EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 10: September 27

Lecturer: Vijay Garg *Scribe: Prateek Srivastava*

10.1 Introduction

This lecture covers following items.

- Project outlines
- Merging Puzzle
- Sequential Consistency

10.2 Projects

Projects need to be done in groups of 2/3. Undergrads should do project with other undergrads and Grads with grads. For undergrads, the project would be to implement and turn in just one plot of the observations. For grads, they need to pick a topic, implement it and possibly improve on it. Topics can be chosen from recent proceedings as well. If undergrads choose same topics they can compete among themselves for the performance. Though grading won't depend much on the competition. Topics are posted on canvas. A brief overview is present here.

- Mutex algorithms not covered in class can be implemented e.g. colored bakery
- Monitors with additional feature like abort.
- Parallel work scheduling and work distribution algorithms using for e.g. language CILK.
- Concurrent trees, queues and skiplist. No serialization by locking.
- Concurrent Hash-tables like Cuckoo Hashing, and Hopscotch Hashing
- Poset lattice theory. some part will be covered in class.
- Graph shortest path, spanning trees, max flows can be implemented on stampede GPU.
- Sorting implement and compare performance
- Image processing on GPU, scientific computation like fft and polynomial.Data mining algorithms like K-means.
- Text analysis used heavily by Google, pattern matching.
- Iphone/Android for this topic need to give exact details on what the app is going to do.
- Run model checker on algo and verify correctness

- Lamport's Temporal Logic of Actions for specifying and verifying the correctness of concurrent algorithms
- Verification of Consistency Conditions

Please note that the project should contain some references. The naive and obvious solution to All Pairs Shortest Path(APSP) problem is to run a Single Source Shortest Path algorithm from each starting vertex v. If the graph has arbitrary edge weights, it takes the Bellman-Ford algorithm $O(|E||V|^2)$ time to solve APSP. But there are better approaches.

10.3 Merge Puzzle contd...

10.3.1 Problem

Merge two sorted arrays of length n to form a sorted array of length 2n

10.3.2 Solutions

Sequential algorithm was like the merge sort. Parallel algorithm computes rank for each element which is the index into the final merged array. Each element computes its rank using its own position and its position in the other array using binary search. Following table summarizes the time and work complexity of already discussed solution.

	Time	Work
Sequential	O(n)	O(n)
Parallel	O(log n)	O(n log n)

10.3.3 Work optimal parallel solution

We would like to reduce the work from $O(n * logn)$. This can be achieved using cascaded algorithm. Divide the input array in log n groups. The starting element in each group is called the splitter.
No. of splitters = $O\left(\frac{n}{logn}\right)$
Fill the splitter in the target array by finding ranks as done in the parallel algorithm.
Find the sublist in α and β such that they are in between the splitters.
Number of such lists = $O\left(\frac{n}{logn}\right)$
Size of each list = $Ologn$)
We know that the $logn$ size lists can be merged in $Ologn$)

Hence, following table summarizes the steps.

	Time	Work
Step 1	O(log n)	O(n)
Step 2	O(log n)	O(n)
Total	O(log n)	O(n)

10.4 Parallel prefix sum puzzle

Given an array find an output array such that each element in output array is sum of input array elements till that index.
e.g. Input 3, 4, 19, 11, 13
Output 3, 7, 26, 37, 50
The algorithm is called scan.

10.5 Consistency condition

Consider a stack with following operations push(40) push(9) pop (Pop should return 9 not 40) Sequential correctness Now consider following situation.

The correctness of output depends on the definition of correct output. Lamport wrote a 2 page paper explaining what it means to be sequential consistent in a multiprocessor environment.

10.5.1 Notations

A method call is split into two events

* *Invocation* method name + args eg f.foo(arg1, arg2)
* *Response* result or exception

P f.foo(arg1, arg2)
this is an invocation of object f on thread P
foo is method name
arg1. arg2 are arguments
Thread P f.response
This is the return value.

We define following inv(e) is invocation of e resp(e) is response of e proc(e) is process on which e runs *History* A sequence of invocations and responses. History $(H, <_H)$ is set of operations in real time order.

$e <_H f$ if resp(e) occurred before inv(f)

e overlaps with f or is concurrent with f

$< H$ is irreflexive (reflexive) transitive This two conditions imply that it is asymmetric. $(H, <_H)$ is a partial ordered set (poset).

A sequential history is legal if it satisfies sequential specs of the objects.

References

[AGM97] N. Alon, Z. Galil and O. Margalit, On the Exponent of the All Pairs Shortest Path Problem, *Journal of Computer and System Sciences* **54** (1997), pp. 255–262.

[F76] M. L. Fredman, New Bounds on the Complexity of the Shortest Path Problem, *SIAM Journal on Computing* **5** (1976), pp. 83-89.