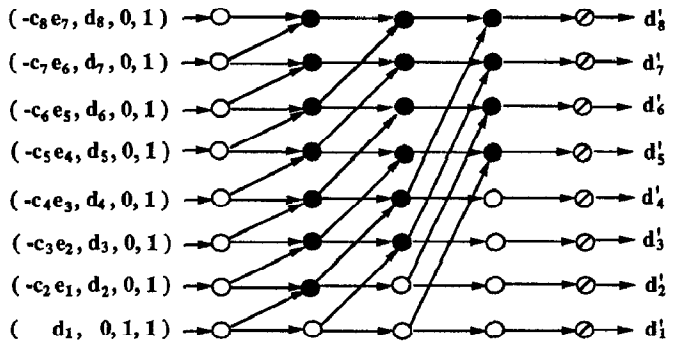

Figure 3.2. A network for computing suffix values of d'_8.

According to Lemma 3.1, $d'_4 = (-c_4e_3, d_4, 0, 1, d'_3) \circ (-c_3e_2, d_3, 0, 1, d'_2) \circ (-c_2e_1, d_2, 0, 1, d'_1) \circ (d_1, 0, 1, 1, d'_0)$ can be computed in any order. Fig. 3.1 shows a possible computation tree of d'_4. The white nodes in the leftmost stage 0 are used for transmitting data only. The black nodes in stages 1 and 2 perform the substitution operation $\circ$. The slash node in stage 3 assigns zero to the pseudo variable d'_0 to obtain the desired result.

If we combine the computation trees of d'_i, $1 \le i \le 8$, we have a network, as shown in Fig. 3.2, for computing all the suffix values of d'_8. The pseudo variables are not real data and hence are omitted there. Initially, the values $-c_ie_{i-1}$, $2 \le i \le 8$, are computed parallely. Each of the white nodes in stage 0 sends out two copies of its input data. Note that 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 processor number is as large as $\Theta(n \log n)$, which is to be greatly reduced to $\Theta(n/\log n)$ in the later sections.

Now for the computation of b'_k in (3.3), the CF for b'_4/b'_3, say, is given by

$$\begin{aligned} X_4 &= \frac{b'_4}{b'_3} \\ &= \frac{b_4 - (c_4/d'_3)b'_3}{b'_3} \\ &= (-c_4/d'_3) + \frac{b_4}{b'_3}. \end{aligned}$$

When x is of the form $r_1 + r_2/r_3$, we define $N(x)$ to be the form $r_1r_3 + r_2$. The sub_CFs for the above CF are

$$\begin{aligned} X_4 &= -c_4/d'_3 + \frac{b_4}{N(X_3)}, \\ X_3 &= -c_3/d'_2 + \frac{b_3}{N(X_2)}, \\ X_2 &= -c_2/d'_1 + \frac{b_2}{N(X_1)}, \end{aligned}$$

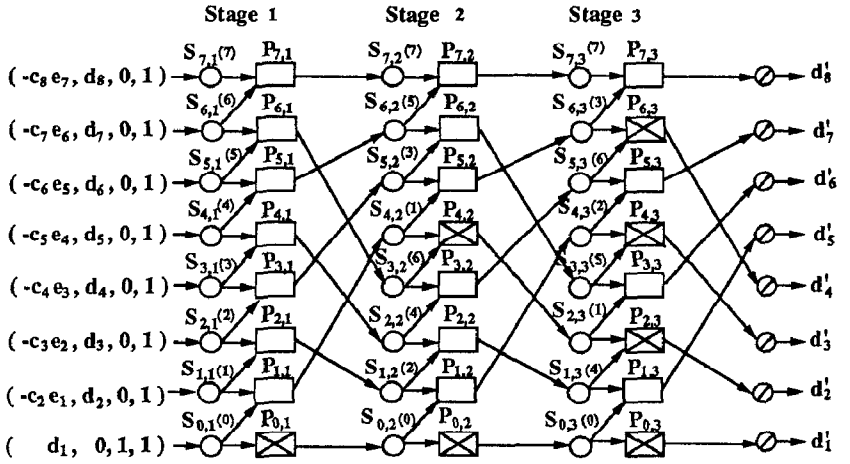

Figure 4.1. A multi-stage unshuffle network.

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

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 is defined as:

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

This operation uses 4 multiplications and 2 additions and is associative also. The problem of solving the recurrence equation in (3.3) becomes the suffix computations of X_n and the network in Fig. 3.2 can be used again.

Similarly, from (3.4), the sub_CFs for , say, x_{n-3}/x_{n-2} is given by

$$Y_{n-3} = -e_{n-3}/d'_{n-3} + \frac{b'_{n-3}/d'_{n-3}}{N(Y_{n-2})},$$
$$Y_{n-2} = -e_{n-2}/d'_{n-2} + \frac{b'_{n-2}/d'_{n-2}}{N(Y_{n-1})},$$
$$Y_{n-1} = -e_{n-1}/d'_{n-1} + \frac{b'_{n-1}/d'_{n-1}}{N(Y_n)},$$
$$Y_n = \frac{b'_n/d'_n}{1 + N(Y_{n+1})}, \qquad (3.6)$$

where $N(Y_{n+1}) = 0$, and $x_i = N(Y_i)$ for all i. So the problem of solving the recurrence equation in (3.4) becomes the suffix computations of Y_1.

4. THE MESH-OF-UNSHUFFLE NETWORK

Recently, Lin and Chung [11] presented a three-phase parallel algorithm on the unshuffle network to solve tridiagonal systems in $O(\log n)$ time. The algorithm reduces the $\Theta(n)$ processors used in [15] and [10] to $\Theta(n/\log n)$. In this section, we shall propose a more general mesh-of-unshuffle network to solve single and multiple tridiagonal systems. The network consists of a mesh of processors in which each row or column of processors form one unshuffle network.

For saving 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.

In Fig. 3.2, there is one white node in the first stage, two in the second stage and four in the third stage. The multi-stage unshuffle network which can simulate that network is depicted in Fig. 4.1, where $S_{i,j}$ and $P_{i,j}$, $0 \le i \le 7, 1 \le j \le 3$, are simple devices and processors respectively. Let $q_0 q_1 \ldots q_{t-1} (t = \log n)$ be the binary representation of i. The output line of $P_{i,j}$ is connected to $S_{q_{t-1} q_0 q_1 \ldots q_{t-2}, j+1}$ to provide the unshuffle (or inverse shuffle) data routing. The simple device $S_{i,j}$ is capable of producing two copies of its input and sends them to $P_{i,j}$ and $P_{i+1,j}$ except that the bottom $S_{11\ldots1,j}$ has only one receiver. For tracing the messages passed in the network, each simple device is labelled by a number inside the parentheses to indicate where its data originate.

Due to the unshuffle connections from the processors to the simple devices, the function of the white nodes in Fig. 3.2 can be controlled by a proper masking mechanism in each stage of the multi-stage unshuffle network. For examples, $P_{0,1}$ in stage 1 is masked by a cross mark to merely transmit the input 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 all the interconnection patterns between the consecutive stages of the multi-stage unshuffle network are identical, we can compress all the stages together to obtain a

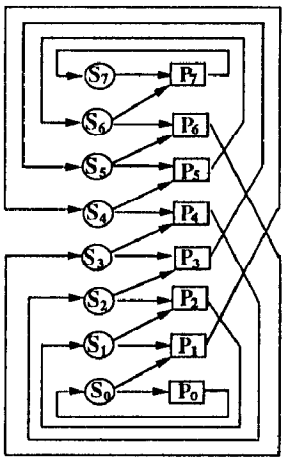

Figure 4.2. A single-stage unshuffle network.

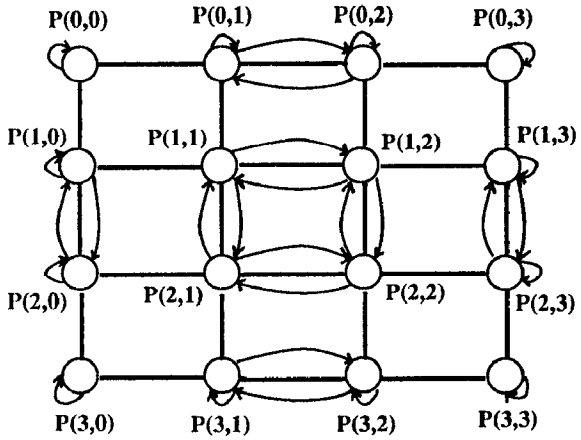

Figure 4.3. The mesh-of-unshuffle network for $h = k = 4$.

single-stage unshuffle network, as shown in Fig. 4.2, which has only n processors. After each iteration of execution, the processors feed the intermediate results back to the simple devices. The masking of the processors in different iterations is exactly what we have described for the corresponding stages in the multi-stage network.

A mesh-of-unshuffle network is proposed next to reduce the number of processors further. The mesh-of-unshuffle network, as illustrated in Fig. 4.3, is a $h \times k$ mesh of processors $P(i,j), 1 \leq i \leq h$ and $1 \leq j \leq k$, where h and k are powers of 2 and $hk \leq n$. The processors in each row or column communicate by the unshuffle routing mechanism. Fig. 4.4 is a simplified version of the network to solve the curve fitting problem. The complete mesh-of-unshuffle network (Fig. 4.3) will be fully utilized later for solving the surface fitting problem.

Let's change the unidirectional connections from the simple devices to the processors to be bidirectional. To each processor $P(r,i)$ in row r, attach a serial memory [12], which consists of a linearly connected array of $n/(hk)$ memory cells $C_r(i,j)$, $0 \leq j \leq n/(hk) - 1$, as depicted in Fig. 4.5 for $n = 32$ and $h = k = 4$. In each execution cycle, the attached memory shifts all its data one position, $P(r,i)$ executes on the data shifted from $C_r(i,0)$ and writes its result to $C_r(i, n/(hk) - 1)$.

According to the value of mask register $M(r,i)$ resided in $P(r,i)$, which is set by the masking mechanism, $P(r,i)$ takes one of the following actions:

(1) $M(r,i) = 0$: $P(r,i)$ transmits the content of its working register $Q(r,i)$ to $S(r,i)$.

(2) $M(r,i) = 1$: $P(r,i)$ receives data from $S(r,i)$ and transmits it to $S(r, q_{t-1}q_0q_1 \ldots q_{t-2})$, where $t = \log k$ and $i = q_0q_1 \ldots q_{t-1}$ in binary.

(3) $M(r,i) = 2$: $P(r,i)$ performs the substitution operation on its two input data and transmits the result to $S(r, q_{t-1}q_0q_1 \ldots q_{t-2})$.

(4) $M(r,i) = 3$: Except $P(r,0)$ which receives input data from $S(r,0)$, $P(r,i)$ receive input data from $S(r,i-1)$. The routing mechanism is forced to stop and the network goes to another phase of computation.

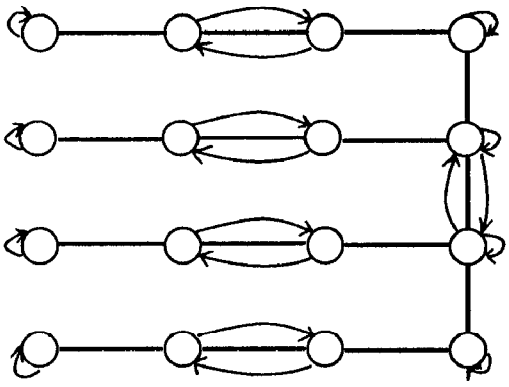

Figure. 4.4. The simplified mesh.

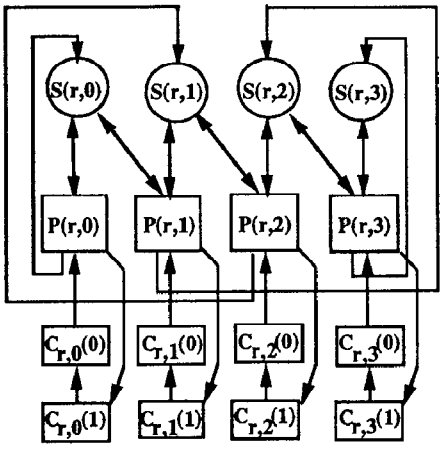

Figure 4.5. The unshuffle network in one row.

The n input data (quadruples) are evenly divided into hk pipes to be stored in the mesh in row-major order. Each serial memory stores a pipe of data. Algorithms 4.1 expresses how the network in each row_r, $0 \leq r \leq h-1$, works. (The algorithm is also applicable to each column of the mesh.)

Algorithm 4.1

Phase 1. (Local computations)

Each $P(r,i)$ sequentially computes $n/(hk)$ suffix values from its corresponding pipe of data and stores them in its serial memory. The register $Q(r,i)$ has the final rational form computed from the pipe of data and transmits it to the register $R(r,i)$ in $S(r,i)$.

Phase 2. (Macro suffix computations)

All the processors work together using the unshuffle routing mechanism for suffix computations. After $\log k$ steps, the working register $R(r,i)$ holds the rational form of $d'_{(i+1)n/(hk)}$, $0 \leq i \leq k-1$.

Phase 3. (Adaptation)

Each $P(r,i)$ except $P(r,0)$ receives the rational form from $S(r,i-1)$, sequentially modifies the suffix values calculated in phase 1 by substituting them into the received rational form.

Both phase 1 and phase 3 need $n/(hk)$ steps. Phase 2 needs $\log k$ steps. So the three-phase unshuffle network in row_r takes $O(n/(hk)+\log k)$ time to finish computing all the suffix values of $d'_{(r+1)n/h}$.

We use d'_{32} $(= d_{32} + \frac{-c_{32}e_{31}}{d_{31}+} \ldots \frac{-c_2e_1}{d_1})$ as an example to illustrate the algorithm. Let $n = 32$ and $h = k = 4$. Initially, the 32 input data (quadruples) for d'_{32} are evenly divided into 16 pipes, each containing 2 data. The cells $C_0(0,0)$ and $C_0(0,1)$ have the data $(d_1, 0, 1, 1)$ and $(-c_2e_1, d_2, 0, 1)$ respectively. The other cells $C_0(i,j)$, $1 \leq i \leq 3, 0 \leq j \leq 1$, have the data $(-c_{2i+j+1}e_{2i+j}, d_{2i+j+1}, 0, 1)$. The job of the three-phase unshuffle network in row_0 is to compute all the suffix values of d'_8. We denote the rational form of $d_i + \frac{-c_ie_{i-1}}{d_{i-1}+} \ldots \frac{-c_2e_1}{d_1+d'_0}$ by $F_{i,1}$ and $d_i + \frac{-c_ie_{i-1}}{d_{i-1}+} \ldots \frac{-c_je_{j-1}}{d'_{j-1}}$ by $F_{i,j}$ for $i \geq j \geq 2$.

Phase 1.

Each individual $P(0,i)$ computes the two forms $F_{2i+1,2i+1}$ and $F_{2i+2,2i+1}$ and shifts them back to the serial memory sequentially. That is, $P(0,0)$ computes all the suffix values of $F_{2,1}$; $P(0,1)$ computes all the suffix values of $F_{4,3}$, and so on.

The cell $C_0(i,j)$ saves $F_{2i+j+1,2i+1}$. After 2 steps in this phase, we have

$$C_0(0,0) = F_{1,1}, \quad C_0(0,1) = F_{2,1},$$
$$C_0(1,0) = F_{3,3}, \quad C_0(1,1) = F_{4,3},$$
$$C_0(2,0) = F_{5,5}, \quad C_0(2,1) = F_{6,5},$$
$$C_0(3,0) = F_{7,7}, \quad C_0(3,1) = F_{8,7},$$

and

$$Q(0,0) = F_{2,1},$$
$$Q(0,1) = F_{4,3},$$
$$Q(0,2) = F_{6,5},$$
$$Q(0,3) = F_{8,7}.$$

Then, for all i, $M(0,i)$ becomes 0 and the content of $Q(0,i)$ is transmitted to $R(0,i)$.

Phase 2.

The following macro suffix computations are controlled by the unshuffle routing mechanism. During the first iteration, $M(0,0) = 1$ and $M(0,i) = 2$ for $i = 1,2,3$. Once the $P(0,i)$'s complete their substitution operations, the contents of $R(0,i)$'s become

$$R(0,0) = F_{2,1},$$
$$R(0,1) = F_{6,5} \circ F_{4,3} = F_{6,3},$$
$$R(0,2) = F_{4,3} \circ F_{21} = F_{4,1},$$
$$R(0,3) = F_{8,7} \circ F_{6,5} = F_{8,5}.$$

During the second iteration, $M(0,i) = 2$ for all i except $i \equiv 0 \bmod 2$ for which $M(0,i) = 1$. Once the P_i's complete their substitution operations again, the contents of $R(0,i)$'s become

$$R(0,0) = F_{2,1},$$
$$R(0,1) = F_{4,1},$$
$$R(0,2) = F_{6,3} \circ F_{2,1} = F_{6,1},$$
$$R(0,3) = F_{8,5} \circ F_{4,1} = F_{8,1}.$$

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

Phase 3.

During the first step of this phase, $M(0,i) = 3$, each $P(0,i)$ except $P(0,0)$ receives input data from $S(0,i-1)$. The contents of $Q(0,i)$'s become

$$Q(0,0) = F_{2,1},$$
$$Q(0,1) = F_{2,1},$$
$$Q(0,2) = F_{4,1},$$
$$Q(0,3) = F_{6,1}.$$

The unshuffle routing mechanism is forced to stop and each $P(0,i)$ sequentially modifies the contents of $C_0(i,j)$, $j = 0,1$, by performing the substitution operation $Q(0,i) \circ C_0(i,j)$. The final results are

$$C_0(0,0) = F_{1,1},$$
$$C_0(0,1) = F_{2,1},$$
$$C_0(1,0) = F_{3,3} \circ F_{2,1} = F_{3,1},$$
$$C_0(1,1) = F_{4,3} \circ F_{2,1} = F_{4,1},$$
$$C_0(2,0) = F_{5,5} \circ F_{4,1} = F_{5,1},$$
$$C_0(2,1) = F_{6,5} \circ F_{4,1} = F_{6,1},$$
$$C_0(3,0) = F_{7,7} \circ F_{6,1} = F_{7,1},$$
$$C_0(3,1) = F_{8,7} \circ F_{6,1} = F_{8,1}.$$

Like row_0, the unshuffle network in row_r, $1 \leq r \leq 3$, computes all the suffix values of $F_{8(r+1),8r+1}$. These suffix values can be viewed as the results of some local computations. Further macro suffix computations and adaptations are needed. The macro suffix computations can be performed in the right-most column of the mesh. After that, each $P(r,3)$ except $P(0,3)$ receives the rational form from $S(r-1,3)$, distributes the rational form to $P(r,j)$, $0 \leq j \leq 2$, to sequentially adapt all the values of the corresponding pipes of data. It takes $O(\log h + n/(hk))$ time to finish these two parts of computation.

5. COST-OPTIMALITY

The performance of a parallel algorithm can be measured by

$$\text{Cost} = \text{Processor Number} \times \text{Execution Time}.$$

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 bounds for B-spline curve and surface fitting problems are $\Omega(n)$ and $\Omega(mn)$ respectively.

As discussed in Section 4, all the suffix values of $\mathcal{A}_n$ can be computed in $O(n/(hk) + \log h + \log k)$ time using hk processors. For the subsequent parts of the 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 the computation to another takes $O(n/(hk))$ time, where $n/(hk)$ is the size of each data pipe. So we have the following lemma.

Lemma 5.1. *A tridiagonal system can be solved in $O(n/(hk) + \log h + \log k)$ time on the mesh-of-shuffle network of $h \times k$ processors.*

Under the criterion of cost, sometimes it is better not to partition the input data too thoroughly. In Lemma 5.1,

if we select $h = k = \Theta(\sqrt{n/\log n})$, we have the following theorem.

Theorem 5.2. *For solving a tridiagonal system in the mesh-of-unshuffle network, at most $\Theta(n/\log n)$ processors are needed to achieve the cost-optimality, and the computing time is $O(\log n)$.*

Let's return to the curve fitting problem.

Theorem 5.3. *Using the mesh-of-unshuffle network with $\Theta(n/\log n)$ processors, the curve fitting problem can be cost-optimally solved in $O(\log n)$ time .*

Proof. For curve fitting, preparing the $\Theta(n)$ data $l_i(= x_{i+1} - x_i)$ and $m_i(= 6(\frac{y_{i+1}-y_i}{l_i} - \frac{y_i-y_{i-1}}{l_{i-1}}))$, $1 \le i \le n$, takes $O(\log n)$ time using $\Theta(n/\log n)$ processors. Once the values of D are obtained, the coefficients f_i, g_i, h_i, and k_i, $1 \le i \le n-1$, in (2.1) can be calculated according to (2.5), (2.4), (2.6), and (2.2) in $O(\log n)$ time by Theorem 5.2. ∎

Now consider the surface fitting problem. We have shown in Section 2 that solving the problem can be transformed into a two-step process, namely, solving the m tridiagonal systems in (2.13) for $H_{m\times n}$ first, and then solving the n tridiagonal systems in (2.12) for the control points $W_{m\times n}$.

We begin with an unshuffle network with $\Theta(n/\log n)$ processors. By taking $h = 1$ and $k = \Theta(n/\log n)$ in Lemma 5.1, one can see that a tridiagonal system in (2.13) can be solved in $O(\log n)$ time on such a row of processors. Since there are m tridiagonal systems, we need use m such unshuffle networks as m rows of processors. If we compact $\Theta(\log m)$ rows into one row, each processor in these $\Theta(m/\log m)$ unshuffle networks has a local memory of size $\Theta(\log n \log m)$. The time to solve (2.13) 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 (2.12) in $O(\log n \log m)$ time.

Theorem 5.4. *Using the mesh-of-unshuffle network with $\Theta\big(mn/(\log m \log n)\big)$ processors, the surface fitting problem can be cost-optimally solved in $O(\log m \log n)$ time.*

Suppose we increase the number of processors from $\Theta\big(mn/(\log m \log n)\big)$ to $\Theta(mn)$ and decrease the size of each local memory from $\Theta(\log m \log n)$ to $\Theta(1)$. By similar arguments, (2.13) can be solved in $O(\log n)$ time and (2.12) can be solved in $O(\log m)$ time.

Theorem 5.5. *Using the mesh-of-unshuffle network with $\Theta(mn)$ processors, the surface fitting problem can be solved in $O(\log m + \log n)$ time.*

6. CONCLUDING REMARKS

The parallel substitution scheme used in this paper was originally proposed in [7] for the parallel computation of general CFs. It allows all the first n prefix values of any CF to be computed in $O(\log n)$ time using $\Theta(n/\log n)$ processors. We also embed a self-substitution concept into the substitution scheme to derive a sequential $O(\log n)$ algorithm to fast compute periodic CFs [6]. Its applications include the approximation of quadratic surd numbers and solving second-order linear recurrence with constant coefficients.

In this paper, we first transform the B-spline curve and surface fitting problems into tridiagonal systems of linear equations. Each tridiagonal system is then transformed into three recurrence equations. The recursive-doubling method [15] is an alternative approach to solve these equations through a rather tricky divide-and-conquer reformulation of the recurrences into ones which use two indices. Our approach is simply doing the recurrences' CF expansion and applying the straightforward substitution concept. The associativity of the substitution not only naturally brings out the parallelism for the computation, but also supplies the idea of three-phase computation to reduce the processor number used in [10,15]. It follows that the related suffix computations are cost-optimally performed on a newly proposed mesh-of-unshuffle network. Due to the associative property used in the suffix computations of CF, our two-dimensional mesh-of-shuffle network can be generalized to higher dimensional networks to fit practical I/O port limitations.

Finally, we conjecture that the surface fitting problem can be solved in $O(\log m + \log n)$ time using $\Theta\big(nm/(\log m + \log n)\big)$ 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] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[4] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[5] F. H. Cheng and A. Goshtasby, A parallel B-spline surface fitting algorithm, *ACM Trans. on Graphics* **8**, 1 ,1989, 41–50.

[6] K. L. Chung, W. C. Chen and F. C. Lin, Fast computation of periodic continued fractions, *Information Processing Letters* **33**, 2, 1989, 67–72.

[7] K. L. Chung, F. C. Lin and W. C. Chen, Parallel computation of continued fractions, submitted for publication.

[8] I. D. Faux and M. J. Pratt, *Computational Geometry*

for Design and Manufacture, Wiley, New York, 1979.

[9] C. F. Gerald and P. O. Wheatley, *Applied Numerical Analysis*, 4th ed., Addison-Wesley, New York, 1989.

[10] S. L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Sta. Comput.* **8**, 3, 1987, 354–392.

[11] F. C. Lin and K. L. Chung, A cost-optimal parallel tridiagonal system solver, to appear in *Parallel Computing*.

[12] R. M. Owens and J. Ja'Ja', Parallel sorting with serial memories, *IEEE Trans. Comput.* **C-34**, 4, 1985, 379–383.

[13] T. Pavlidis, Curve fitting as a pattern recognition problem, in *Proc. 1982 Int. Conf. on Pattern Recognition*, IEEE Computer Society Press, Silver Spring, Md., 1982, 853–859.

[14] B. Pham, Conic B-spline for curve fitting: a unifying approach, *Comput. Graphics Image Process.* **45**, 1, 1989, 117–125.

[15] H. S. Stone, Parallel tridiagonal equation solvers, *ACM Trans. Math. Software* **1**, 4, 1975, 289–307.