Chapter 1
A Scalable Divide-and-Conquer Parallel Algorithm for Finite State Automata and Its Applications*

Z. George Mou† Sevan G. Ficici†

Abstract

Finite state automata (FSA) have been used to model dynamic systems found in many areas. They are also the building blocks of cellular automata. We present a new scalable divide-and-conquer alogrithm for the parallel simulation of FSAs that is fast and efficient, regardless of the relation between input size and the number of processors, and discuss its application to sequential circuit and queuing system analysis.

1 Description

The input string to a FSA is, by definition, read sequentially. The insight needed to overcome this seemingly inherent limitation is that each input symbol can be viewed as a function, mapping one state to another, thus reducing the problem to one of applying a parallel reduction or scan using functional composition as the binary operator [1], [4]. To simulate a finite state automaton M operating on input string I of length n by p processors, I is first evenly partitioned over the p processors. The algorithm proceeds in three stages.

Assuming $n >> p$, each processor first applies a serial scan to its local segment of n/p input symbols. This requires $O(nm/p)$ parallel time, where m is the number of states of M, and no inter-processor communication. Next, the resultant p composite functions are further composed by

*This work was partially supported by NSF grant CCR-89 43333.

†Dept. of Comp. Science, National Center for Complex Systems, Brandeis University, Waltham, MA.

divide-and-conquer (DC) [3]. The computation for this global scan takes $O(m \log(p))$ time on any parallel machine, whereas the communication time depends on the topology of the machine. Optimal DC mappings exist such that commumication cost is exactly equal to the diameter of the network [5], which in the case of a k dimensional mesh is $k(p^{1/k} - 1)$. Finally, each processor accumulates the composite functions applied to it during the DC scan, and adjusts the first $(n/p) - 1$ composite functions of its local segment accordingly. If one is concerned only with the final state and output of the FSA, this third step is omitted and a reduction is substituted for the scan in stages one and two. The total complexity of the algorithm is thus:

$$T(n) = O(nm/p + m \log(p) + k(n^{1/k} - 1)) \tag{1}$$

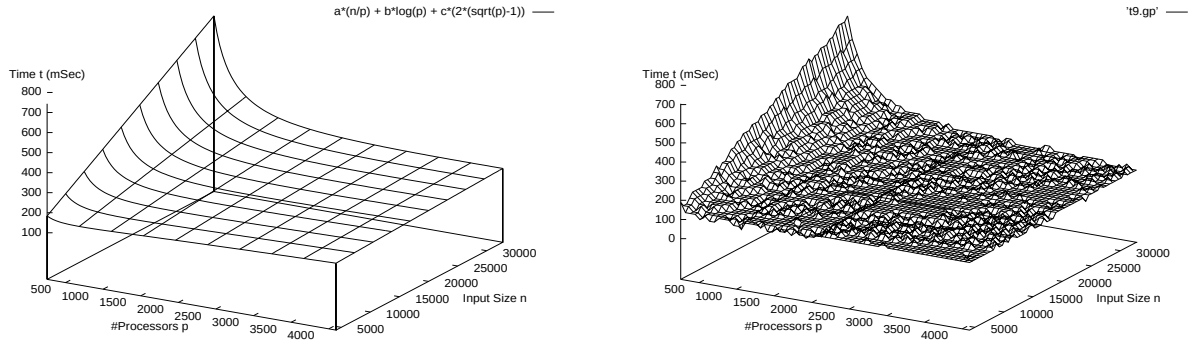
Switching between optimal serial and parallel methods gives our algorithm superior performace compared to the use of recursive doubling or odd-even reduction throughout, neither of which have optimal mappings on any $k \geq 2$ dimensional mesh [2]. Particularly, when $n = O(p)$, odd-even reduction has a larger constant factor.

2 Applications

Sequential circuits are readily described as FSAs. Thus, the algorithm provides a simulator where the parallelism is unaffected by the vagaries of a circuit's structure. To simulate a finite length queue, the size of a FSA is simply the size of the queue + 2 (empty queue/free server + empty queue/busy server). The input consists of time-stamped arrival and departure events (symbols), which encode a queue length and server status for each state at an instant of time. The composition of two input events represents a span of time, transforming the scalar values of the earlier event into curves via multiplication by time. Subsequent DC steps pair composite events and sum the areas of their respective curves, as well as calculate the curves between them. Thus, a second, superimposed scan operation calculates a running integration of curves that depict queue length and server status over time, making calculation of server utilization, average wait time, etc., possible.

3 Benchmarks

Figure 1 depicts idealized and actual performace of the algorithm, and indicates an excellent fit. As predicted, for most values of n and p, the nm/p



(a) (b)

Figure 1: (a) Theoretical analysis and (b) benchmark data from MasPar MP-2 with 4096 processors. p is number of processors, n is input size, and t is time in mSecs. For large values of n, a virtually linear speedup is achieved as p increases.

term dominates, and a speedup virtually linear to the number of processors used is achieved. For small values of n and p, however, DC computation and communication times dominate: benchmarks indicate that p=8 for n=32 is optimal at 23 mSecs; timings for p=16 and p=32 are 46 mSecs and 70 mSecs, respectively. This behavior is still sub-optimal and predicted by our model.

References

[1] R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *JACM*, (27/4), 831-838.

[2] Z. G. Mou and M. Goodman, "A Comparison of Communication Costs for Three Parallel Programming Paradigms on Hypercube and Mesh Architectures", *Proc. 5th SIAM Conf. on Parallel Processing*, 491-500.

[3] Z. G. Mou and P. Hudak, "An Algebraic Model for Divide-and-Conquer and Its Parallelism", *The Journal of Supercomputing*, (2/3), 257-278.

[4] W. D. Hillis and G. L. Steele, Jr., "Data Parallel Algorithms", *CACM*, (29/12), 1170-1183.

[5] Z. G. Mou and X. Wang, "Optimal Mappings of m Dimensional FFT Communication to k Dimensional Mesh for Arbitrary m and k", *Lecture Notes in Comp. Science*, (694), 104-119.