Efficient Matrix Chain Ordering in Polylog Time* (Extended Abstract)

Phillip G. Bradford   Gregory J. E. Rawlins   Gregory E. Shannon
Department of Computer Science
Indiana University
215 Lindley Hall
Bloomington, Indiana 47405
{ bradford, rawlins, shannon }@cs.indiana.edu

Abstract

This paper gives an $O(\lg^3 n)$-time and $n/\lg n$ processor algorithm for solving the matrix chain ordering problem and for finding optimal triangulations of a convex polygon on the Common CRCW PRAM model. This algorithm works by finding shortest paths in special digraphs modeling dynamic programming tables. Also, a key part of the algorithm is improved by computing row minima of a totally monotone matrix, this lets the algorithm run in $O(\lg^2 n)$ time with n processors on the EREW PRAM or even $O(\lg^2 n \lg\lg n)$ time with $n/\lg\lg n$ processors on the CRCW PRAM.

1 Introduction

The *matrix chain ordering problem* (MCOP) is to find the cheapest way to multiply a chain of n matrices, where the matrices are pairwise compatible but of varying dimensions [2, 9]. The MCOP is often the focus of dynamic programming research and pedagogy because of its amenability to an elementary dynamic programming solution. Until recently none of this work has given an efficient (linear-processor) polylogarithmic time algorithm for the MCOP.

This paper is an improvement of Ramanan's $O(\lg^4 n)$ time and n processor solution [17, 18] as well as the $O(\lg^4 n)$ time and $n/\lg n$ processor solution due to Bradford, Rawlins and Shannon [7] (which is a correction and update of [5]). Our approach follows [5, 6], recasting the MCOP as a shortest path problem in a graph modeling a dynamic programming table. This graph has $O(n^2)$ nodes and with an all-pairs shortest paths algorithm finding a shortest path in this graph results in a $n^6/\lg n$ processor MCOP algorithm which is just like Rytter's [19]. Reducing the number of nodes to $O(n)$ using a tree decomposition and applying an all-pairs shortest path algorithm gives an $n^3/\lg n$ processor and polylog time algorithm.

In this paper, we convert the successive applications of the brute force all-pairs shortest paths algorithm to successive applications of parallel partial prefix and binary search algorithms. As in the $n^3/\lg n$-processor algorithm, the applications of the prefix and binary search algorithms are controlled by a rake-compress paradigm operating on a tree based decomposition of the original graph. All of this results in a polylog-time ($O(\lg^3 n)$) and linear-processor ($n/\lg n$) parallel algorithm for MCOP on the Common CRCW PRAM.

Elementary dynamic programming algorithms sequentially solve the matrix chain ordering problem in $O(n^3)$ time [2, 9]. However, the best serial solution of the MCOP is Hu and Shing's $O(n \lg n)$ algorithm [14, 15]. Valiant et al. [20] gave algorithms taking $\Theta(\lg^2 n)$ time and n^9 processors to solve problems such as the MCOP. Rytter [19] gave more efficient parallel algorithms for a similar class of optimization problems costing $O(\lg^2 n)$ time with $n^6/\lg n$ processors. Huang et al. [13] and Galil and Park [12] gave algorithms that can solve the MCOP in $O(\lg^2 n)$ time using $n^6/\lg^5 n$ and $n^6/\lg^6 n$ processors respectively. In [6], an algorithm was given that takes $O(\lg^3 n)$ time and $n^3/\lg n$ processors and Czumaj, in [10], gave an algorithm that takes $O(\lg^3 n)$ time and $n^2/\lg^3 n$ processors. In [17] and [18] Ramanan independently gives an $O(\lg^4 n)$ time and n processor algorithm for solving the MCOP on the CREW PRAM. Ramanan gives an extended abstract of his n processor and $O(\lg^4 n)$ time CREW PRAM algorithm for solving the MCOP in [17], his full version is in [18]. A version of our

*Please address correspondence to Phil Bradford.

$O(\lg^4 n)$ time and $n/\lg n$ processor algorithm is in [7].

Section 2 shows how to interpret the MCOP as a shortest path problem as in [6] and summarizes this $n^3/\lg n$ processor algorithm. Section 3 isolates this algorithm's $n^3/\lg n$ processor bottlenecks which are due to an all-pairs shortest paths algorithm. Section 4 replaces this shortest path algorithm with parallel prefix and an all-pairs *comparison* algorithm. Finally, section 5 replaces the all-pairs comparison algorithm with applications of parallel prefix and binary search, then with the row minima problem on a totally monotone matrix.

2 An $O(\lg^3 n)$ Time and $n^3/\lg n$ Processor MCOP Algorithm

This section briefly reviews the polylog time and $n^3/\lg n$ processor MCOP Algorithm from [6].

Let T be an $n \times n$ dynamic programming table for the matrix chain ordering problem, it has entries $T[i,k]$ representing the cheapest cost of the matrix products $M_i \bullet \cdots \bullet M_k$. For any such T there is a graph D_n where the cost of a shortest path to node (i,k), denoted $sp(i,k)$, is the same as the final value of $T[i,k]$. Given a chain of n matrices finding a shortest path from $(0,0)$ to $(1,n)$ in D_n solves the MCOP [6].

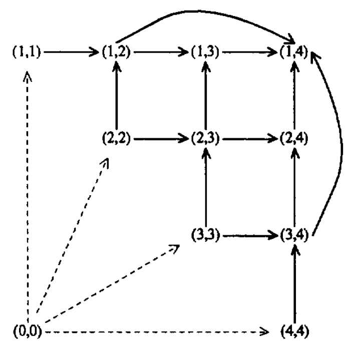

Figure 1: The Weighted Graph D_4

D_n has vertices,

$$\{(i,j) : 1 \le i \le j \le n\} \cup \{(0,0)\}$$

and edges,

$$\{(i,j) \rightarrow (i,j+1) : 1 \le i \le j < n\} \cup$$
$$\{(i,j) \uparrow (i-1,j) : 1 < i \le j \le n\} \cup$$
$$\{(0,0) \nearrow (i,i) : 1 \le i \le n\}$$

known as *unit* edges, together with the edges,

$$\{(i,j) \Longrightarrow (i,t) : 1 < i < j < t \le n\} \cup$$
$$\{(s,t) \Uparrow (i,t) : 1 \le i < s < t \le n\}$$

known as *jumpers*, see the jumper from $(1,2)$ to $(1,4)$ in figure 1. The unit edge $(i,j) \rightarrow (i,j+1)$ represents the product $(M_i \bullet \cdots \bullet M_j) \bullet M_{j+1}$ and weighs $f(i,j,j+1) = w_i w_{j+1} w_{j+2}$ which is taken as the cost of multiplying a $w_i \times w_{j+1}$ matrix and a $w_{j+1} \times w_{j+2}$ matrix. Similarly, the unit edge $(i,j) \uparrow (i-1,j)$ represents the product $M_{i-1} \bullet (M_i \bullet \cdots \bullet M_j)$ and costs $f(i-1,i-1,j) = w_{i-1} w_i w_{j+1}$. A shortest path to (i,k) through the jumpers $(i,j) \Longrightarrow (i,k)$ and $(j+1,k) \Uparrow (i,k)$ both represent the product $(M_i \bullet \cdots \bullet M_j) \bullet (M_{j+1} \bullet \cdots \bullet M_k)$ and these jumpers weigh $sp(j+1,k) + f(i,j,k)$ and $sp(i,j) + f(i,j,k)$ respectively. Where $f(i,j,k) = w_i w_{j+1} w_{k+1}$ and $sp(j+1,k)$ is the cost of a shortest path to node $(j+1,k)$. See figure 1.

Theorem 1 (Duality Theorem [6]) If a shortest path from $(0,0)$ to (i,k) contains the jumper $(i,j) \Longrightarrow (i,k)$ then there is a *dual* shortest path containing the jumper $(j+1,k) \Uparrow (i,k)$.

A D_n graph has $\Theta(n^2)$ nodes, now we sketch some techniques that show we can solve the MCOP with $O(n)$ nodes of a D_n graph.

Given an associative product with the level of each parenthesis known, then for each parenthesis find its matching parenthesis by solving the all nearest smaller value (ANSV) problem [4]: Given $w_1, w_2, \ldots, w_n$, for each w find the largest j such that $1 \le j < i$, and smallest k where $i < k \le n$, so that $w_j < w_i$ and $w_k < w_i$ if such values exist. Let's call this pair of indices, if they exist, an ANSV *match*. In D_n, a *critical node* is (i,k) such that $[w_i, w_{k+1}]$ is an ANSV match.

By solving the ANSV problem we can compute all critical nodes of D_n. Figure 2 shows a weight list where dashed lines representing four key ANSV matches. The four corresponding critical nodes are circled in D_n.

In our nomenclature, [4] shows on a common CRCW PRAM and in [8, 16] on a EREW PRAM:

Theorem 2 Computing all critical nodes costs $O(\lg n)$ time with $n/\lg n$ processors.

Two critical nodes on the same diagonal are *compatible* if no vertices other than $(0,0)$ can reach both of them by a unit path. Since a path of critical nodes represents a parenthesization, all critical nodes are compatible. Also, D_n has at most $n-1$ critical nodes

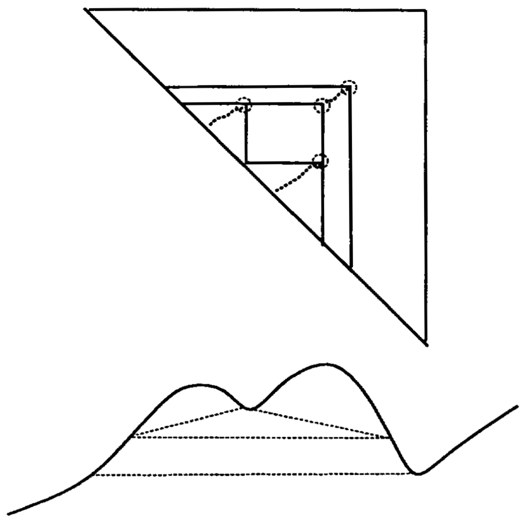

Figure 2: Two Leaf Subgraphs Inside an Band Subgraph with Critical Nodes Shown

and there is at least one path from $(0,0)$ to $(1,n)$ that includes all critical nodes [6].

All vertices and edges that can reach (i,t) by a unit path form the subgraph $D(i,t)$. If $D(i,j)$ has a monotonic weight list $w_i,\ldots,w_{j+1}$, then $D(i,j)$ is monotonic. A *band* is a canonical subgraph $D^{(i,t)}_{(j,k)}$ which contains the maximal unit edge-connected path of critical nodes beginning at critical node (j,k) and terminating at critical node (i,t) with the vertex set

$$\{V[D(i,t)] - V[D(j+1,k-1)]\} \cup \{(0,0)\}$$

and associated edges. A canonical subgraph of the form $D^{(i,t)}_{(j,j+1)}$, is a *leaf* canonical subgraph and is written $D^{(i,t)}$ and has the same nodes and edges as $D(i,t)$. Figure 2, shows two leaf subgraphs nested inside of a band subgraph. From here on p denotes the path of critical nodes in band or leaf canonical subgraphs.

If $D(i,u)$ has a monotone list of weights $w_i \le w_{i+1} \le \cdots \le w_{u+1}$, then a shortest path from $(0,0)$ to (i,u) is the straight unit path $(0,0) \nearrow (i,i) \to (i,i+1) \to \cdots \to (i,u)$, that costs $w_i \sum_{j=i+1}^{u} w_j w_{j+1}$. On the other hand, if $D(i,t)$ has no critical nodes, then its associated weight list is monotonic. As in [14, 15, 6] let $\|w_i : w_k\| = \sum_{j=i}^{k-1} w_j w_{j+1}$, which is easily computable using differences of parallel partial prefixes $\|w_1 : w_i\|$ for $2 \le i \le n+1$. This is useful since the unit path $(i,j) \to \cdots \to (i,k)$ costs

$$w_i\|w_{j+1} : w_{k+1}\| = w_i(\|w_1 : w_{k+1}\| - \|w_1 : w_{j+1}\|).$$

Suppose (j,k) and (i,t) are two critical nodes in a canonical graph such that from (j,k) we can reach (i,t) by a unit path, that is $i \le j \le k \le t$, then the *angular* paths of (j,k) and (i,t) are, (see figure 3)

$$(j,k) \Uparrow (i,k) \to \cdots \to (i,t)$$

and

$$(j,k) \Longrightarrow (j,t) \uparrow \cdots \uparrow (i,t).$$

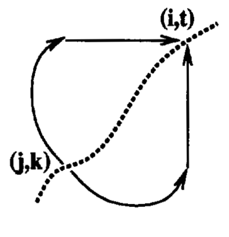

Figure 3: Two Angular Paths

Theorem 3 ([6]) In a canonical subgraph the shortest path between any two critical nodes that contains no other critical nodes is an angular path or edge.

In addition, any shortest path not including critical nodes is a straight path of unit edges. Thus, any shortest path to a critical node that contains no other critical nodes is a straight path of unit edges [6].

Now a polylog time algorithm for finding shortest paths to all critical nodes in $D^{(1,m)}$ leaf graphs is given. This algorithm takes $O(\lg^2 m)$ time and uses $m^3/\lg m$ processors.

First compute the parallel partial prefixes $\|w_1 : w_i\|$ for $2 \le i \le m+1$. Now find all critical nodes, then in constant time using m processors compute the costs of all of the unit paths to nodes in p. Next compute the cost of the $O(m^2)$ angular paths in constant time with m^2 processors. Finally, compute the shortest path to each node in p by treating every angular path as an edge and applying a parallel all-pairs shortest path algorithm.

In D_n, a *canonical tree* joins all of the canonical subgraphs. Initially, for every leaf $D^{(i,j)}$ the critical node (i,j) is the tree leaf $\overline{(i,j)}$. *Internal* tree nodes are either isolated critical nodes or $\overline{(i,t)}$ and $\overline{(j,k)}$ in the band $D^{(i,t)}_{(j,k)}$. Tree edges are straight unit paths connecting tree nodes and jumpers may reduce the cost of tree edges [6].

Given an instance of the MCOP with the weight list $l_1 = w_1, w_2, \ldots w_{n+1}$, then cyclically rotating it getting l_2 and finding an optimal parenthesization for l_2 gives an optimal solution to the original instance of the MCOP with l_1, [14, 11]. So in the *rest of this* *paper* let w_1 denote the smallest weight in any weight list.

A result of Hu and Shing [14] gives the following.

Corollary 1 (Atomicity Corollary [6])
Given a weight list $w_1, \ldots, w_{n+1}$, with the three smallest weights w_1, w_{j+1}, and w_{k+1}, such that $1 < j < k-1$, then the critical nodes $(1,j)$ and $(1,k)$ are in a shortest path from $(0,0)$ to $(1,n)$ in D_n.

For this Corollary to work, it is central that if w_1, w_{j+1}, and w_{k+1} are the three smallest weights, then $j+1 > 2$ and $k > j+1$. This means generally Corollary 1 cannot be applied in a canonical subgraph. But, Corollary 1 can be used to break D_n into a tree of canonical graphs.

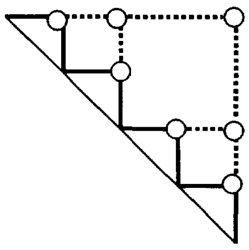

Figure 4: A Tree of Canonical Leaf Graphs, the Circles Denote Internal Tree Nodes

If D_n has fewer than $n-1$ critical nodes, then D_n may have disconnected canonical trees and monotone subgraphs. But, there is at least one path joining these subtrees and at the same time we can discount the monotone subgraphs [6]. The relationships canonical graphs may have all follow directly from critical nodes and tree nodes.

The tree edge $(i,j) \rightarrow \cdots \rightarrow (i,v)$ along row i initially costs $w_i||w_{j+1} : w_{v+1}||$ where $w_i < w_{v+1} < w_{j+1}$, are the three smallest weights in $D(i,v)$. Let $\overline{p}$ denote a shortest path of critical nodes in $D(j+1,v)$ from $(j+1,v)$ back to $(0,0)$. ***Edge minimizing*** the unit path along the i^{th} row to the critical node (i,v) is done as follows, first let $L = w_i||w_{i+1} : w_{v+1}||$ and $W((i,k) \Longrightarrow (i,u)) = sp(k+1,u) + f(i,k,u)$ and $(k+1,u) \in V[D(j+1,v)]$, then compute

$$A[i,v] \quad = \quad \min_{\forall (k+1,u) \in V[\overline{p}]} \{ \, L, \; w_i||w_{i+1} : w_{v+1}|| - w_i||w_{k+1} : w_{u+1}|| + W((i,k) \Longrightarrow (i,u)) \, \}.$$

Since the three smallest weights in $D(i,v)$ are $w_i < w_{v+1} < w_{j+1}$, by Corollary 1 the cheapest cost to critical node (i,v) is now in $A[i,v]$.

Theorem 4 ([6]) When edge minimizing a tree edge $(i,j) \rightarrow \cdots \rightarrow (i,v)$ in a canonical subgraph we only have to consider jumpers $(i,k) \Longrightarrow (i,t)$ such that $(k+1,t) \in V[\overline{p}]$.

Theorem 4 allows the use of tree contraction techniques to be applied where the raking operation is edge minimization and band merging.

The critical node (i,u) in the band $D^{(i,u)}_{(j,k)}$ is the *front* critical node. In general Theorem 4, holds when $\overline{p}$ is a shortest path through a band from the front critical node back to $(0,0)$. Also, Theorem 4 holds for leaves in the canonical tree that, after raking, have become conglomerates of other leaves, bands, and isolated critical nodes. Here, jumpers derived from critical nodes in different subtrees are independent so we can minimize tree edges with them simultaneously.

Assume that all critical nodes (i,j) in tree leaves have the minimum cost back to $(0,0)$ stored in $sp(i,j)$. Compute these values using an all-pairs shortest path parallel algorithm. There is an ordering of the leaves that prevents the simultaneous raking of two adjacent leaves by Corollary 1.

Given two nested bands, say $D^{(i,v)}_{(j,u)}$ is nested around $D^{(k,t)}_{(r,s)}$, that is $j \leq k < t \leq u$. Without loss, say any trees between $D^{(i,v)}_{(j,u)}$ and $D^{(k,t)}_{(r,s)}$ have been contracted, then joining these bands costs $O(\lg^2 n)$ time with $n^3/\lg n$ processors using edge minimizing and band merging with an all pairs min-path algorithm [6].

3 The Structure of Shortest Paths in Canonical Subgraphs

This section highlights the $n^3/\lg n$ processor bottlenecks of the algorithm in section 2.

For each critical node in a D_n graph there are two pointers called ***front-ptr*** and ***back-ptr***. The costs associated with ***front-ptr*** are ***cost-of-front-ptr*** and ***cost-to-front***; the cost associated with ***back-ptr*** is ***cost-to-back***. These costs have the obvious values associated with them. Initially, the pointers connect critical nodes and tree edges in the canonical tree.

There are three parts in the algorithm of [6] that use $n^3/\lg n$ processors. All other parts of that algorithm use a total of $n/\lg n$ processors and $O(\lg n)$ time. These three bottlenecks are: Finding minimal paths to critical nodes in $D^{(1,m)}$ graphs, see figure 5a. Merging two bands, see figure 5b. Merging two bands that have contracted canonical trees between them. In figure 5c, contracted trees A and B are used to edge minimize the unit paths marked by **"Min-A"** and **"Min-B."**

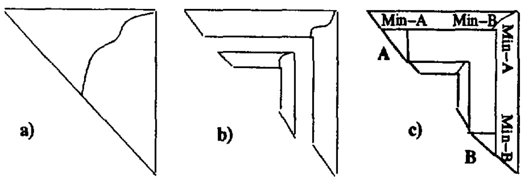

Figure 5: Bottlenecks 1, 2 and 3 for the $n^3/\lg n$ Processor Algorithm

Let $\overline{p}$ denote a minimal path in any such band and $\overline{p}$ is made up of ***super*** critical nodes. Such a minimal path must be critical nodes interspersed with angular paths, by Theorem 3. A $D^{(1,m)}$ graph can be broken into nested bands. Therefore by starting with constant width bands in a $D^{(1,m)}$ graph and recursive double the band merging algorithm while performing tree collapsing we can treat bottlenecks 2 and 3 as a special case of bottleneck 1. Hence, from here on band merging is the focus.

Any angular edge is equivalent to some jumper in a straight line unit path by Theorem 1. Since the following measures the contribution of jumpers to minimal paths along straight unit paths we can use it to get the contribution (if any) of angular edges to minimal paths. Take a node $(s,t) \in V[\overline{p}]$, where $sp(s,t)$ is the minimal path with respect to a band, then in row i compare the contribution of the jumper $(i, s-1) \Longrightarrow (i,t)$ with the associated unit path $(i, s-1) \rightarrow \cdots \rightarrow (i,t)$.

Given $(s,t) \in V[\overline{p}]$, take row i above p with the jumper $(i, s-1) \Longrightarrow (i,t)$, define

$$\begin{aligned}\Delta_i(s,t) &= w_i \| w_s : w_{t+1} \| - \\ &\quad [\, sp(s,t) + f(i, s-1, t) \,]\end{aligned}$$

So, if $\Delta_i(s,t) > 0$, then the jumper $(i, s-1) \Longrightarrow (i,t)$ provides a *cheaper* path along row i than the unit path $(i, s-1) \rightarrow \cdots \rightarrow (i,t)$. Given two nested bands, we want to find the jumpers that make the unit paths in the outer band the cheapest so we can compute the shortest path from the front critical node of the outer band back to $(0,0)$.

4 An $O(\lg^2 n)$ Time and $n^2/\lg n$ Processor MCOP Algorithm

This section uses an induction invariant to break through the bottlenecks of the last section.

In figure 6 is the induction invariant for merging two neighboring bands. Let $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ be two such bands with paths of critical nodes labeled $p^{(i,v)}_{(j,t)}$ and $p^{(j,t)}_{(k,s)}$ respectively. Let $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$, be made by two linked lists of ***back-ptrs*** along super critical nodes and shortest paths forward form all critical nodes by linked lists of trees of ***front-ptrs***.

1.	All critical nodes in both bands have their ***front-ptrs*** in trees of shortest paths that eventually go to super critical nodes which go to the front critical nodes of their respective bands.
2.	In the two bands shortest paths back to $(0,0)$ of super critical nodes are known. These shortest paths of super critical nodes are made of linked lists of ***back-ptrs*** from the front critical nodes of each band back through their respective bands to $(0,0)$.

Figure 6: Inductive Invariant for Band Merging

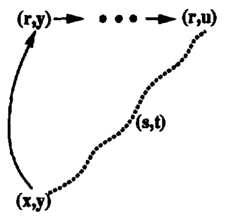

Figure 7: $(s,t) \notin V[\overline{p}]$ and the Angular Edge $(x,y) \Uparrow (r,y) \rightarrow \cdots \rightarrow (r,u)$ in $\overline{p}$

Theorem 5 Given a critical node (s,t) between the ***super*** critical node (x,y) and critical node (r,u) and say $i < r < s < x$ and row i is above row r, that is $w_i < w_r$, where rows i and r are above p then

if $(r, x-1) \Longrightarrow (r,y)$ makes row r cheaper than $(r, s-1) \Longrightarrow (r,t)$ does
then $(i, x-1) \Longrightarrow (i,y)$ makes row i cheaper than $(i, s-1) \Longrightarrow (i,t)$ does

See figures 7 and 8 to illustrate Theorem 5.

Two angular edges above p, say $(x,y) \Uparrow (r,y) \rightarrow \cdots \rightarrow (r,u)$ and $(i,j) \Uparrow (s,j) \rightarrow \cdots \rightarrow (s,t)$, are *compatible* if they don't cross each other. A symmetric definition holds for angular paths below p. Theorem 6 shows that ***when merging*** two bands and computing shortest paths forward, only compatible angular edges need to be considered. Figure 9 shows two conflicting angular paths.

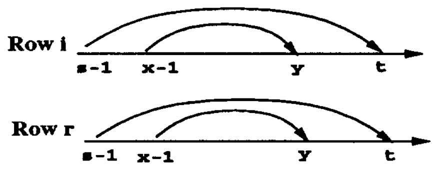

Figure 8: Two Jumpers in Different Rows

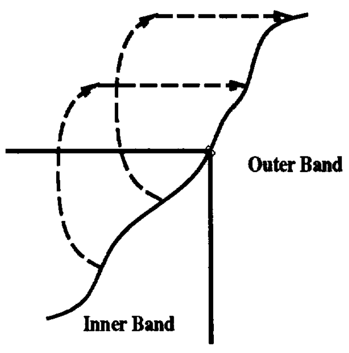

Figure 9: Conflicting Angular Paths *Between* Two Bands Being Merged

Theorem 6 (Main Theorem)
In merging two nested bands computing shortest paths forward from super critical nodes of the inner band, we can consider only compatibly nested angular edges.

It is important to notice that Theorem 6 only holds when merging two bands. When merging two bands and computing shortest paths forward between them all angular edges are compatible. Hence the front pointers form a tree which is consistent with the linked list of back pointers.

Theorem 7 Given two *super* critical nodes (v, z) and (x, y) where $r < s < x < v$ and $w_r < w_s$ such that rows s and r are above p, then we have

if $(r, x-1) \Longrightarrow (r, y)$ makes row r cheaper
than $(r, v-1) \Longrightarrow (r, z)$ does
then $(s, x-1) \Longrightarrow (s, y)$ makes row s cheaper
than $(s, v-1) \Longrightarrow (s, z)$ does

See figure 10 to illustrate Theorem 7.

While merging $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ to form $\overline{p}^{(i,v)}_{(k,s)}$ the next lemma shows that we only need minimal path values backwards to (k, s) from super critical nodes. Hence, the *back-ptrs* can form a linked list and compute the *cost-to-back* weights by a pointer jumping partial parallel prefix.

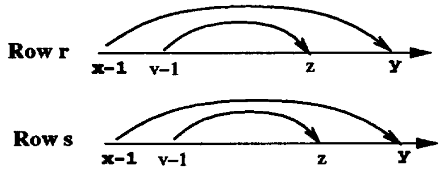

Figure 10: Two Jumpers in Different Rows

Lemma 1 Take $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$ in $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ respectively, for *non*-super critical nodes, say $(u, z) \in V[p^{(i,v)}_{(j,t)}]$ and $(u, z) \notin V[\overline{p}^{(i,v)}_{(j,t)}]$, then we don't need minimal paths *back* to (k, s).

Every critical node has a shortest path to the front critical node of the band it is in that eventually goes through a super critical node. For every critical node we want to maintain a shortest path forward during band merging. This is because some angular path from some future inner band may terminate at any critical node. Therefore, after finding each super critical node's minimal cost to the front of the outer band then do a tree partial prefix sum from the critical nodes to the super critical node so we know their minimal paths to the front of the outer band.

Theorem 8 In $\overline{p}^{(i,v)}_{(j,t)}$ the front pointers form a tree and the back pointers form a linked list.

Theorem 8 shows the inductive invariant holds given the appropriate data structures.

The algorithm in figure 11 costs $O(\lg n)$ time and $n^2/\lg n$ processors for merging two bands. Adding the cost of recursive doubling and tree contraction which together can be done in $O(\lg n)$ time making the total cost $O(\lg^2 n)$ time and $n^2/\lg n$ processors. The for loops labeled by 1 in figure 11 find the shortest path from the front critical node of the outer band back to $(0, 0)$ through both bands. This is the only part of the algorithm that uses $n^2/\lg n$ processors. And the base case for the recursive doubling is established by breaking the canonical subgraphs into bands of constant width.

5 Efficient Polylog Time MCOP Algorithms

This section reduces the processor complexity of the band merging algorithm of section 4 with a parallel divide and conquer form of binary search.

In a band, if $(i,x) \Longrightarrow (i,y)$ is minimal in row i and $(s,x) \Longrightarrow (s,y)$ is minimal in row s, then $(r,x) \Longrightarrow (r,y)$ is minimal in row r for all $r, i \leq r \leq s$. A proof of this follows from Theorems 5 and 7.

Theorem 9 gives a parallel divide and conquer binary search algorithm that finds the jumpers that minimize each unit path edge in a canonical graph.

Theorem 9 Suppose that r is the row in the outer band such that the dual of $(r,x) \Longrightarrow (r,y)$ gives a shortest path forward from the super critical node $(x-1,y)$ of the inner band to the front node of the outer band then,

- It is sufficient to consider only larger nested jumpers in any row s below row r, that is $w_s > w_r$,
- It is sufficient to consider only smaller nested jumpers in any row i above row r, that is $w_i < w_r$.

A proof of this theorem comes directly from Theorems 5, 6 and 7, also see figures 8 and 10.

This next algorithm replaces the two **for** loops in step 1 in the algorithm of figure 11. Say each band has m critical nodes, the next procedure finds the jumpers that edge minimize the unit paths in the outer band. As before, begin assuming the inductive invariant.

1. Find the middle super critical node in the inner band, say $(x-1,y)$.
2. Using $m/\lg m$ processors and in $O(\lg m)$ time find a shortest path forward from $(x-1,y)$ to the front of the outer band. Say this shortest path forward is from the super critical node $(x-1,y)$ and has an angular edge between the two bands that terminates in row r. Let $X = (r,x) \Longrightarrow (r,y)$.
3. Split the jumpers into two sets,
   (a) those smaller than or equal to X call them S, these are nested *inside* $X = (r,x) \Longrightarrow (r,y)$
   (b) those larger than or equal to X call them L, these are nested *around* $X = (r,x) \Longrightarrow (r,y)$
4. Do the following two steps in parallel:
   (a) assign $|S|$ processors to rows r up through 1 and recurse starting at step 1 with S
   (b) assign $|L|$ processors to rows r down through m and recurse starting at step 1 with L

This algorithm finds what jumpers minimize what unit paths. It takes $O(\lg^2 n)$ time and uses $n/\lg n$ processors in the worst case. Considering the cost of the recursive doubling and the tree contraction, gives the $O(\lg^3 n)$ time and $n/\lg n$ processor matrix chain ordering algorithm.

We also mention that with a little more work Theorem 9 shows that the problem of merging two bands is reducible to the problem of finding row minima in a totally monotone matrix. Hence, by Aggarwal and Park [1] and by Atallah and Kosaraju [3] we can run our algorithm in $O(\lg^2 n \lg\lg n)$ time using $n/\lg\lg n$ processors on the CRCW model or in $O(\lg^2 n)$ time using n processors on the EREW PRAM respectively.

Take two adjacent nested bands, say $D^{(i,v)}_{(j,t)}$ nested around $D^{(j,t)}_{(k,s)}$, such that the inductive invariant holds for each band.

```
1. for all super critical nodes $(x,y) \in V[\overline{p}^{(j,t)}_{(k,s)}]$
     in parallel do
       for all angular edges from $(x,y)$ to all
         $(u,z) \in V[p^{(i,v)}_{(j,t)}]$ in parallel do
           Find the angular edge connecting the bands
           from $(x,y)$ that gives a shortest path
           all the way to $(i,v)$, for each of these new
           edges compute cost-of-front-ptrs.
           For the super critical nodes in $\overline{p}^{(j,t)}_{(k,s)}$
           put the angular edge that gives them a
           shortest path forward to $(i,v)$ in $M$.
2. for all angular edges in $M$ in parallel do
       Find the shortest path $N$ from $(i,v)$ back to
       $(0,0)$ through $D^{(i,v)}_{(k,s)}$.
   for all nodes in the path $N$ in parallel do
       Using pointer jumping build the back-ptrs
       giving any new super critical nodes
       and compute the values of cost-to-back for
       each new super critical node.
3. for all non-super critical nodes in
     $p^{(j,t)}_{(k,s)}$ in parallel do
       Using pointer jumping expand the tree of
       front-ptrs through the new angular
       edges in $M$ and their minimal values
       to $(i,v)$. This gives trees joined by a linked
       list through the super critical nodes.
       Now find the shortest path to $(i,v)$ for
       all non-super critical nodes in $p^{(j,t)}_{(k,s)}$ by
       computing a partial prefix in a rooted tree.
       Also compute all of the new cost-to-front
       values using a parallel partial prefix.
```

Figure 11: Algorithm for Merging Two Bands

Acknowledgments

Conversations with Alok Aggarwal, Artur Czumaj, Larry Larmore and Kunsoo Park were helpful and enjoyable.

References

[1] A. Aggarwal and J. Park: "Parallel Searching Multidimensional Monotone Arrays," to appear in the *Journal of Algorithms* and parts of 29th FOCS, 497-512, 1988.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[3] M. J. Atallah and S. R. Kosaraju: "An Efficient Parallel Algorithm for the Row Minima of a Totally Monotone Matrix," 394-403, SODA 1991.

[4] O. Berkman, D. Breslauer, Z. Galil, B. Schieber and U. Vishkin: "Highly Parallelizable Problems," STOC, 309-319, 1989.

[5] P. G. Bradford: "Efficient Parallel Dynamic Programming," Technical Report # 352, Indiana University, April 1992.

[6] P. G. Bradford: "Efficient Parallel Dynamic Programming," Extended Abstract in the Proceedings of the 30th *Allerton Conference*, 185-194, 1992. Full version submitted.

[7] P. G. Bradford, G. J. E. Rawlins and G. E. Shannon: "Matrix Chain Ordering in Polylog Time with $n/\lg n$ Processors," Technical Report # 360, Indiana University, December, 1992. An update of this full version has been submitted.

[8] D. Z. Chen: "Efficient Geometric Algorithms on the EREW PRAM," Proceedings of the 28th *Allerton Conference*, 1990, full version submitted.

[9] T. H. Cormen, C. E. Leiserson and R. L. Rivest: *Introduction to Algorithms*, McGraw Hill, 1990.

[10] A. Czumaj: "Parallel algorithm for the matrix chain product and the optimal triangulation problem," *STACS*, Springer Verlag, LNCS # 665, 294-305, 1993.

[11] L. E. Deimel, Jr. and T. A. Lampe: "An Invariance Theorem Concerning Optimal Computation of Matrix Chain Products," North Carolina State Univ. Tech Report # TR79-14.

[12] Z. Galil and K. Park: "Parallel Dynamic Programming," Manuscript, 1992.

[13] S.-H. S. Huang, H. Liu, V. Viswanathan: "Parallel Dynamic Programming," *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing*, 497-500, 1990.

[14] T. C. Hu and M. T. Shing: "Computation of Matrix Product Chains. Part I," *SIAM J. on Computing*, Vol. 11, No. 3, 362-373, 1982.

[15] T. C. Hu and M. T. Shing: "Computation of Matrix Product Chains. Part II," *SIAM J. on Computing*, Vol. 13, No. 2, 228-251, 1984.

[16] S. K. Kim: "Optimal Parallel Algorithms on Sorted Intervals," TR 90-01-04, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1990.

[17] P. Ramanan: "An Efficient Parallel Algorithm for Finding an Optimal Order of Computing a Matrix Chain Product," Technical Report, WSUCS-92-2, Wichita State University, June, 1992.

[18] P. Ramanan: "An Efficient Parallel Algorithm for the Matrix Chain Product Problem," Technical Report, WSUCS-93-1, Wichita State University, January, 1993. Submitted for publication.

[19] W. Rytter: "On Efficient Parallel Computation for Some Dynamic Programming Problems," *Theoretical Computer Science*, Vol. 59, 297-307, 1988.

[20] L. G. Valiant, S. Skyum, S. Berkowitz and C. Rackoff: "Fast Parallel Computation of Polynomials Using Few Processors," *SIAM J. on Computing*, Vol. 12, No. 4, 641-644, Nov. 1983.