A Parallel DFA Minimization Algorithm

Ambuj Tewari, Utkarsh Srivastava, and P. Gupta

Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur 208 016, INDIA
pg@iitk.ac.in

Abstract. In this paper, we have considered the state minimization problem for Deterministic Finite Automata (DFA). An efficient parallel algorithm for solving the problem on an arbitrary CRCW PRAM has been proposed. For n number of states and k number of inputs in Σ of the DFA to be minimized, the algorithm runs in $O(kn \log n)$ time and uses $O(\frac{n}{\log n})$ processors.

1 Introduction

The problem of minimizing a given DFA (Deterministic Finite Automata) has a long history dating back to the beginnings of automata theory. Consider a deterministic finite automaton M as a tuple $(Q, \Sigma, q_0, F, \delta)$ where Q, Σ, $q_0 \in Q$, $F \subseteq Q$ and $\delta : Q \times \Sigma \to Q$ are a finite set of states, a finite input alphabet, the start state, the set of accepting (or final) states and the transition function, respectively. An input string x is a sequence of symbols over Σ. On an input string $x = x_1 x_2 \dots x_m$, the DFA visits a sequence of states $q_0 q_1 \dots q_m$ starting with the start state by successively applying the transition function δ. Thus $q_{i+1} = \delta(q_i, x_{i+1})$ for $0 \leq i \leq m-1$. The language $L(M)$ accepted by a DFA is defined as the set of strings x that takes the DFA to an accepting state, i.e. x is in $L(M)$ if and only if $q_m \in F$. Two DFAs are said to be equivalent if they accept the same set of strings.

The problem of DFA minimization is to find a DFA with the minimum number of states which is equivalent to the given DFA. A fundamental result in automata theory states that such a minimal DFA is unique up to renaming of states. The number of states in the minimal DFA is given by the number of equivalence classes in the partition on the set of all strings in Σ^* defined as follows: Two strings x and y are equivalent if and only if for all strings z, $xz \in L(M)$ if and only if $yz \in L(M)$ [6].

Besides being widely studied, DFA has many applications in the diverse fields like pattern matching, optimization of logic programs, protocol verification and specification and modeling of finite state systems [14]. It is known that nondeterministic finite automata (NFA) are equivalent to deterministic ones as far as the languages recognized by them are concerned. Huffman [4] and Moore [10] have presented $O(n^2)$ algorithms for DFA minimization and are sufficiently fast for most of the classical applications. However, there exist numerous algorithms

S. Sahni et al. (Eds.) HiPC 2002, LNCS 2552, pp. 34–40, 2002.

which are variations of the same basic idea. An efficient $O(n \log n)$ algorithm is due to Hopcroft [5]. Blum [2] has also proposed a simpler algorithm with same time complexity.

Traditional applications of the DFA minimization algorithm involve a few thousand states and the sequential algorithms available generally perform well in these settings. But if the number of states in a DFA is of the order of a few millions, then the efficient sequential algorithms may take a significant amount of time and may need much more than the available physical memory. One way to achieve the speed-up is the use of multiple processors.

DFA minimization has been extensively studied on many parallel computation models. Jaja and Kosaraju [7] have presented an efficient algorithm on mesh connected computer for the case when $|\Sigma| = 1$. A simple NC algorithm has also been outlined in their work. Cho and Huynh [3] have proved the problem to be NLOGSPACE-complete. Efficient and close to cost-optimal algorithms are known only for the case of the alphabet consisting of a single input symbol. A very simple algorithm is proposed by Srikant [13] but the best algorithm for this problem is due to Jaja and Ryu [8]. It is a CRCW algorithm with time complexity $O(\log n)$ and cost $O(n \log \log n)$. Unfortunately, no efficient algorithm, which is also economical with respect to cost, is known for the general case of multiple input symbols. The standard NC algorithm for this problem requires $O(n^6)$ processors. This is due the use of transitive closure computation on a cross product graph. In [12], a simple parallel algorithm for DFA minimization along with its implementation is presented.

In this paper we have proposed an efficient parallel algorithm for DFA minimization having $O(kn\log n)$ time complexity on an arbitrary CRCW PRAM with $(\frac{n}{\log n})$ processors. The rest of the paper is organized as follows. Section 2 contains some preliminaries along with the naive sequential algorithm. In Section 3, we discuss a fast parallel algorithm for the problem which uses a large number of processors. An efficient parallel algorithm has been proposed in Section 4. Conclusion is given in the last section.

2 Review

The DFA minimization problem is closely related to the coarsest partitioning problem which can be stated as follows. We are given a set Q and its initial partition into m disjoint sets $\{B_0, \ldots, B_{m-1}\}$, and a collection of functions, $f_i : Q \to Q$. We have to find a coarsest partition of Q, say $\{E_1, \ldots, E_q\}$, such that: (1) each E_i is a subset of some B_j , and (2) the partition respects the given functions, i.e. $\forall j$, if a and b both belong to the same E_i then $f_j(a)$ and $f_j(b)$ also belong to the same E_k for some k.

Assume, Q is the set of states, f_i is the restriction of the transition function δ to the ith input symbol, i.e. $f_i(q) = \delta(q, a_i)$, the initial partition contains two sets, namely F and $Q - F$. The size of the minimal DFA is the number of equivalence classes in the coarsest partition. For the general case of multiple function coarsest partition problem, an $O(n \log n)$ solution is given in [1]. Later, a linear

Sequential Algorithm

1. *for all* final states q_i *do* block_no$[q_i] = 1$
2. *for all* non-final states q_i *do* block_no$[q_i] = 2$
3. *do*
4. *for* $i = 0$ *to* k-1 *do*
5. *for* j $= 0$ *to* n-1 *do*
6. $b_1 =$ block_no$[q_j]$
7. $b_2 =$ block_no$[\delta(q_j, x_i)]$
8. label state q_j with (b_1, b_2)
9. *endfor*
10. *Assign* same block number to states having same labels
11. *endfor*
12. *while* <number of blocks is changing>

Algorithm 1: The sequential algorithm for DFA minimization

time solution has been proposed in [11] for the case of the single function coarsest partition problem. The single function version of the problem corresponds to having only a single symbol in the input alphabet Σ.

But the simplest sequential algorithm for solving the problem, given in Algorithm 1, runs in $O(kn^2)$ time, where $|\Sigma| = k$ [1]. It performs as follows. Initially there are only two blocks: one containing all the final states and the other containing all the non-final states. If two states q and q' are found in the same block such that for some input symbol a_i, the states $\delta(q, a_i)$ and $\delta(q', a_i)$ are in different blocks, q and q' are placed in different blocks for the next iteration. The algorithm is iterated for at most n times because, in the worst case, each state will be in a block containing just itself. In each iteration new block numbers are assigned in $O(kn)$ time. Therefore, the total time taken is $O(kn^2)$.

3 A Fast Parallel Algorithm

The fastest known algorithm for the multiple input symbol case is due to Cho and Huynh [3]. The DFA minimization problem is initially translated to an instance of the multiple function coarsest partition problem to yield the set S of states, the initial partition B containing sets of final and non-final states and the functions f_i which are restrictions of the transition function to single input symbols. A graph $G =< V, E >$ may be generated as follows: $V = \{(a, b) | a, b \in S\}$ and $E = \{((a, b), (c, d)) | c = f_i(a),\ d = f_i(b)$ for some $i\}$.

For any pair $x, y \in S$, x and y get different labels in the coarsest partition if and only if there is a path from node (x, y) to some node (a, b) such that a and b have different B-labels. The algorithm is given below. It can be shown that the algorithm can be implemented on a EREW in time $O(\log^2 n)$ with total cost $O(n^6)$. The bound $O(n^6)$ arises because of the transitive closure computation of a graph with n^2 nodes. So far, it has not been possible to find an algorithm with a reasonable cost, say $O(n^2)$, and a small running time.

Parallel Algorithm

1. *Construct* graph G as defined above
2. *Mark* all nodes $(p, q) \ni p$ and q belong to different sets of the B-partition
3. *Uses* transitive closure to mark pairs reachable from marked pairs
4. *comment* Note that all unmarked pairs are equivalent

Algorithm 2: A fast parallel algorithm for DFA minimization

4 An Efficient Algorithm

In this section we have proposed a parallel version of the simple $O(n^2)$ time sequential algorithm outlined in Section 3. We use $O(\frac{n}{\log n})$ processors to achieve an expected running time of $O(\log n)$. Using an arbitrary-CRCW PRAM, we have parallelized the inner for-loop which iterates over the set of states. The the new labels obtained in this for-loop are hashed using parallel hashing algorithm of Matias and Vishkin [9] to get new block numbers for the states. The algorithm is given below.

Lines 1-8 are the same as in the sequential algorithm except that Lines 5-8 of Algorithm 3 are now done in parallel. The n labels obtained in Line 8 are hashed to $[1..O(n)]$ using the hashing technique due to Matias and Vishkin [9].

Theorem 1 (Parallel Hashing Theorem). *Let W be a multiset of n numbers from the range $[1..m]$, where $m + 1 = p$ is a prime. Suppose we have $\frac{n}{\log n}$*

New Parallel Algorithm

1. *Initialize* block_no array
2. *do*
3. *for* $i = 0$ *to* k-1 *do*
4. *for* $j = 1$ *to* $\frac{n}{\log n}$ *do in parallel*
5. *for* $m = (j-1) * \log n$ *to* $j * \log n - 1$ *do*
6. $b_1 =$ block_no$[q_m]$
7. $b_2 =$ block_no$[\delta(q_m, a_i)]$
8. label state q_m with (b_1, b_2)
9. *endfor*
10. *endfor*
11. *Use* parallel hashing to map the n labels to $[1..O(n)]$
12. a number to which a state's label gets mapped to its new block_no
13. *Reduce* the range of block_no from $O(n)$ to n
14. *endfor*
15. *while* number of blocks is changing

Algorithm 3: New parallel algorithm for DFA minimization

processors on an arbitrary-CRCW PRAM. A one-to-one function $F : W \longmapsto [1..O(n)]$ *can be found in* $O(\log n)$ *expected time. The evaluation of* $F(x)$ *for each* $x \in W$*, takes* $O(1)$ *arithmetic operations (using numbers from* $[1..m]$*).*

We have to assign new block numbers to states such that states with different labels get different block numbers and states with same label get same block number. Also we do not want the block numbers to become too large. The labels are pairs of the form (b_1, b_2) where $b_1, b_2 \leq n$. Therefore we can map a label (b_1, b_2) to $b_1 * (n+1) + b_2$ and we can treat these labels as numbers in the range $[1..2n^2]$. A number m such that $2n^2 < m \leq 4n^2$ and $m + 1$ is a prime can be found in $O(\log n)$ time (see [9]) and this has to be done just once. For instance, it can be done after the initialization phase of step 1.

We want the block numbers to remain in the range $[1..n]$. However, after hashing, we get numbers in the range $[1..Kn]$ some fixed K. This range shrinking can be implemented on an arbitrary-CRCW PRAM in time $O(\log n)$. The procedure is given as Algorithm 4.

First, each processor hashes $\log n$ labels and sets $PRESENT[x]$ to 1 if some label got hashed to the value x, where $PRESENT[1..Kn]$ is an array. Several processors might try to write in the same location in the array but since we have assumed an arbitrary-CRCW PRAM, one of them will succeed arbitrarily. Each of the $\frac{n}{\log n}$ processors now considers a range of $K \log n$ indices of the $PRESENT$ array and computes the number of locations which are set to 1 using $O(\log(\frac{n}{\log n}))$ prefix sum algorithm. New block number for a given location is simply the number of 1's occurring before that location. A processor can easily compute the new block number for a location in its range by adding the number of 1's occurring before that location within the processor's range to the number of 1's occurring in earlier ranges (this has already been computed by prefix-sum).

To find the time complexity, let us consider first Algorithm 4. From the parallel hashing theorem we know that evaluation of the hashing function in line 3 takes $O(1)$ time. Line 4 is an assignment and so the loop in Lines 2-5 takes $O(\log n)$ time. Similarly the loop in Lines 9-11 takes $O(\log n)$. Prefix sum computation in Lines 13-14 also takes $O(\log n)$ time. The loop in Lines 17-21 takes $O(\log n)$ time. Evaluation of the hash function at line 25 takes $O(1)$ time. Therefore, the last loop (Lines 24-27) too takes $O(\log n)$ time.

The outermost for-loop (Lines 3-14) of Algorithm 3 runs exactly k times where k is the size of the input alphabet and the outer do-while-loop (Lines 2-15) of our algorithm can run for at most n times since the minimal DFA does not have more than n states. Therefore, the expected time complexity of our algorithm is $O(kn \log n)$. Since we use $O(\frac{n}{\log n})$ processors, the cost is $O(kn^2)$ which is the cost optimal parallel adaptation of the $O(kn^2)$ sequential method.

5 Conclusion

In this paper, we have considered a well-known problem from classical automata theory and have presented a parallel algorithm for the problem. We have essentially adapted the naive $O(kn^2)$ sequential algorithm and have shown that our

algorithm requires $O(n \log n)$ time using $O(\frac{n}{\log n})$ processors. Thus it is a cost optimal parallelization on an arbitrary-CRCW PRAM.

Finally, it will be of immense theoretical and practical importance to come up with impossibility results about the limited parallelizability of the sequential DFA minimization algorithms.

```
// Initialize the PRESENT[1..Kn] array
1.  for i = 1 to n/log n do in parallel
2.      for j = (i − 1) ∗ K log n + 1 to i ∗ K log n do
3.          Let x be the value to which the label of q_j hashes
4.          PRESENT[x] = 1
5.      endfor
6.  endfor

// Compute number of 1's in each processor's range
7.  for i = 1 to n/log n do in parallel
8.      a_i = 0
9.      for j = (i − 1) ∗ K log n + 1 to i ∗ K log n do
10.         a_i = a_i + PRESENT[j]
11.     endfor
12. endfor

//Compute partial sums
13. s_0 = 0
14. Compute s_i = Σ_{k=1}^{i} a_i for 1 ≤ i ≤ n/log n using prefix sum

//Compute new block numbers
15. for i = 1 to n/log n do in parallel
16.     a_i = s_{i−1}
17.     for j = (i − 1) ∗ K log n + 1 to i ∗ K log n do
18.         a_i = a_i + PRESENT[j]
19.         if PRESENT[j] = 1 then
20.         new_block_no[j] = a_i
21.     endfor
22. endfor

// Update the block_no array with the new block numbers
23. for i = 1 to n/log n do in parallel
24.     for j = (i − 1) ∗ log n to i ∗ log n − 1 do
25.         Let x be the value to which the label of q_j hashes
26.         block_no[q_j] = new_block_no[x]
27.     endfor
28. endfor
```

Algorithm 4: Reducing the range of block numbers

References

[1] Aho A. V., Hopcroft J. E. and Ullman J. D.: The design and analysis of computer algorithms. Addison-Wesley, Reading, Massachusetts (1974) 35, 36

[2] Blum N.: An $O(n \log n)$ implementation of the standard method of minimizing n-state finite automata. Information Processing Letters **57** (1996) 65-69 35

[3] Cho S. and Huynh D. T.: The parallel complexity of coarsest set partition problems. Information Processing Letters **42** (1992) 89-94 35, 36

[4] Huffman D. A.: The Synthesis of Sequential Switching Circuits. Journal of Franklin Institute **257** (1954) 161-190 34

[5] Hopcroft J. E.: An $n \log n$ algorithm for minimizing states in a finite automata. Theory of Machines and Computation, Academic Press (1971) 189-196 35

[6] Hopcroft J. E. and Ullman J. D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading, Massachusetts (1979) 34

[7] Jaja J. and Kosaraju S. R.: Parallel algorithms for planar graph isomorphism and related problems. IEEE Transactions on Circuits and Systems **35** (1988) 304-311 35

[8] Jaja J. and Ryu K. W.: An Efficient Parallel Algorithm for the Single Function Coarsest Partition Problem. Theoretical Computer Science **129** (1994) 293-307 35

[9] Matias Y. and Vishkin U.: On parallel hashing and integer sorting. Journal of Algorithms **4** (1991) 573–606 37, 38

[10] Moore E. F.: Gedanken-experiments on sequential circuits. Automata Studies, Princeton University Press (1956) 129-153 34

[11] Paige R., Tarjan R. E. and Bonic R.: A linear time solution to the single function coarsest partition problem. Theoretical Computer Science **40** (1985) 67-84 36

[12] Ravikumar B. and Xiong X.: A parallel algorithm for minimization of finite automata. Proceedings of the 10th International Parallel Processing Symposium, Honululu, Hawaii (1996) 187-191 35

[13] Srikant Y. N.: A parallel algorithm for the minimization of finite state automata. International Journal Computer Math. **32** (1990) 1-11 35

[14] Vardi M.: Nontraditional applications of automata theory. Lecture Notes in Computer Science, Springer-Verlag **789** (1994) 575-597 34