A randomized parallel algorithm for dfa-minimization

Abstract

The problem of finding the coarsest partition of a set S with respect to another partition of S and one or more functions on S has several applications, one of which is the state minimization of finite state automata. The problem has a well known $O(n \log n)$ sequential algorithm. In this paper, we present efficient parallel randomised algorithms for the problem.

Keywords: Parallel-algorithms, Partitioning, DFA minimization

1 Introduction

The **single function coarsest partitioning** problem can be stated as follows:

Problem : *Given a set S of n elements, $a_1, a_2, ..., a_n$, a partition $\pi = (\pi_1, \pi_2, ..., \pi_k)$ of S and a function $f : S \to S$, we wish to find the* **coarsest** *(i.e., having the fewest blocks) partition $\pi' = (\pi'_1, \pi'_2, ..., \pi'_l)$ of S, such that*

1. *$\forall a_i, a_j \in S$, $a_i, a_j \in \pi'_m \Rightarrow \exists p$ such that, $f(a_i), f(a_j) \in \pi'_p$.*
2. *$\forall m, 1 \leq m \leq l, \exists p, 1 \leq p \leq k$, such that $\pi'_m \subseteq \pi_p$.*

The more general version of the problem, is the **multiple function coarsest partition** problem, where we are given c, $c \geq 1$, functions $f_1, f_2, ..., f_c$. All the c functions must satisfy condition 1 above. This problem has several interesting applications, some of the important ones being *minimizing the number of states in a DFA, checking isomorphism of DFAs* and *deciding isomorphism of triply connected planar graphs.* For instance, given a DFA, we start with the initial partition comprising of 2 blocks - the set of final, and non-final states. The functions are the state transitions on each symbol of the input alphabet. The final partition yields the minimum DFA.

The first $O(n \log n)$ algorithm for the problem was given in [1]. This algorithm successively refines the partition, by examining the inverse image of the elements under the given mappings. An alternate $O(n \log n)$ algorithm for the problem was given by Rao et al [5], using an inverted list implementation. Paige et al [4] give an $O(n)$ time algorithm for the single function case. Their approach exploits the structure of the mapping, and does not use the idea of successive refinements. However, their approach does not appear to be useful for handling the more general problem of multiple function partitioning. Parallel algorithms for these problems have been given by Já Já and Kosaraju [2]. Their approach yields an n^2 processor, $O(log^2 n)$ time algorithm for the single function case, and an n^4 processor, $O(log^2 n)$

time algorithm for the multiple function problem. These algorithms run on the CREW PRAM. Their algorithm for the single function case can be implemented on a $\sqrt{n} \times \sqrt{n}$ mesh in $O(\sqrt{n})$ time. As an open question, they ask if there is any parallel algorithm for the problem, that uses only a linear number of processors. More recently, Já Já and Ryu [3] have given efficient parallel algorithms for the single function case. Their algorithm runs on the weakest concurrent write PRAMs in $O(\log n)$ time and requires $O(n \log\log n)$ work. In this paper, we present randomised parallel algorithms for the dfa-minimization problem. The method can be easily generalised to handle the case where we have 3 or more functions. These algorithms use ideas similar to those of Ullman and Yannakakis [8] to do BFS in sparse graphs. Our algorithms run in time $O(n^{2-\alpha+\epsilon})$ using $n^{2\alpha}$ processors, for all $\alpha < 2/(\beta-1)$, where n^β is the complexity of boolean matrix multiplication. The probability of the algorithm giving the correct answer is $1 - 2^{-n^\epsilon}$. We say that an event occurs with *High Probability (HP)*, if the probability of the event occuring is at least $(1-2^{-n^\delta})$, for some $\delta > 0$. Unlike the randomised algorithms given in [6,7,8], we can actually verify whether or not our algorithm has given the correct answer, within the same time and processor bounds; this allows us to repeat the algorithm in case of error.

2 The Randomised Algorithm

As stated earlier, the approach adopted by Hopcroft was to repeatedly split the partitions based on their inverse images under the given mappings. Our approach is to refine the partitions by examining the images of the partitions under the given mappings. For the sake of simplicity, we shall assume that there are exactly 2 functions - f and g. The idea is as follows - for each pair of elements (a_i, a_j), we answer the question - do a_i and a_j belong to the same final partition? This information defines an equivalence relation on S, which gives us the final partition.

It is convenient to view the input as a digraph of out-degree 2. Let $G(V, E)$ be the graph, $V = \{a_1, a_2, ..., a_n\}$, $E = \{(a_i, a_j) | f(a_i) = a_j\} \cup \{(a_i, a_j) | g(a_i) = a_j\}$. We construct the graph $G^2(V^2, E^2)$, where V^2 consists of the n^2 ordered pairs (a_i, a_j), $1 \le i, j \le n$, and $E^2 = \{((a_i, a_j), (f(a_i), f(a_j)))\} \cup \{((a_i, a_j), (g(a_i), g(a_j)))\}$. We shall first present a simple n^2 processor, $O(n^2)$ time parallel algorithm that explains the idea mentioned earlier.

Algorithm Partition

- For each pair (a_i, a_j),

  $Split[a_i, a_j]$:= True, if a_i and a_j belong to different initial partitions,

  $Split[a_i, a_j]$:= False, otherwise.

- Repeat (n^2+1) times

  For all pairs (a_i, a_j), in parallel, do

  $$Split[a_i, a_j] := Split[f(a_i), f(a_j)] \bigvee Split[g(a_i), g(a_j)]$$

  end Repeat

end Partition

Lemma 1 : *Algorithm partition correctly finds the coarsest partition of the given set S, with respect to the initial partition and the two mappings.*

Proof : If a pair (a_i, a_j) must belong to different final partitions, then there must exist a pair (a_k, a_l), such that a_k and a_l belong to different initial partitions, and there is an path from (a_i, a_j) to (a_k, a_l) in G^2. Since this path is of length atmost n^2, and we have explored all paths of length n^2+1 from every vertex in G^2, we would have found this path. $\square$

We shall now present the randomised algorithm. The idea is as follows - we pick n^α vertices in G^2, uniformly at random. The algorithm partition is executed for $n^{2-\alpha+\epsilon}$ steps; we then create a graph $G^{2\prime}$ which has n^α vertices, on which we compute transitive closure. This enables us to know with high probability, which of the randomly picked pairs must belong to different final partitions. We can then use this information to decide which pairs in G^2 must split. These steps are detailed in the algorithm Rand-Partition.

Algorithm Rand-Partition

Choose n^{α} vertices, uniformly at random from G^2. Call these the *chosen* vertices.

We shall say that a vertex u has visited a vertex v, if we have found a path from u to v in G^2. Each vertex keeps track of the chosen vertices that it has visited, and maintains this list as a queue. Initially, the queues of the chosen vertices contain exactly one vertex (itself) and all the other queues are empty. The array *Split* is initialised as in the previous algorithm.

1. Repeat $(n^{2-\alpha+\epsilon})$ times

   For all pairs (a_i, a_j), in parallel, do

   $Split[(a_i, a_j)] := Split[(f(a_i), f(a_j))] \bigvee Split[(g(a_i), g(a_j))]$

   end Repeat

2. Repeat $4(n^{2-\alpha+\epsilon})$ times

   For each vertex (a_i, a_j) in G^2, in parallel, do

   Examine the next $n^{2-2\alpha}/2$ vertices in the queues of $(f(a_i), f(a_j))$ and $(g(a_i), g(a_j))$ respectively. All those vertices that have not been visited by (a_i, a_j) are marked as visited and inserted in the queue maintained by (a_i, a_j).

   end Repeat

3. Construct the graph $G^{2\prime}(V^{2\prime}, E^{2\prime})$ as follows:

   $V^{2\prime}$ consists of the n^{α} chosen vertices. $E^{2\prime}$ is the set $\{(u, v) | u, v \in V^{2\prime}, and \text{ v } has\ been\ visited\ by \text{ u}\}$.

4. Compute the transitive closure of $G^{2\prime}$. For every vertex $u \in V^{2\prime}$, in parallel, do

   If $\exists\ v$, such that v is reachable from u and $(Split[v] = True)$, Then $Split[u] := True$.

5. For every vertex $v \in G^2$, in parallel, do

   If there is a chosen vertex u, such that u is in the queue of v, and $Split[u] = True$, Then $Split[v] := True$.

end Rand-Partition

Let (a_i, a_j) be a vertex in G^2. We shall designate the edge $((a_i, a_j), (f(a_i), f(a_j)))$ as an f-edge, and the edge $((a_i, a_j), (g(a_i), g(a_j)))$ as a g-edge. Thus every edge in G^2 is either an f edge, or a g edge, and every path p, starting at some given vertex v, can be characterised by a sequence of f's and g's, called the *characteristic* sequence of p. We can now lexicographically order all paths starting at v, by ordering the characteristic sequences.

Lemma 2: *For each path p, of length $m = n^{2-\alpha+\epsilon}$, one of the following must be true, with HP.*

1. *There is a chosen vertex on p.*

2. *p contains a cycle.*

3. *There is a lexicographically smaller path q, starting at the same vertex as p, which contains a vertex that is on p.*

Proof: Consider any vertex v, and all paths of length m, starting at v. We shall first prove that for a given path, either (1) or (2) must hold with HP.

If, on this path, we havent repeated a vertex, then this path consists of $n^{2-\alpha+\epsilon}$ distinct vertices. The probability that a chosen vertex was one of these is $n^{-\alpha+\epsilon}$. The process of choosing n^{α} vertices can be viewed as independent bernoulli trials. The sum of these expectations is therefore binomially distributed; applying the Chernoff bounds, we get that each path satisfies the requirement of the lemma.

We shall now show that if we consider all paths of length m, one of the three conditions must hold with HP. Since there are no more than n^2 vertices in G^2, there can be no more than n^2 vertex disjoint paths starting from each vertex. This implies that there are no more than n^4 paths that we have to consider. Adding up the failure probabilities for the events corresponding to each path, we have the required result. □

The following lemma needs no proof.

Lemma 3: *At the end of step 2, every vertex in G^2 knows all the chosen vertices that are at a distance of no more than $n^{2-\alpha+\epsilon}$ from itself. In fact, all these chosen vertices will be the ones in its queue at the end of step 2.*

Lemma 4 : *For every chosen vertex (a_i, a_j), such that a_i and a_j must belong to different final partitions, $Split[(a_i, a_j)]$ is True at the end of step 4. (with HP)*

Proof: If a_i and a_j must belong to different final partitions, then there exists an acyclic path of length $\leq n^2$ from (a_i, a_j) to a pair (a_k, a_l) such that a_k and a_l belong to different initial partitions. Step 1 of the algorithm traverses this path for a distance $n^{2-\alpha+\epsilon}$, and as shown in lemma 2, we encounter a chosen vertex with HP. Now, lemma 2 holds for every chosen vertex along this path. Since we have computed the transitive closure of $G^{2\prime}$, we know that there is a path from (a_i, a_j) to (a_k, a_l). Therefore the result of the lemma follows. □

The next lemma can be proved using a similar argument.

Lemma 5: *For every vertex (a_i, a_j), such that a_i and a_j must belong to different final partitions, $Split[(a_i, a_j)]$ is True at the end of the algorithm.* □

Theorem 6: *Algorithm Rand-Partition correctly computes the coarsest partition of the set S with respect to f and g using $n^{2\alpha}$ processors in time $O(n^{2-\alpha+\epsilon})$, with HP.*

Proof: Each execution of the loop in steps 1 and 2 can be carried out in $O(1)$ time. It is straightforward to see that steps 3 and 5 can be carried within the claimed time and processor bounds. Step 4 can be carried out in polylog time using n^{β} processors, and therefore in $O(n^{2-\alpha+\epsilon})$ steps using $n^{2\alpha}$ processors for $\alpha \leq 2/(\beta - 1)$. $\square$

The output of the algorithm is correct, if for every pair (a_i, a_j) that belong to the same final partition, the pairs $(f(a_i), f(a_j))$ and $(g(a_i), g(a_j))$ also belong to the same final partition. Since this can be verified in $O(1)$ time using n^2 processors, we can check whether or not the algorithm has the correct answer, repeating in case of error.

References

[1] A.V.Aho, J.E.Hopcroft and J.D.Ullman, The design and analysis of computer algorithms, Addison-Wesley, 1974.

[2] J.Já Já and S.R.Kosaraju, Parallel algorithms for planar graph isomorphism and related problems, IEEE trans. on circuits and systems, vol 35, 3, March 1988, pp 304-310.

[3] J.F. JáJá and K.W.Ryu, An Efficient Algorithm for the Single Function Coarsest Partition Problem, to appear in SPAA 1993.

[4] R.Paige, R.E.Tarjan and R.Bonic, A linear time solution to the single function coarsest partition problem, Theoretical computer science, 40, 1985, pp67-84.

[5] N.Rao, R.Sridhar and S.S.Iyengar, An $O(n \log n)$ set refinement algorithm with applications, Intl journal of computer math, vol 40, 1991, pp 129-138.

[6] Sairam S. and Klein P., A randomised parallel approximation scheme for shortest paths, in Proceedings of 24th annual ACM STOC, pp 750-758, 1992.

[7] Sarnath R., A randomised NC algorithm for finding the longest upsequence, TR no. 93-08, dept of computer science, SUNY, Buffalo, NY 14260.

[8] Ullman J.D. and Yannakakis M., High probability parallel transitive closure algorithms, SIAM J comput, vol 20, no 1, pp 100-125, Feb 1991.