CHAPTER VII

AN EFFICIENT PARALLEL ALGORITHM
FOR APSL PROBLEM ON A STRONGLY CHORDAL GRAPH

7.1. Introduction

A very rich group of problems can be cast as problems on some kind of graph. These problems originate not just in connection with computers, but through the sciences, industry and business. The development of efficient algorithms to solve many graph problems has had a major impact on investigators' ability to solve real problems in all these fields. In virtually all areas of computer science, graphs are used to organize data, to model algorithms, and as generally as a powerfully tool to represent computational concepts. Paths [78], in particular, are omnipresent. Many branches of engineering and science relay on graph for representing a wide variety of objects from electrical circuits, chemical compounds, and crystals to genetical processes, sociological structures, and economic systems. Keeping this in view, it is pertinent to develop efficient algorithms to manipulate graph.

A thorough investigation reveals that most of the preceding literature implicitly shows that the All-Pairs-Shortest-length (APSL) problem [95,96,98] has been quite rigorously studied on various graphs. Though significant results have been established on APSL problem in different classes of graphs, no work has been reported at least to the best of our knowledge on the class of strongly chordal graph using CREW PRAM model. This chapter addresses an efficient parallel
algorithm for APSL problem on strongly chordal graphs using CREW PRAM model, which runs in $O(n')$ time with efficiency 1, where $n'$ is the number of vertices.

7.2. Preliminaries

7.2.1. A Chord:

An edge is a Chord of a cycle if it connects two vertices of the cycle, but is not itself an edge within the cycle.

7.2.2. A Clique:

A Clique in a graph is a subset of vertices which induce a complete subgraph. A maximal clique is a clique that is not properly contained in any other clique.

7.2.3. Chordal graph:

A graph is a Chordal Graph if and only if every cycle of length greater than three has a chord.

7.2.4. Strongly chordal graph:

A graph is Strongly chordal if it is chordal and every even cycle of length six or more has an odd chord. The class of strongly chordal graphs is an interesting subclass of strongly chordal graphs which properly includes trees and directed path graphs.
7.2.5. Single-source-shortest-length (SSSL) problem:

Let $G = (V, E)$ be an undirected, simple graph with no self loops. Let $|V| = n'$ and $|E| = m$. The Single-Source-Shortest-Length (SSSL) problem is to find the length of the shortest path from a given source vertex $v$, to all other vertices.

7.2.6. Single-source-shortest-path (SSSP) problem:

Let $G = (V, E)$ be an undirected, simple graph with no self loops. Let $|V| = n'$ and $|E| = m$. The Single-Source-Shortest-Path (SSSP) problem is to find the shortest path from a given source vertex $v$ to all other vertices.

7.2.7. All-pairs-shortest-path (APSP) problem:

Let $G = (V, E)$ be an undirected, simple graph with no self loops. Let $|V| = n'$ and $|E| = m$. The All-Pairs-Shortest-Path (APSP) problem is to find the shortest paths between all pairs of vertices.

7.2.8. All-pairs-shortest-length (APSL) problem:

Let $G = (V, E)$ be an undirected, simple graph with no self loops. Let $|V| = n'$ and $|E| = m$. The All-Pairs-Shortest-Length (APSL) problem is to find the length of the shortest path between all pairs of vertices. The length of the path is denoted by the number of edges in the path.

7.2.9. Simplicial:

A vertex $v$ is called simplicial if the subgraph induced by $N[v]$ is a clique.
7.2.10. Perfect elimination ordering (PE)

A graph $G$ is chordal if and only if it is possible to order the vertices $(v_1,v_2,\ldots,v_n)$ in such way that, for each $i \in \{1,2,\ldots,n\}$, $v_i$ is a simplicial vertex of $G_i$, where $G_i$ is the subgraph induced by a vertex set $\{v_i,v_{i+1},\ldots,v_n\}$. Such an ordering is called a perfect elimination ordering.

7.2.11. Strong elimination ordering (SE):

Let $N_0[V]$ denote the closed neighborhood of $V$ in $G$. The ordering of vertices $(v_1,v_2,\ldots,v_n)$ is called a Strong Elimination (SE), if it is a perfect elimination ordering and for each $i<j<k$, if $v_j,v_k \in N_0[V_i]$, then $N_0[V_j] \subseteq N_0[V_k]$.

7.3. Description of Graph

Let us assume that the given graph is connected and that a strong elimination ordering is available for the given graph. Let $G = (V,E)$ be an undirected, strongly chordal graph with strong elimination ordering. Let $\text{Next}[i]$ denote the largest vertex that is adjacent to $i$, and $U_{ij}$ denotes the set of all vertices reachable from $i$ by a path of length $j-1$, then $\text{Max}[i,j]$ is defined as

$$\text{Max}[i,j] = \begin{cases} \max(U_{ij}), & \text{if } j < t_i \\ \text{NULL}, & \text{otherwise} \end{cases}$$

where $t_i$ be the smallest $j$ such that $S_{ij}$ contains the last vertex. Figure-7.1 gives a strongly chordal graph in which the vertices have been numbered according to strong elimination ordering of a graph.
Property-1:
Let $G = \{V, E\}$ be an undirected, strongly chordal graph with strong elimination ordering. Let $(v_1, v_2, \ldots, v_n)$ is ordering of vertices. For all $v_i < v_n$, there exists a $v_j$ such that $v_i > v_j$ and $(v_i, v_j) \in E$.

Property-2:
For all $i, j$, if $\text{Max}[i,j]$ and $\text{Max}[i, j + 1]$ are non-null, then $\text{Max}[i,j] < \text{Max}[i, j+1]$ and $(\text{Max}[i,j], \text{Max}[i, j+1]) \in E$.

Property-3:
If the vertex $v_i$ comes before $v_j$ and $k$ be the integer satisfying $\text{Max}[i,k] < j < \text{Max}[i,k+1]$, then there exists a shortest path from $i$ to $j$ containing $\text{Max}[i,k]$.

Property-4:
Let $v_i < v_j$, and $j$ satisfy the condition that $\text{Max}[i, j] < j < \text{Max}[i, k+1]$. If $(\text{Max}[i,k], j) \notin E$, there exists a shortest path from $i$ to $j$ containing $\text{Max}[i, k+1]$.

7.4. Description of Algorithm:
Any parallel algorithm viewed as a solution to a problem is to be mentioned along with the model(s) with which the algorithms are designed. The reason is that the topology of the model sometimes contributes to the time taken by the algorithm. So, we choose CREW PRAM model in accordance with the parallelizing requirements.
Figure 7.1. A Strongly Chordal Graph.
The CREW PRAM model is an SM SIMD instance with which concurrent read exclusive write facility is provided. It is the facility of simultaneous reading from a memory location by more than a processor, but when it comes to writing onto a location, it can be done by only one processor.

Our CREW PRAM algorithm for APSL problem on strongly chordal graph starts to work by computing a shortest path between j and Max[i,k+1], where Max[i,k] < j < Max[i,k+1], for all possible values of i, j and k using Pq processors. According to [106], either the shortest path between vertices i and j has Max[i,k] as the penultimate vertex or is the shortest path between vertex i and Max[i,k] followed by the shortest path between Max[i, k+1] and j. Let Mi denote the i\(^{th}\) largest vertex, such that Next[v] = Mi for some vertex v and Li denote the smallest vertex such that Next [Li] = Mi. The values of Next[i], Max [i,j], Mi and Li corresponding to the Figure 7.1 are tabulated in the tables, Table 7.1 and Table 7.2. When we try to find length of the shortest path, we come across three cases. In the first case, when (Mq,j) ∈ E, then the shortest path between j and Mq is of length one. The second case is: if Next[j] < Mq, then the shortest path is between j and Mq, such that j takes values in the descending order of the strong elimination ordering. In the third case, next[j] > Mq, our algorithm computes the shortest path
Table 7.1. The value of the arrays Next (i) and Max (i,j) for the graph of Figure 7.1.
<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 7.2. The arrays M and L for the graph*
between j and M_q in the ascending order of q such that L_q < j < M_q. This method is presented in the form of an algorithm in the next section.

7.5. CREW-APSL Algorithm

**Input:** Some representation of a given strongly chordal graph with the vertices numbered in accordance with a SE ordering.

**Output:** LSP[i,j], denoting the length of a shortest path from i to j and S[i,j] denoting the vertex next to i in some shortest path from i to j.

**Method:**

Step 1: begin

for q ← 1 to n do in parallel

each processors P_q performs:

If (M_q ≠ NULL) then

begin

v ← (M_q - 1)

while( v > L_q)

if (v,M_q) ∈ E then

begin

LSP[v,M_q] ← 1

S[v,M_q] ← M_q

end

v ← v - 1

end

denoting the vertex next to i in some shortest path from i to j.
step 2: for q ← 1 to n' do in parallel

each processors P_q performs:

if (M_q ≠ NULL) then

begin

v ← M_q + 1

while (v > L_q)

begin

if (v,M_q) ∉ E then

begin

LSP[v,M_q] ← LSP[Next[v],M_q] + 1

S[v,M_q] ← Next[v]

end

v ← v - 1

end

end

step 3: for i ← 1 to n'-1 do in parallel

each processor P_i performs:

begin

k ← 2

j ← i + 1

while (Max[i,k] ≠ NULL)

begin

while (j ≤ Max[i,k])

begin

123
if \( j \leftarrow \text{Max}[i,k] \) or \((j, \text{Max}[i,k-1]) \in E\) then

begin

\( \text{LSP}[i,j] \leftarrow k - 1 \)

\( S[i,j] \leftarrow \text{Next}[i] \)

end

else

begin

\( \text{LSP}[i,j] \leftarrow \text{LSP}[i, \text{Max}[i,k]] + \text{LSP}[\text{Max}[i,k],j] \)

\( S[i,j] \leftarrow \text{Next}[i] \)

end

\( j \leftarrow j + 1 \)

end

\( k \leftarrow k + 1 \)

end

end
7.6. Time complexity analysis

**Theorem 7.6.1:**

Our CREW-APSL algorithm correctly finds the All-Pairs-Shortest-Length on strongly chordal graph.

**Proof:**

The input is a strongly chordal graph with the vertices numbered in accordance with a SE ordering. The arrays Next, Max, M and L can be easily obtained in $O(n')$ time. Our algorithm starts from step1, which invokes all the processors $P_q$, to check if $(v, M_q) \in E$. If it is true, then any shortest path between $v$ and $M_q$ is of length one. In step 2, all the processors check if $(v, M_q) \not\in E$. If it is true then

$$LSP(v, M_q) = LSP(Next[v], M_q) + 1$$

and

$$S[v, M_q] = Next[v].$$

So we obtain some LSP’s which satisfies the condition $(v, M_q) \in E$ and $(v, M_q) \not\in E$. It is clear from properties, that if $\text{Max}[i,k]$ and $\text{max}[i,k+1]$ are non-NULL, $i<j$, $k$ be the integer satisfying $\text{Max}[i,k] < j < \text{Max}[i,k+1]$, then there exists a shortest path from $i$ to $j$ containing $\text{Max}[i,k]$ and also there exists a shortest path from $i$ to $j$ containing $\text{Max}[i,k+1]$ if $\text{Max}[i,k] \not\in E$. Therefore, in step3, this enables each processor $P_q$ to find the remaining LSP’s for remaining all pairs of vertices using some LSP’s which we found in step1 and step2. Hence we conclude that our CREW-APSL finds the solution correctly.
Theorem 7.6.2:

Given a strongly chordal graph and strong elimination ordering of the graph, the CREW-APSL can be solved in \( O(n') \) time.

Proof:

The arrays Next, Max, M and L can be easily computed in \( O(n') \) time. In step 1 each execution of the innermost 'while' loop inside the 'for' loop takes \( O(n') \) time. In step 2 each execution inside the 'for' loop takes \( O(n') \) time. In step 3, we invoke all the processor to carry out the execution of two 'while' loops, which runs in \( O(n') \) time. Hence the time complexity of the algorithm is \( O(n') \).

Let \( t(n) \) and \( N \) denote the parallel running time and number of processor respectively, the cost \( c(n) \) is defined as

\[
c(n) = t(n) \times N
= O(n') \times O(n')
= O(n'^2).
\]

Hence, the cost \( c(n) = O(n'^2) \).

Corollary:

The CREW-APSL algorithm gains Efficiency 1.

Proof:

Efficiency is a measure employed in the place of speed-up when the optimal sequential algorithm is not known. It is defined as

\[
\text{Efficiency} = \frac{\text{The time complexity of the best known sequential algorithm}}{\text{The cost of the parallel equivalent for the same problem}}.
\]
For APSL problem on strongly chordal graph, the time complexity of the best known sequential algorithm is $O(n^3)$ and the cost of the CREW-APSL algorithm is also $O(n^3)$. Therefore

\[
\text{Efficiency} = \frac{O(n^3)}{O(n^3)} = 1.
\]

Hence, the proof.

**7.7. An Illustration**

Before concluding this chapter, let us see an illustration that exemplifies how the algorithm determines the APSL on strongly chordal graph in parallel. Figure 7.1 represents a strongly chordal graph with strong elimination ordering. The arrays Next, Max, M and L corresponding to figure 7.1 are calculated and shown in section 7.4.

In step 1, $P_1$ takes $M_1 = 10$ and $L_1 = 8$. The value of $v = 10 - 1 = 9$. Here $(9, 10) \in E$. Therefore, $LSP(9, 10) = 1$ and $S[9,10] = 10$. Similarly the second processor $P_2$ takes $M_2 = 9$ and $L_2 = 7$. The value of $v = 9 - 1 = 8$. Here $(8,9) \in E$. Therefore, $LSP(8,9) = 1$ and $S(8,9) = 9$. Similarly, all the processors find the LSP's which satisfies the condition $(v,M_q) \in E$.

In step 2, $P_3$ takes $M_3 = 8$ and $L_3 = 6$. The value of $v = 8 - 1 = 7$. Here $(7,8) \not\in E$. Therefore, $LSP(7,8) = LSP(9,8) + 1 = 1 + 1 = 2$. Similarly, all the LSP's are calculated for the case $(v,M_q) \not\in E$.

In step 3, all the processors find the remaining LSP's and the results of $LSP[i,j]$ and $S[i,j]$ for the graph of figure 7.1. are shown in Table 7.3. and Table 7.4.
<table>
<thead>
<tr>
<th>LSP</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

*Table 7.3. The array LSP for the graph*
<table>
<thead>
<tr>
<th>S</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 7.4. The array $S$ for the graph*
7.8. Discussion

In this chapter, we have dealt with a problem, all-pairs-shortest-length problem, on the class of strongly chordal graphs. The task that we set out to achieve is the solution to shortest-length query problem. The proposed efficient parallel algorithm for APSL problem on the class of strongly chordal graph takes $O(n')$ time employing CREW-PRAM with $O(n')$ processors. The time complexity is exactly matched with the sequential upper bound. Hence it is claimed that it is efficient. However, different approach of paralysing the step 1 to step 3 in our CREW PRAM algorithm may still improve the cost of the algorithm and may lead to removing of the constraints between the problem size and the number of processor.