Finding A Minimum Cost Acceptable Path in Parallel[1]

Theodore Johnson
ted@squall.cis.ufl.edu

Panos E. Livadas
pel@cis.ufl.edu

University of Florida, Dept. of CIS
Gainesville, FL 32611-2024

Abstract

We consider the problem of finding a minimum cost acceptable path on a toroidal grid graph, where each horizontal and each vertical edge have the same orientation. An acceptable path is closed path that makes a complete horizontal and vertical circuit. We exploit the structure of this graph to develop efficient parallel algorithms for a message passing computer. Given p processors and an m by n toroidal graph, our algorithm will find the minimum cost acceptable path in $O(mn\log(m)/p)$ steps, if $p = O(mn/((m+n)\log(mn/(m+n))))$, which is an optimal speedup. Also we show that the algorithm will send $O(p^2(m+n))$ messages. The solution of this problem has applications in surface reconstruction [6, 5, 9]

1 Introduction

The task of finding minimum cost paths in parallel is a well studied problem. Parallel single source shortest path algorithms include those by Deo, Pang, and Lord [2], Quinn and Yoo [11], and Paige and Kruskal [10]. Parallel all-pairs-shortest path algorithms have been proposed based on solving the single source problem at each source [1, 8], matrix multiplication [4, 7, 13, 14, 10]. While Paige and Kruskal's algorithms [10] achieve optimal speedups under certain conditions, the proposed parallel shortest path algorithms are generally either limited in parallelism, or suboptimal. Efficient and optimal algorithms can be found by restricting the problem domain. For example, Eckstein [3], and Reghbati and Corneil [12] use breadth-first search to construct efficient parallel algorithms for the single source problem when all edges have the same weight.

In this paper, we will examine a particular type of shortest path problem, that of finding the minimum cost acceptable path in a toroidal graph. This problem has been studied by Keppel [6], Fuchs, Kedem, and Uselton [5], and by Livadas [9] in the context of surface reconstruction from a set of points on two planes. There, it is shown that the best surface can be found in $O(mn\log(m))$ time, where the toroidal graph that the surface maps to is of dimensions m by n. We make use of the structure of the graph to find an optimal parallel algorithm, suitable for implementation on a message passing parallel computer.

[1] University of Florida Dept. of CIS Technical Report 92-010 . available bia anonymous ftp at cis.ufl.edu:cis/tech-reports/tr92/tr92-010.ps.Z

1.1 Definition of the problem

We review the notation from [5] and [9]. Let m and n be two positive integers. We will denote by $\mathcal{M}$ and $\mathcal{N}$ the sets of all non-negative integers bounded above by $(m-1)$ and $(n-1)$, respectively. In addition we will denote by $\mathcal{R}^+$ the set of positive real numbers. We will search for paths on a particular type of toroidal graph, a p^2-toroidal graph, which we define to be a toroidal directed and weighted graph, $G = (V, E, \omega)$, where the sets V, E, and the function ω are defined as follows:

1. The set of vertices, V, can be enumerated in such way that

$$V = \{v \in G \ : \ v \ = \ v_{i,j} \ \forall_{i \in \mathcal{M}} \ \forall_{j \in \mathcal{N}}\}$$

2. The set of edges, E, is defined as follows

$$E = \{e_{\overline{ij}} \ : \ e_{\overline{ij}} = v_{i,j}\, v_{i,(j+1) \bmod n} \ \forall_{i \in \mathcal{M}} \ \forall_{j \in \mathcal{N}}\} \cup \{e_{i\overline{j}} \ : \ e_{i\overline{j}} = v_{i,j}\, v_{(i+1) \bmod n, j} \ \forall_{i \in \mathcal{M}} \ \forall_{j \in \mathcal{N}}\}$$

3. The function ω is a weight function (i.e, $\omega : \ E \rightarrow \mathcal{R}^+$) and we write $\omega_{\overline{ij}}$ and $\omega_{i\overline{j}}$ to denote the images $\omega(e_{\overline{ij}})$ and $\omega(e_{i\overline{j}})$ respectively.

Let i be an integer such that $i \in \mathcal{M}$ and consider the vertex $v_{i,0} \in V$. We define an *acceptable path at* i, π, to be a closed path with initial vertex $v_{i,0}$ and satisfying the following two properties

1. $\forall_{s \in \mathcal{M}} \ \exists_{k \in \mathcal{N}}$ st. $((e_{\overline{sk}} \in \pi) \wedge (e_{\overline{s_1 k}} \in \pi \ \wedge \ e_{\overline{s_2 k}} \in \pi \ \rightarrow s_1 = s_2))$

2. $\forall_{k \in \mathcal{N}} \ \exists_{s \in \mathcal{M}}$ st. $\Big((e_{s\overline{k}} \in \pi) \wedge (e_{s\overline{k_1}} \in \pi \ \wedge \ e_{s\overline{k_2}} \in \pi \ \rightarrow k_1 = k_2)\Big)$

To say it differently, each acceptable path contains exactly one horizontal edge between any two adjacent columns and exactly one vertical edge between any two adjacent rows.

Now let $\mathcal{P}(i)$ denote the set of all acceptable paths at i. It is easy see that if $\pi \in \mathcal{P}(i)$, the length of π is equal to $(m+n)$. An acceptable path is either a vertex-simple cycle, or consists of two vertex-simple cycles that share one vertex. Figure 1 illustrates a p^2-toroidal graph ($m = 5$, $n = 6$), and the two types of acceptable paths.

We define that *cost* of a path, π to be the sum of the weights of the edges in π. We define the minimum cost path at i denoted by $\mu(i)$ to be the one among all $\pi \in \mathcal{P}(i)$ such that its cost is minimal. Finally, let μ_G be the path among all $\mu(i)$'s that yields the minimal cost.

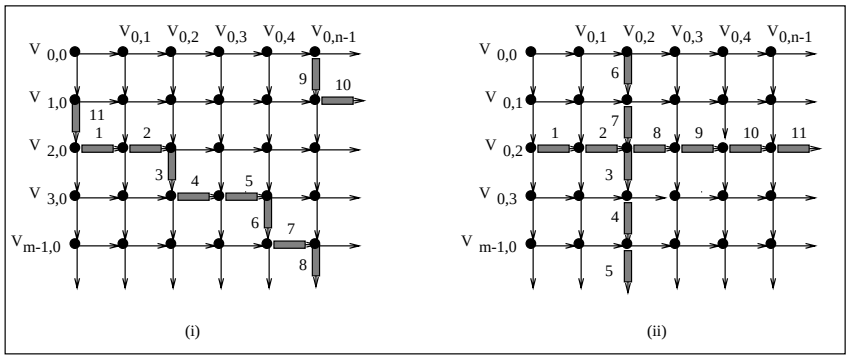

Figure 1: A p^2-toroidal graph and two acceptable paths at $i = 2$

2 Mapping the Toroidal Graph to a Planar Graph

By cutting G open and gluing the two ends of G together we obtain a new planar graph $G'(V', E', \omega')$, as illustrated in Figure 2. We define $\mathcal{M}' = \{0, 1, 2, \ldots, 2m\}$ and $\mathcal{N}' = \{0, 1, 2, \ldots, n\}$, so that

$$V' = \{v' \in G \,:\, v = v'_{i,j} \;\forall_{i \in \mathcal{M}} \;\forall_{j \in \mathcal{M}}\}$$

and

$$E' = \{e'_{\overline{ij}} \,:\, e'_{\overline{ij}} = v'_{i,j} v'_{i,(j+1) \bmod n} \;\forall_{i \in \mathcal{M}'} \;\forall_{j \in \mathcal{N}}\} \cup \{e'_{i\overline{j}} \,:\, e'_{i\overline{j}} = v'_{i,j} v'_{(i+1) \bmod n,j} \;\forall_{i \in \mathcal{M}'} \;\forall_{j \in \mathcal{N}'}\}$$

Furthermore, the weight function $\omega' : E' \rightarrow \mathcal{R}^+$ is defined by $\omega'(e'_{\overline{ik}}) = \omega(e_{\overline{i \bmod m, k \bmod n}})$ and $\omega'(e'_{i\overline{k}}) = \omega(e_{i \bmod m, \overline{k \bmod n}})$ for each i and k.

Let i be an integer variable with $i \in \mathcal{M}$; then the simple path in G' defined by

$$v'_{i,0} = v'_{i_1,j_1},\; v'_{i_2,j_2},\; \ldots,\; v'_{i_{(m+n+1)},j_{(m+n+1)}} = v'_{(m+1),n}$$

has a preimage in G the path

$$v_{i,0} = v_{i_1 \bmod m, j_1 \bmod n},\; v_{i_2 \bmod m, j_2 \bmod n},\; \ldots,\; v_{i_{(m+n+1) \bmod m}, j_{(m+n+1) \bmod n}} = v_{i,0} \qquad (1)$$

which is is an acceptable path at i in G.

Hence, there is a natural one-to-one and onto mapping from the set of simple paths from $v'_{i,0}$ to $v'_{(m+1),n}$ in G' to the set of acceptable paths in G which start and end at $v_{i,0}$ $\forall i \in \mathcal{M}$. Our problem is therefore equivalent to the following:

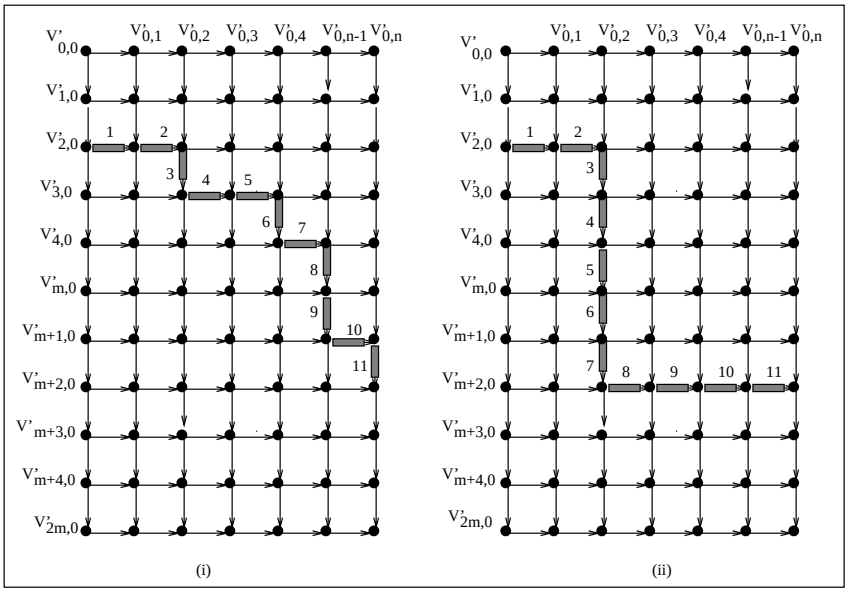

Figure 2: Mapping the p^2-toroidal graph of Figure 1 to a planar graph. The images of the two acceptable paths in Figure 1 (i) and (ii) are shown in (i) and (ii), respectively.

Find the minimum cost path $\pi'(i)$ among all simple paths with initial vertex at $v'_{i,0}$ and terminal vertex at $v'_{(m+i),n}$ $\forall i \in \mathcal{M}$. The minimum cost path $\mu'_{G'}$ of G' is the one among all $\pi'(i)$'s ($i \in \mathcal{M}$) of minimum cost, and the required path $\mu(G)$ is the preimage of $\mu'_{G'}$ and can be obtained via the mapping defined in equation (1).

In the remainder of this paper, we will work in the planar graph, and we will drop the primes that distinguish between the toroidal and the planar graph.

3 Non-crossing Paths

Let $\mu(i)$ and $\mu(j)$ be two minimal cost acceptable paths at $v_{i,0}$ and $v_{j,0}$, respectively. Assuming that $i < j$ we say that $\mu(i)$ *does not cross* $\mu(j)$, if and only if, whenever $e^{i}_{sk} \in \mu(i)$ then for all $r > 0$ there is no edge of the form $e^{j}_{s-rk} \in \mu(j)$. In other words, $\mu(i)$ lies entirely "above" $\mu(j)$ in the sense that they do not "cross" but they may still have common vertices and edges. In the planar image, no two minimal cost paths $\mu(i)$ and $\mu(j)$ cross; for if they did, we could splice the paths and come up with a shorter path for one of $\mu(i)$ and $\mu(j)$.

For each $k \in \mathcal{N}$ we define two integers min_{i_k} and max_{j_k} as follows

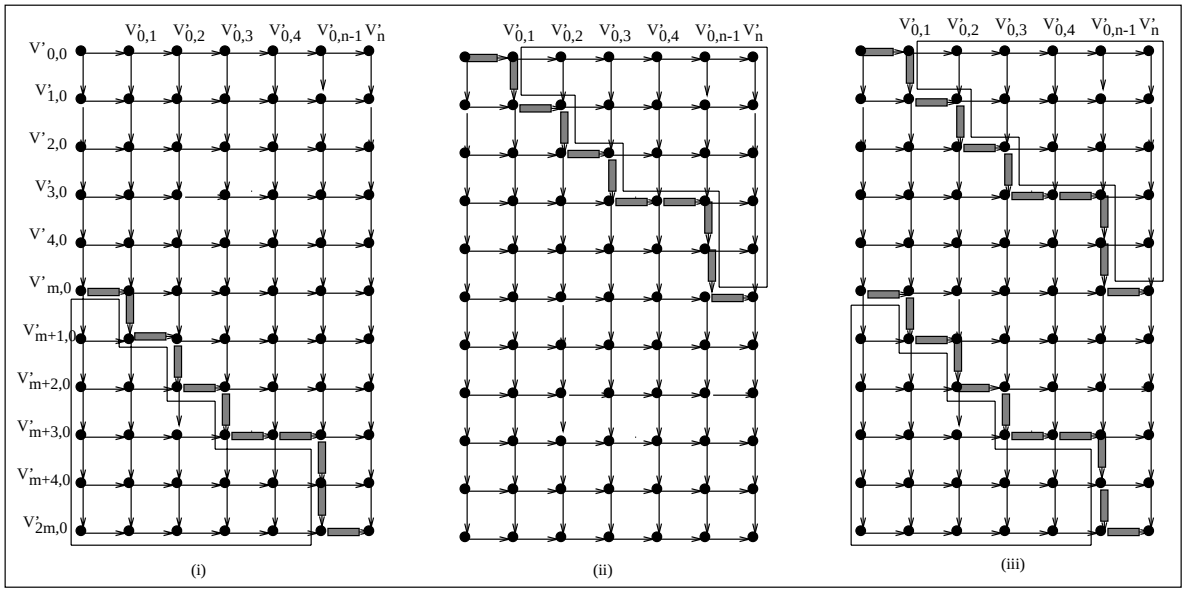

Figure 3: Determining the subgraph of G spanned by $V(0,m)$.

$$min_{i_k} = \quad \min\{i \in \mathcal{M} \text{ st. } (e_{\overline{ik}} \vee e_{i\overline{k}}) \in \mu(i)\}$$

$$max_{i_k} = \quad \max\{j \in \mathcal{M} \text{ st. } (e_{\overline{jk}} \vee e_{j\overline{k}}) \in \mu(j)\}$$

We next define $U_k(i,j) = \{v_{i,k} \in V \ : \ min_{i_k} \leq i \leq max_{j_k}\}$ and $G_{i,j}$ to be the subgraph of G that is spanned by the set of vertices $V(i,j)$ where

$$V(i,j) = \quad \bigcup_{k=0}^{n} U_k(i,j) \tag{2}$$

According to our earlier discussions we can see that the determination of the path μ_G can be done as follows. First, we determine the path $\mu(0)$ (Fig. 3(i)). Now, $\mu(m)$ is a copy of $\mu(0)$ which starts at $v_{m,0}$ and ends at $v_{(m+m),n}$ (Fig. 3(ii)). Hence, calculation of all other paths $\mu(1)$, $\mu(2)$, ..., $\mu(m-1)$, can be restricted to the subgraph of G that is obtained by "chopping off" the portion of the graph G lying above $\mu(0)$ and below $\mu(m)$. In view of equation 2, these paths may be found in the subgraph $G_{0,m}$ of G spanned by $V(0,m)$ (Fig. 3(iii)).

Suppose that instead of calculating $\mu(1)$ after calculating $\mu(0)$, you calculate $\mu(m/2)$. Now, the paths $\mu(i)$, $i = 1 \ldots m/2-1$ can be calculated in $V(0,m/2)$, and the paths $\mu(i), i = m/2+1 \ldots m-1$ can be calculated in $V(m/2,m)$. The regions $V(0,m/2)$ and $V(m/2,m)$ can also be divided into finer pieces

recursively, resulting in the Serial Algorithm. We define $W(m)$ to be the subgraph of G that the serial algorithm searches to find $\mu(m)$. Fuchs et al. [5] show this algorithm reduces the time to compute the minimum cost acceptable path from $mT(m,n)$ to $T(m,n)\log(m)$, where $T(m,n)$ is the time to solve the single source shortest path problem on a p^2-toroidal graph.

Serial Algorithm

```
shortestpath(G) {
      /* find the shortest path in 2n x m graph G */
   P0 = findonepath(G,0)
   G'=restrict(G,P0)
   {P1,..,Pm}=findallpaths(G',1,m)
   return(min(P0,..,Pn))
}


findallpaths(G,pathlo,pathhi) {
   m=(pathlo+pathhi)/2
   if(pathlo<m) {
      Pm=findonepath(G,m)
      Plo=findallpaths(restrict(G above Pm),pathlo,m-1)
      Phi=findallpaths(restrict(G below Pm)m+1,pathhi)
   }
return(union(Plo,Pm,Phi)
}
```

4 Finding Acceptable Paths in Parallel

In the serial algorithm, after path $\mu((i+j)/2)$ is found in $V(i,j)$, G can be further divided into $V(i,(i+j)/2)$ and $V((i+j)/2,j)$. These are independent subtasks, so after finding the path $\mu((i+j)/2)$, we can find the paths $\mu(i+1),\ldots,\mu((i+j)/2-1)$ and $\mu((i+j)/2+1),\ldots,\mu(j-1)$ in parallel.

As with a parallel quicksort algorithm, this algorithm will be efficient only if we can find a single shortest path efficiently in parallel. Given the structure of the graph G, we can accomplish this via a dynamic programming algorithm.

Let us consider the problem of finding the shortest path from a node v_{i_1,j_1} to another node v_{i_2,j_2}, $i_1 \le i_2$, $i_1 \le i_2$, $v_{i_1,j_1} \neq v_{i_2,j_2}$ in some subgraph $W \subseteq G$. To simplify the presentation, we will normalize the graph coordinates by translating $v_{r,s}$ to $u_{r-i_1,s-j_1}$. Therefore the problem translates into finding the shortest path $P(i,j)$ from $u_{0,0}$ to $u_{i,j}$, where $i = i_2 - i_1$ and $j = j_2 - j_1$. The edges in the graph W only head down or to the right, never up or to the left. Therefore, all paths to $u_{i,j}$ must travel through $u_{i-1,j}$

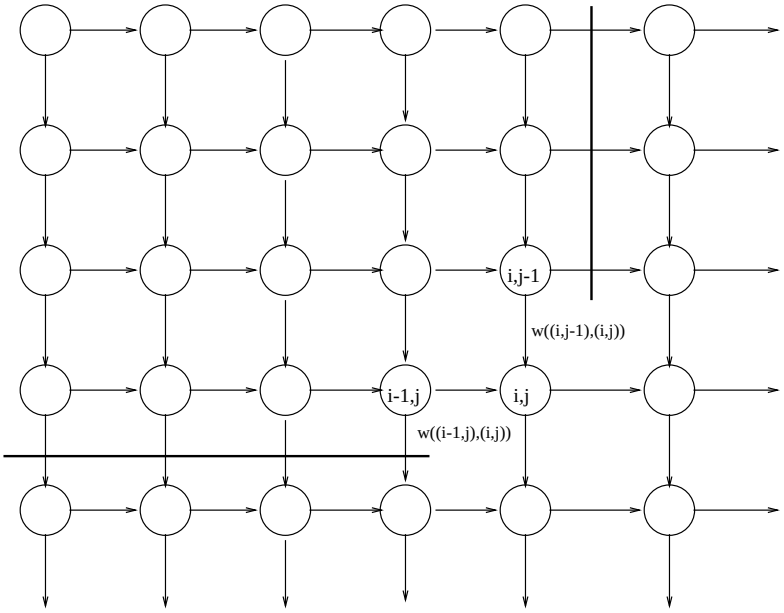

Figure 4: Dynamic programming calculation of the shortest path.

or $u_{i,j-1}$, so that the minimum length path from $u_{0,0}$ to $u_{i,j}$ must have as its penultimate node either $u_{i-1,j}$ or $u_{i,j-1}$. See figure 4.

Let $C(i,j)$ be the cost of the minimum cost path from $u_{0,0}$ to $u_{i,j}$. Let:

$$w(\overline{i},j) = \begin{cases} \omega_{\overline{(i-i_1)},j-j_1} & v_{(i-i_1),j-j_1} \in W \\ \infty & v_{(i-i_1),j-j_1} \notin W \end{cases}$$

$$w(i,\overline{j}) = \begin{cases} \omega_{i-i_1,\overline{(j-j_1)}} & v_{i-i_1,(j-j_1)} \in W \\ \infty & v_{i-i_1,(j-j_1)} \notin W \end{cases}$$

Then we can find the recurrence:

$$C(i,j) = \quad \min\left(C(i-1,j) + w(\overline{i-1},j), C(i,j-1) + w(i,\overline{j-1})\right)$$

Further, $P(i,j)$ is either $P(i-1,j) \mid e_{\overline{(i-i_1)-1},j-j_1}$ or $P(i,j-1) \mid e_{i-i_1,\overline{(j-j_1)-1}}$, depending on which path has lower weight. Calculating a shortest path μ_s thus reduces to a dynamic programming problem, iterating across all $v_{i,j} \in W(s)$ such that $i+j = d+s$ and all d between 0 and $m+n$.

We can parallelize the dynamic programming calculation for a message passing parallel computer by splitting up the iteration for a particular value of d. In order to minimize communication, the nodes for which a processor is responsible should be contiguous. To balance the load, each processor should be responsible for the same number of nodes along a diagonal. These two considerations lead to the distribution of nodes among the processors shown in figure 5.

Let $S_d \subseteq W(s)$ be the set of nodes at manhattan distance (i.e, L_1 norm) d from $v_{s,0}$, $k_d = |S_d|$, and

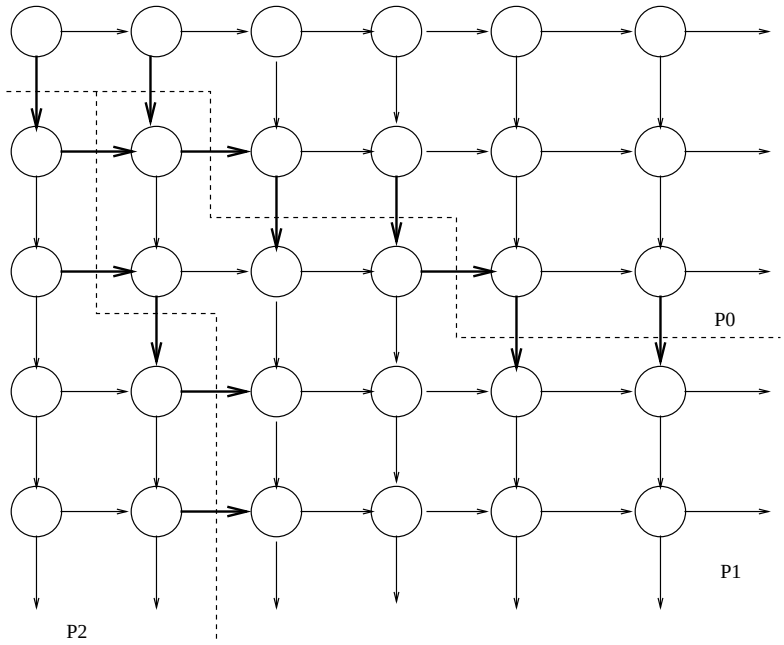

Figure 5: Allocation of nodes to processors

let $v_{q,r}$ be the node in S_d st. $v_{q',r'} \in S_d \Rightarrow q' \geq q$. If there are p processors computing the paths, then processor i, $i = 0, \ldots, p-1$ is responsible for the computing the shortest paths to the nodes $(q+l, r-l)$, where $\text{floor}(k_d i/p) \leq l < \text{floor}(k_d(i+1)/p)$. The communication in this algorithm takes place across the dotted lines that separate the nodes for which each processor is responsible. If a processor q calculates a shortest path to node to node v, $(v,w) \in E$, and processor r calculates the shortest path to w, then q sends the length of the shortest path to v to r. Correspondingly, r waits until it is sent the length of the shortest path to v before calculating the length of the shortest path to w.

After calculating the cost of $\mu(s)$, the actual path can be found by working backwards from the sink. Once this path is calculated, it is distributed to the all p processors that co-operated to calculate $\mu(s)$. The path $\mu(s)$ is used to divide $W(s)$ into $W(r)$ and $W(t)$ by the processors that will calculate $\mu(r)$ and $\mu(t)$, respectively. The number of processors assigned to calculate $\mu(r)$ and $\mu(t)$ is proportional to relative sizes (number of edges) of $W(r)$ and $W(s)$, respectively. The sizes of $W(r)$ and $W(t)$ can be calculated while $\mu(s)$ is calculated.

These considerations lead to the our parallel algorithm, listed below. Our parallel algorithm has two noteworthy features. First, all synchronization and communication can be performed using message passing, so that the algorithm can be implemented on available parallel computers. Second, when the graph is split up, the number of processors assigned to calculate paths above $\mu(s)$ is proportional to the

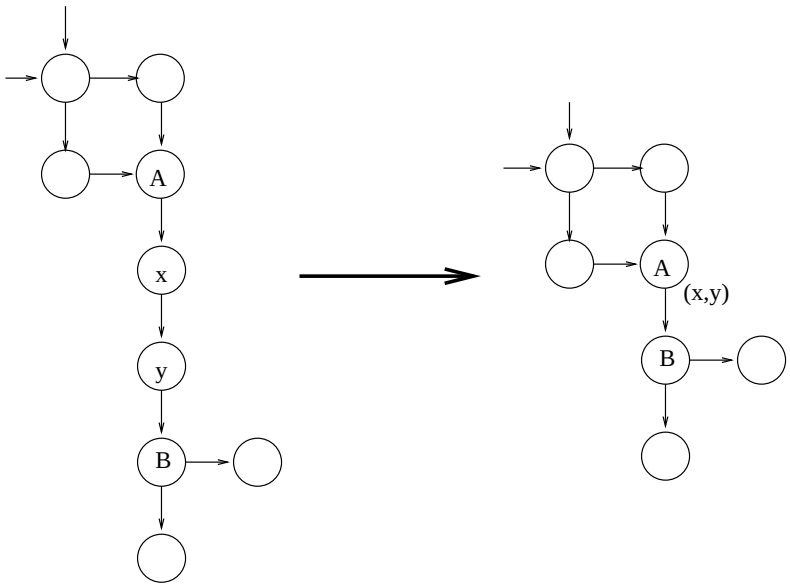

Figure 6: Removing cut edges from further processing

number of graph nodes above $\mu(s)$. This load balancing is necessary because the graph can be split in a very unbalanced fashion.

We define W_{lo} to be the subgraph in which the paths $\mu(\texttt{pathlo})$ through $\mu(\texttt{m}-\texttt{1})$ will be found, and we define W_{hi} similarly. Our load balancing mechanism is based on the sizes of W_{lo} and W_{hi}, respectively. There is one problem – one of W_{hi} and W_{lo} might be very small, and contain sections where all $\mu(i)$ that are found in the subgraph must contain the same path. In this case, the remaining work contained in the subgraph might be somewhat larger than is indicated by the size of the subgraph. To account for this problem, whenever there is a cut edge in the subgraph, its edges are joined together (see figure 6). This condition occurs whenever there are two adjacent d in the subgraph such that $|S_d| = 1$, and can be detected and corrected for when the paths are split and distributed with only a constant time penalty. As a result, we can guarantee that the remaining work to find $\mu(\texttt{pathlo})$ through $\mu(\texttt{m}-\texttt{1})$ is [5] $|W_{lo}|l_{lo}\log(l_{lo})$, where $l_{lo} = pathlo - m + 1$, and the remaining work to find $\mu(\texttt{m}+\texttt{1})$ through $\mu(\texttt{pathhi})$ is $|W_{hi}|l_{hi}\log(l_{hi})$. If p_{lo} processors are assigned to find the lower numbered paths and p_{hi} processors are are assigned to find the higher numbered paths, then the load will be balanced if $|W_{lo}|l_{lo}\log(l_{lo})/p_{lo} = |W_{hi}|l_{hi}\log(l_{hi})/p_{hi}$. Since $p_{lo}+p_{hi} = p = \texttt{prhi}-\texttt{prlo}+1$, we find that to balance the load, $p_{lo} = p/(1+|W_{lo}|/|W_{hi}|) = |W_{lo}|p/(|W_{lo}+|W_{hi}|)$.

Parallel Algorithm

```
shortestpath(G,p,n) {
      /* find the shortest path in 2m x n planar graph G using p processors*/
   P0 = findonepath(G;1,p;0)
   G'=restrict(G,P0)
   findallpaths(G;1,p;1,n)
   return(min(P0,..,Pn))
}


findallpaths(G,prlo,prhi,pathlo,pathhi){
   m=(pathlo+pathhi)/2
   if(pathlo<m){
      Pm=findonepath(G; prlo,prhi; m)
      if(prlo<prhi) then
         in parallel{
            n_pr_lo=(prhi-prlo+1)*|W_lo|/(|W_lo + |W_hi|
            findallpaths(restrict(G below Pm),prlo,prlo+n_pr_lo,m+1,pathhi)
            findallpaths(restrict(G above Pm),prlo+n_pr_lo+1,prhi,pathlo,m-1)
         }
      else{
         findallpaths(restrict(G above Pm),prlo,prlo,pathlo,m-1)
         findallpaths(restrict(G below Pm),prlo,prlo,m+1,pathhi)
      }
   }
}
```

4.1 Analysis

Theorem 1 *If $p = O(mn/((m+n)\log(mn/(m+n))))$, then the execution time of the algorithm is $O(mn\log(m)/p)$. Further, the number of messages sent is bounded by $O(p^2(m+n))$.*

Proof: For the purposes of the analysis, we will assume that every entry into another parallel recursive step is synchronized across all processors. This assumption is not required for the algorithm to execute correctly, and provides a worst-case bound. We will also assume that each individual message is large enough to carry information about a single node. That is, we don't count bit complexity, but that each message is limited to a 'reasonable' size. Counting bit complexity multiplies the message passing cost by a factor of $\log(mn)$.

If $p > 1$ processors are assigned to calculate $\mu(s)$ in $W(s)$, then calculating the cost of the $\mu(s)$ requires $O(max(m+n, |W(s)|/p))$ steps, and $O(p(m+n))$ messages. Determining the path of $\mu(s)$ requires an additional $O(m+n)$ steps and an additional $O(m+n)$ messages. Distributing the path of $\mu(s)$ requires

$O(p(m+n))$ messages and $O((m+n)\log p)$ steps. So, the total cost of finding a single shortest path $\mu(s)$ using p processors is $O(max((m+n)\log p, |W(s)|/p))$ steps and $O(p(m+n))$ messages.

The first path, $\mu(0)$ is calculated in $V(0, 2n-1)$, which contains $2nm$ nodes. The second path, $\mu(n/2)$, is calculated in $W(n/2)$, which contains mn nodes. So, the cost to calculate the first two paths and subsequently divide the graph is $O(max((m+n)\log p, mn/p)$ steps and $O(p(m+n))$ messages. The subsequent two paths, $\mu(n/4)$ and $\mu(3n/4)$ are calculated in parallel. Suppose $p_{n/4}$ processors are assigned to calculate $\mu(n/4)$ and $p_{3n/4}$ are assigned to calculate $\mu(3n/4)$, $p_{n/4} + p_{3n/4} = p$. The number of steps required to calculate these paths is $O(max((m+n)\log p_{n/4}, (m+n)\log p_{3/4}, |W(n/4)|/p_{n/4}, |W(3n/4)|/p_{3n/4})$. Since the processor allocation is proportional to the subgraph size, the number of steps is bounded by $O(max((m+n)\log p, mn/p)$. The number of messages sent is bounded by $O(p_{n/4}(m+n)+p_{3n/4}(m+m)) = O(p(m+n))$.

If there are k recursive steps before every processor is executing the dynamic programming algorithm serially, then at most $O(k \max((m+n)\log p, mn/p))$ steps are executed. Afterwards, there are $\log m - k$ recursive rounds of the serial algorithm, each of which requires $O(mn/p)$ steps. There are at most $O(\log m)$ recursive steps in which a parallel shortest path algorithm is executed, so the algorithm will calculate all paths after $O(max((m+n)k\log p + (logm - k)mn/p, mn\log(m)/p)) = O(max((m+n)\log(m)\log(p), mn\log(m)/p)$ steps. In addition, at most $O((m+n)p^2)$ messages will be sent. To finish, the processors must agree on the lowest cost path. If all processors keep track of the lowest cost path that they have helped to calculate, this final step requires only $\log(p)$ steps and $O(p)$ messages. This step does not affect the complexity of the algorithm.

This algorithm has an optimal speedup if $(m+n)\log(m)\log(p) < mn\log(m)/p$, or if $p = O(mn/((m+n)\log(mn/(m+n))))$ •.

5 Conclusions

We have parallelized a shortest-path problem whose structure allows efficient solutions. The problem has applications in image reconstruction. Our algorithm has an optimal speedup for $p = O(d/log(d))$ processors on a d by $d\,p^2$-toroidal graph. Further, the algorithm is based on message passing, and does not require that the processors be synchronized, so that the algorithm can be practically implemented on current multiprocessors. The algorithm has applications in surface reconstruction [6, 5, 9].

References

[1] C.C. Chen. A distributed algorithm for shortest paths. *IEEE Trans. Computers*, C-31(9):898–899, 1982.

[2] N. Deo, C.Y. Pang, and R.E. Lord. Two parallel algorithms for shortest path problems. In *Proceedings of the 1980 Int's Conference on Parallel Processing*, pages 244–253, 1980.

[3] D.M. Eckstein. *Parallel Algorithms for Graph Theoretic Problems.* PhD thesis, University of Illinois, Dept. of Mathematics, 1977.

[4] A. Frieze and L. Rudolph. A parallel algorithm for all pairs shortest paths in a random graph. In *Proc.* 22^{nd} *Allerton conf.*, pages 663–670, 1984.

[5] H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, 1977.

[6] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19:2–11, 1975.

[7] L. Kucera. Parallel computation and conflicts in memory access. *Inf. Proc. Letters*, 14(2):93–96, 1982.

[8] G.D. Lakhani. An improved distribution algorithm for shortest paths problem. *IEEE Transactions on Computers*, C-33(9):855–857, 1984.

[9] Panos Livadas. A reconstruction of an unknown 3-D surface from a collection of its cross sections: An implementation. *Int'l Journal of Computer Math*, 26, 1989.

[10] R.C. Paige and C.P. Kruskal. Parallel algorithms for shortest path problems. In *Int'l Conference on Parallel Processing*, pages 14–20, 1985.

[11] M. Quinn and Y. Yoo. Data structures for the efficient solution of graph theoretic problems on tightly-coupled computers. In *Proceedings of the International Conference on Parallel Processing*, pages 431–438, 1984.

[12] E. Reghbati and D.G. Corneil. Parallel computations in graph theory. *SIAM J. Computing*, 7(2):230–236, 1978.

[13] J.H. Reif and J. Spirakis. The expected time complexity of parallel graph and digraph algorithms. Technical Report TR-11-82, Aiken Computation Lab., Harvard University, 1982.

[14] C. Savage. *Parallel Algorithms for Graph Theoretic Problems.* PhD thesis, University of Illinois, Dept. of Mathematics, 1977.