APPENDIX

PROOF OF THEOREMS

Lemma 1. *For any path $p \in P_{CPPR}^{mode}(k)$ with $p.lauFF \neq p.capFF$ and $depth(LCA(p.lauFF, p.capFF)) = d$, we have $p \in P_d^{mode}(k)$.*

Proof. By contradiction. Suppose $p \notin P_d^{mode}(k)$. Then for any $q \in P_d^{mode}(k)$, we have

$$slack^{mode}(q, d) \leq slack^{mode}(p, d).$$

For path p, because $depth(LCA(p.lauFF, p.capFF)) = d$, we have

$$slack_{\text{CPPR}}^{mode}(p) = slack^{mode}(p, d).$$

For any $q \in P_d^{mode}(k)$, because $depth(LCA(q.lauFF, q.capFF)) \leq d$, we have

$$slack_{\text{CPPR}}^{mode}(q) \leq slack^{mode}(q, d).$$

Combining the equations above, we get

$$slack_{\text{CPPR}}^{mode}(q) \leq slack^{mode}(q, d) \leq slack^{mode}(p, d) = slack_{\text{CPPR}}^{mode}(p).$$

That means every path $q \in P_d^{mode}(k)$ has smaller post-CPPR slack than p. There are a total of k paths in $P_d^{mode}(k)$. Thus, p cannot be ranked top-k in $P_{\text{CPPR}}^{mode}(k)$, which is a contradiction. □

Lemma 2. *For any path $p \in P_{CPPR}^{mode}(k)$ with $p.lauFF = p.capFF$, we have $p \in P_*^{mode}(k)$.*

Proof. By contradiction. Suppose $p \notin P_*^{mode}(k)$. Then for any $q \in P_*^{mode}(k)$, we have

$$slack^{mode}(q, depth(q.lauFF)) \leq slack^{mode}(p, depth(p.lauFF)).$$

Whether or not q is a self-loop path, there must be $depth(q.lauFF) \geq depth(LCA(q.lauFF, q.capFF))$, and thus we have

$$slack_{\text{CPPR}}^{mode}(q) \leq slack^{mode}(q, depth(q.lauFF)).$$

On the other side, p is a self-loop path by our assumption, and thus

$$slack_{\text{CPPR}}^{mode}(p) = slack^{mode}(p, depth(p.lauFF)).$$

Combining the equations above, we get

$$\begin{aligned} slack_{\text{CPPR}}^{mode}(q) &\leq slack^{mode}(q, depth(q.lauFF)) \\ &\leq slack^{mode}(p, depth(p.lauFF)) \\ &= slack_{\text{CPPR}}^{mode}(p). \end{aligned}$$

That means every path $q \in P_*^{mode}(k)$ has smaller post-CPPR slack than p. There are a total of k paths in $P_*^{mode}(k)$. Thus, p cannot be ranked top-k in $P_{\text{CPPR}}^{mode}(k)$, which is a contradiction. □

Lemma 3. *For any path $p \in P_{CPPR}^{mode}(k)$ that originates from a primary input rather than a launching FF, we have $p \in P_{PI}^{mode}(k)$.*

Proof. This one is apparent because every path $q \in P_{\text{PI}}^{mode}(k)$ originates from primary inputs and they are sorted by their post-CPPR slacks. □

Theorem 1. *With all the path candidates, `selectTopPaths` (Algorithm 6) correctly selects and returns global top-k paths ranked by their post-CPPR slacks.*

Proof. Because of Lemma 1, 2 and 3, we have encountered every path that has the potential to become one of the global top-k paths within the execution of Algorithm 6. By filtering out paths that we do not want, every path appears at most once. Thus, the global top-k paths can be obtained by selecting top-k of all kinds of path candidates. □

Lemma 4. *For any circuit pin v whose input edges are $u_1 \to v, u_2 \to v, ..., u_k \to v$, by the definition of $at(v)$ and $at'(v)$, we have*

$$at(v).time = \min_{1 \leq i \leq k} at(u_i).time + delay^{early}(u_i, v),$$

$$at'(v).time = \min_{1 \leq i \leq k} at_{auto}(u_i, at(v).groupid).time + delay^{early}(u_i, v).$$

This ensures that we can correctly propagate the two sets of arrival time tuples using tuples on previous pins. The above statements are for hold check. For setup check, one needs to replace min by max, and early by late.

Proof. The first equation is obvious according to the optimal substructure of shortest path on DAG. For the second one, the right-hand side (RHS) gives a valid solution to the problem defined by left-hand side (LHS), so we must have LHS≤RHS. We assume LHS<RHS and prove by contradiction. Suppose the shortest path given by LHS is from u_i. Then the arrival time of the path at u_i must be smaller than both $at(u_i).time$ and $at'(u_i).time$, and that contradicts the optimality of them. □

Theorem 2. *Procedure `getPathsAtLCALevel` (Algorithm 5) correctly computes $P_d^{mode}(k)$.*

Proof. According to the way the algorithm assigns the arrival time of launching FFs and capturing FFs, $slack(p, d)$ is added to the slack of path p in both modes. For every circuit pin v, the algorithm maintains two sets of arrival time tuples, $at(v)$ and $at'(v)$, the latter of which serves as a fallback for the former. From these two sets of arrival time, according to Lemma 4, we can always find the shortest path (for hold check, longest path for setup check) to v subject to any node grouping contraint. Thus, the algorithm computes top-1 path candidate at level d correctly.

For the correctness of top-k path finding, we represent each path as a list of deviations from a shortest path. Since by definition all paths can be regarded as a list of deviations from a shortest path, it suffices to show that we find paths in ascending order of their slacks. In other words, each deviation introduces a non-negative increase on the slack of a path (the *cost* in Algorithm 5 line 16, 18). For any circuit pin u and group index gid,

$$cost^{\text{hold}} = at_{\text{auto}}(w, gid).time + delay^{\text{early}}(w, u) - at_{\text{auto}}(u, gid).time$$

for hold check and

$$cost^{\text{setup}} = at_{\text{auto}}(u, gid).time - delay^{\text{late}}(w, u) - at_{\text{auto}}(w, gid).time$$

for setup check are non-negative. This is obvious according to Lemma 4 and the definition of at_{auto}. □

Theorem 3. *The overall Algorithm 1 outputs $P_{CPPR}^{mode}(k)$ correctly.*

Proof. The correctness of procedures `getPathsFromSelfLoops` and `getPathsFromPIs` can be proved similarly to Theorem 2. The correctness follows from Theorem 1. □

Theorem 4. *For $k = 1$, Algorithm 1 runs in $O(nD)$ time complexity.*

Proof. The algorithm calls `getPathsAtLCALevel` D times, `getPathsFromSelfLoops` once, and `getPathsFromPIs` once. Each of them consists of a single forward propagation and constant times of enumeration on FFs. Thus, they runs in $O(n)$ time. For $k = 1$, the `selectTopPaths` procedure just selects the path with the smallest slack from at most $D + 2$ paths, so it runs in $O(D)$ time. As a result, the overall algorithm runs in $O(nD)$. □

Theorem 5. *For $k > 1$, Algorithm 1 runs in $O(nDk \log k)$ time complexity.*

Proof. In procedures `getPathsAtLCALevel`, `getPathsFromSelfLoops`, and `getPathsFromPIs`, the propagation of arrival time takes $O(n)$. After that, we pop paths from a min-heap for k times. Each time one path is popped from the heap, and we scan for all its deviations and push them into the heap. The count of deviations from a single path cannot exceed the size of the graph, which is n, and thus there are $O(nk)$ heap operations. By using a min-max-heap, we are able to limit the size of the heap and always keep the smallest k paths in the heap. In this way, each heap operation takes $O(\log k)$. As a result, each of the three procedures runs in $O(nk \log k)$.

The `selectTopPaths` procedure selects the top-k paths from at most $k(D+2)$ paths, which can be done in $O(kD \log k)$. We conclude that the overall algorithm runs in $O(nDk \log k)$. □

Theorem 6. *The algorithm runs with space complexity $O(T(n+k)+kp)$, where T denotes the number of threads, and $p < n$ denotes the average length of critical paths.*

Proof. For every call to `getPathsAtLCALevel`, `getPathsFromSelfLoops`, and `getPathsFromPIs`, we need $O(n)$ of memory to store arrival time tuples for circuit pins. We represent each path as a list of deviation edges. Because deviation edges are added one by one, we do not need to store all of them on a single path. Instead, we arrange them in a prefix tree [2], where each path is denoted by a node and each deviation edge is denoted by an edge, and thus we need $O(k)$ memory to store all the paths. Each thread will have its own dedicated memory for working on a call. Thus, the overall memory complexity is $O(T(n+k))$. The additional $O(kp)$ of memory is the size of the resulting global top-k paths. □