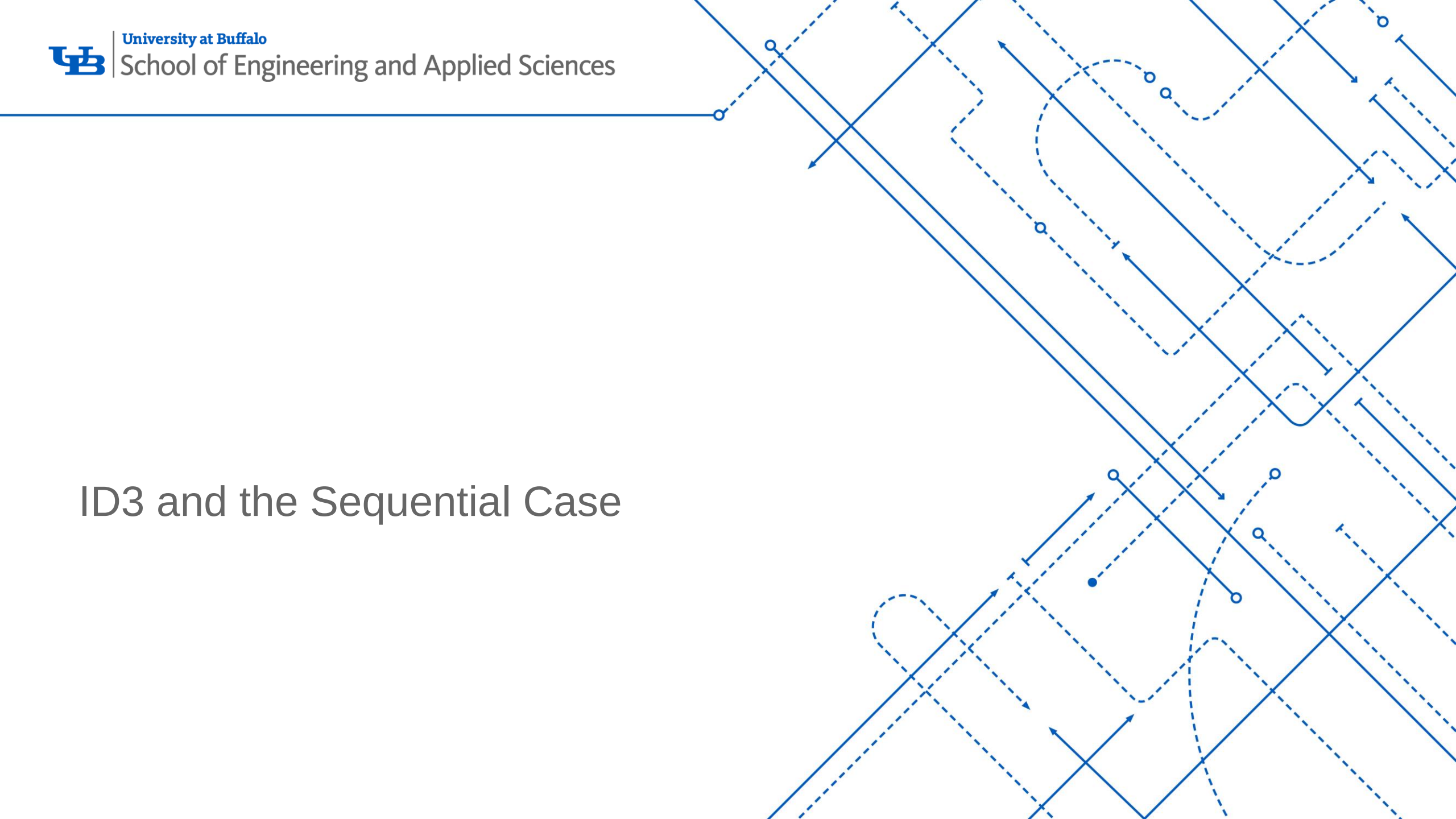
PARALLEL ID3

Jeremy Dominijanni

CSE633, Dr. Russ Miller

University at Buffalo
School of Engineering and Applied Sciences

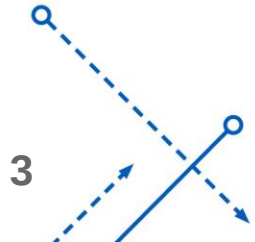
University at Buffalo
School of Engineering and Applied Sciences



ID3 and the Sequential Case

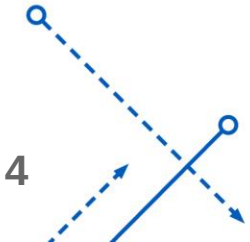
ID3

- Decision tree classifier
- Works on *k-ary* categorical data
- Goal of ID3 is to maximize information gain at each split
  - Same as minimizing the difference in entropy before and after splitting on a feature
- Produces a tree which represents each class as a disjunction of conjunctions (of features)
- Can think of it as producing multiple DNF expressions, one for each class

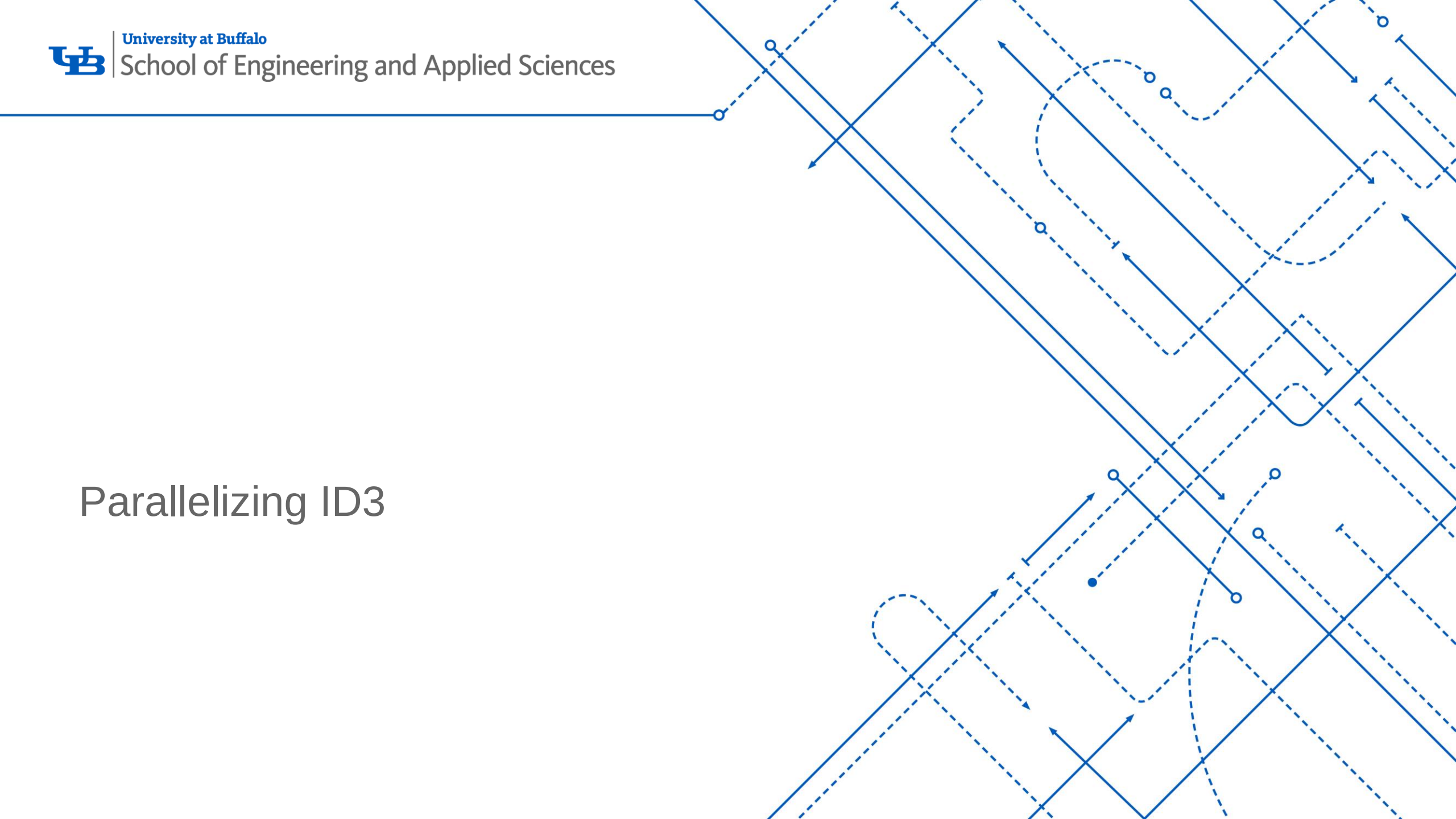


Sequential Performance Analysis (Worst-Case)

- Given D dimensional data with each dimension taking k categories
- Number of nodes $N = \frac{k^{D+1}-1}{k-1}$
- Number of observations $M = k^D$
- Since M and N are both of the same order, we can say performance is
- $O(NM) = O(N^2)$



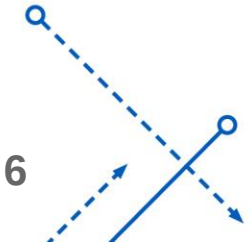
University at Buffalo
School of Engineering and Applied Sciences

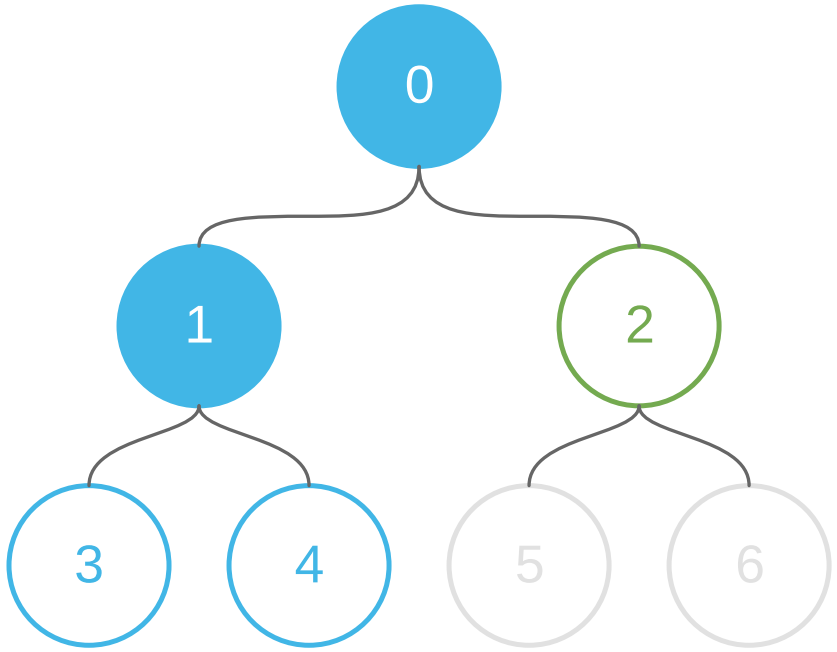


Parallelizing ID3

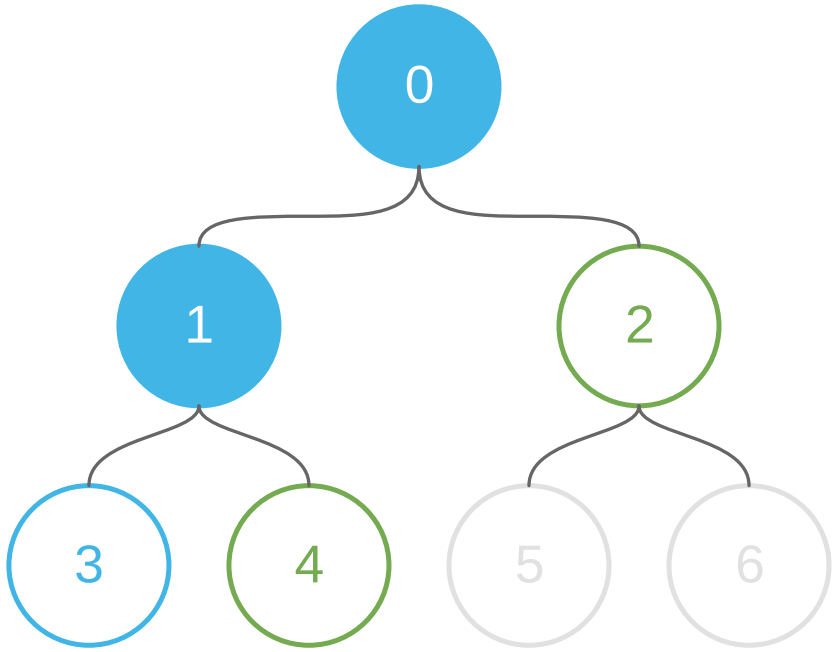
Parallelization of ID3

- “Divide and conquer” is the obvious approach, but it has a major flaw
- If the data is unbalanced, then a large amount of CPU time is wasted
- Solution: add a master processor and send out individual tasks to free processors whenever it is possible to do so
- Overhead is larger than divide and conquer with one processor forced to delegate, but tree imbalance no longer affects speedup
- Implemented using Intel MPI and C++ 11





Divide & Conquer Approach (Four Steps)



Master & Workers Approach (Three Steps)

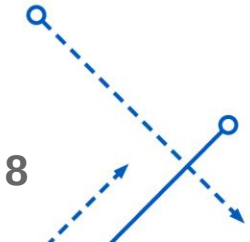
Master-Worker ID3 Algorithm

Master:

compute the topology of the perfect binary tree (maximum size scenario)
for-each un-computed node, if there are no unresolved dependencies and there is a free worker,
send feature restrictions to worker, else break
wait for response from worker, on response, add new information to model, remove node from
working list, and if it's a leaf, remove descendants from un-computed list
goto for-each until completion

Worker:

wait for feature vector from master
remove invalid observations from list of all observations
compute correct ID3 decision
send decision to master



Worst-Case Performance Analysis of Master-Worker ID3

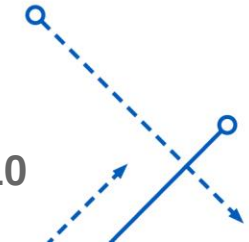
- Pre-processing work done by master $O(Nlog_k N)$
- Time to send tasks to workers $O(logP) \approx O(1)$
- Time to process data from workers $O\left(log_k^2 N\right)$

- Reduce the set of observations $O(DM) \approx O(N)$
- Maximum time to compute ID3 decision $O(DM) \approx O(N)$

- Overall theoretical time complexity $O\left(Nlog_k N + \frac{N^2}{P}\right)$

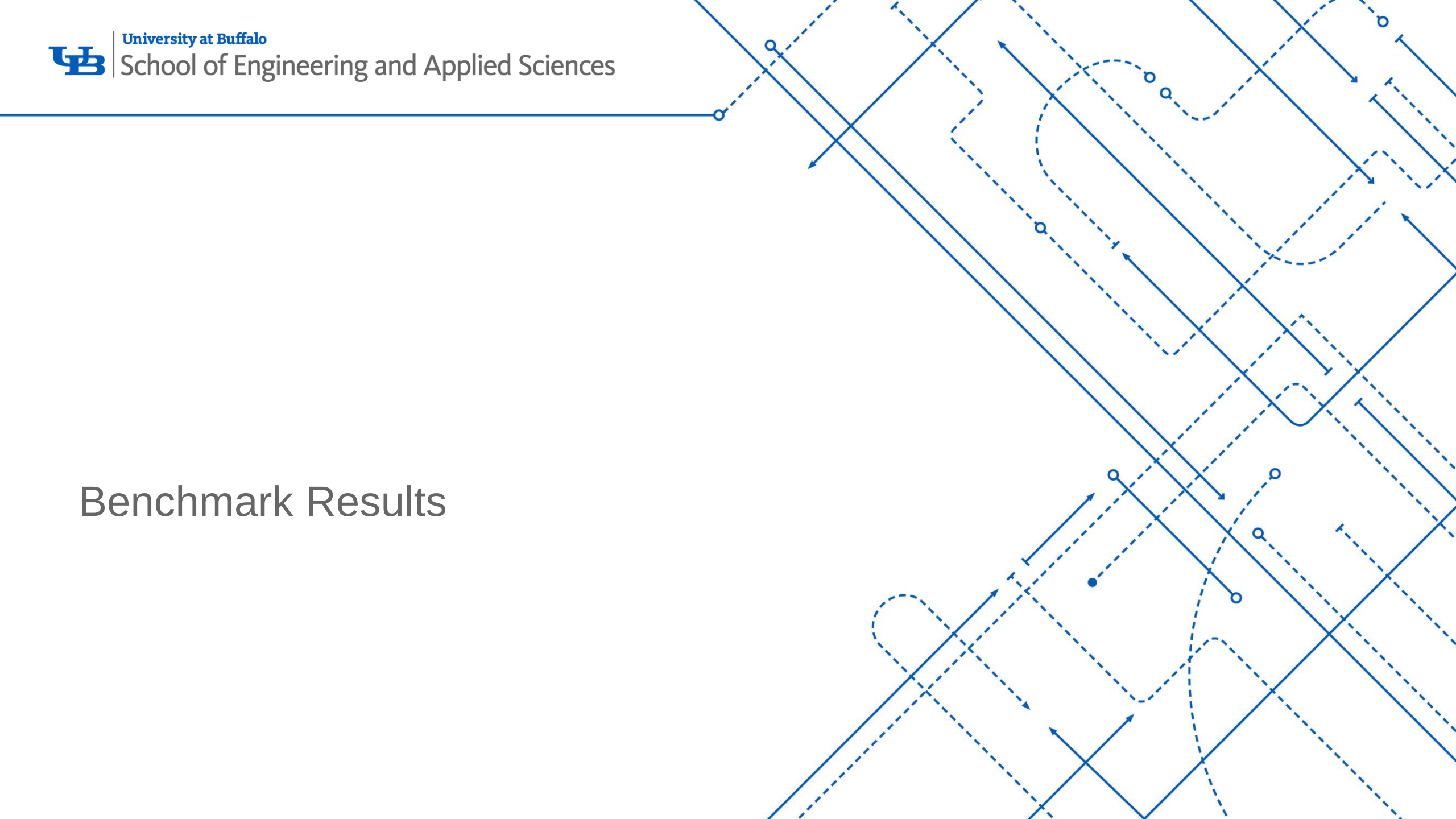
Synthetic Benchmarks

- Tested on UB CCR systems with Intel Xeon E5645 processors
- Tested using 1, 2, 4, 8, and 11 workers plus one master
- 15 binary datasets with 2^2 through 2^{16} observations
- Full datasets have each observation as its own class in order to force a perfect binary tree (worst-case scenario)
- Fractional datasets have the value of one feature determine the class (if 1st feature is 0, then the class is 0, otherwise each value has a unique class)
- Times used exclude I/O and the initial time to transfer data to all processors, sequential overhead and all other MPI operations included



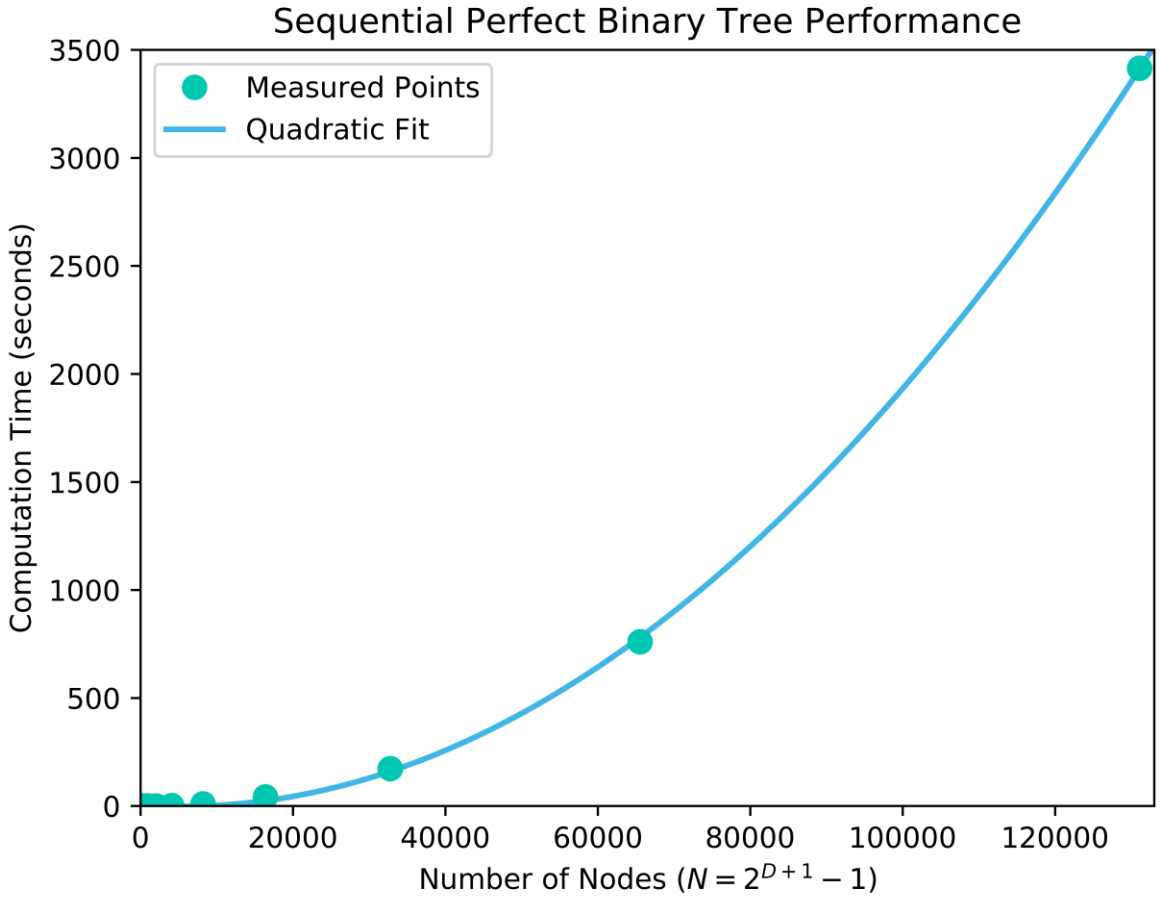
University at Buffalo
School of Engineering and Applied Sciences

Benchmark Results



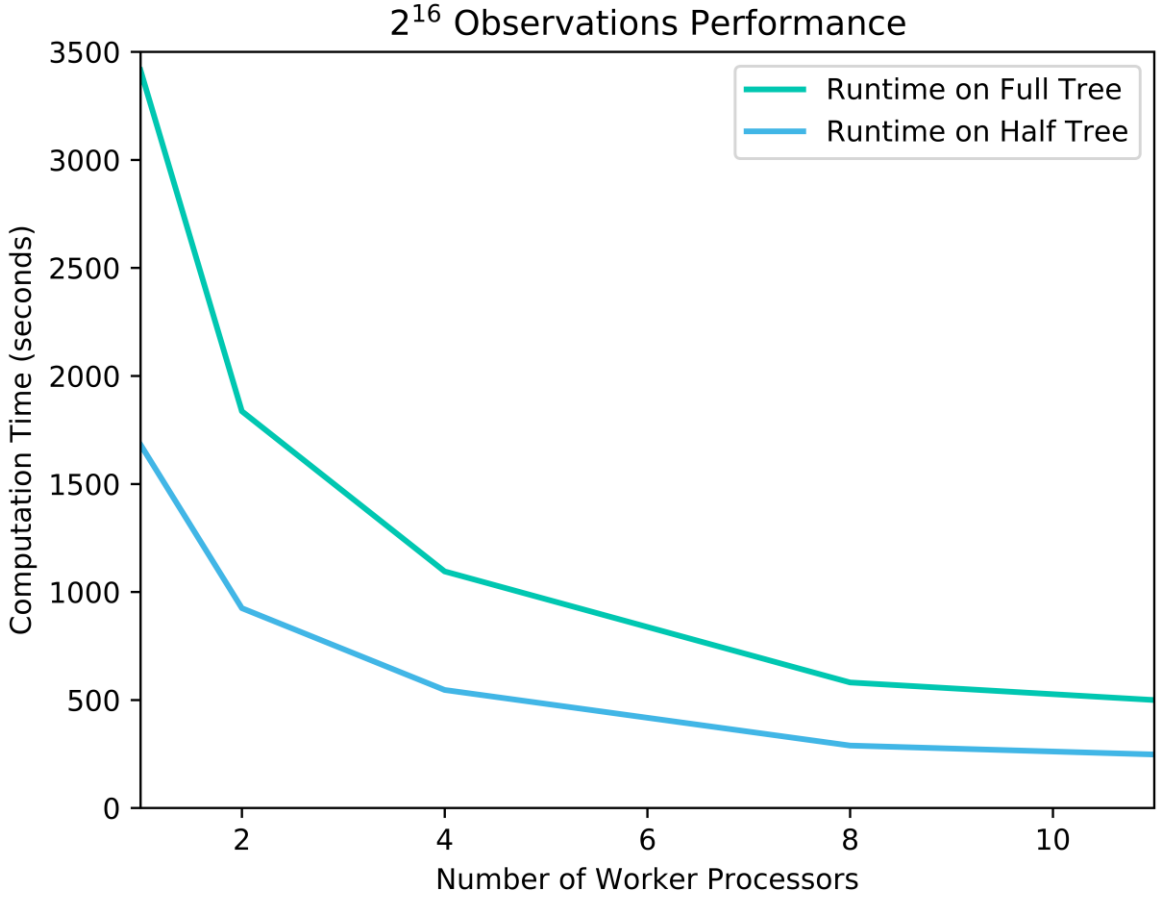
Sequential Performance

- Predicted performance of $O(N^2)$
- Performance measured using two processors (one master and one worker)
- Fit curve using scipy.optimize.curve_fit



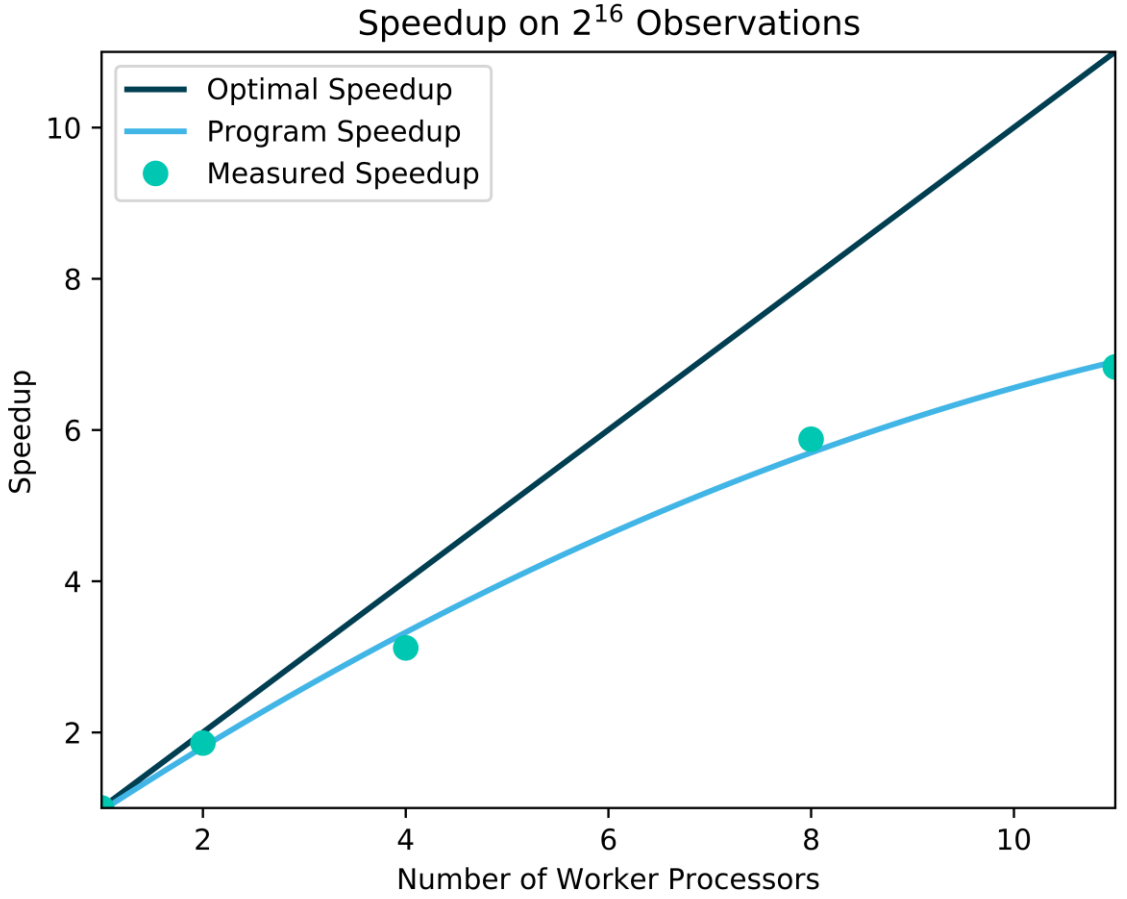
Parallel Performance On Different Tree Constructions

- Master-Worker architecture should have runtime proportional to number of nodes
- Performance curves show that at all measured times, runtime on the full-tree is twice that as on the half-tree
- On smaller test sets, effect is less pronounced due to the larger impact of sequential initialization



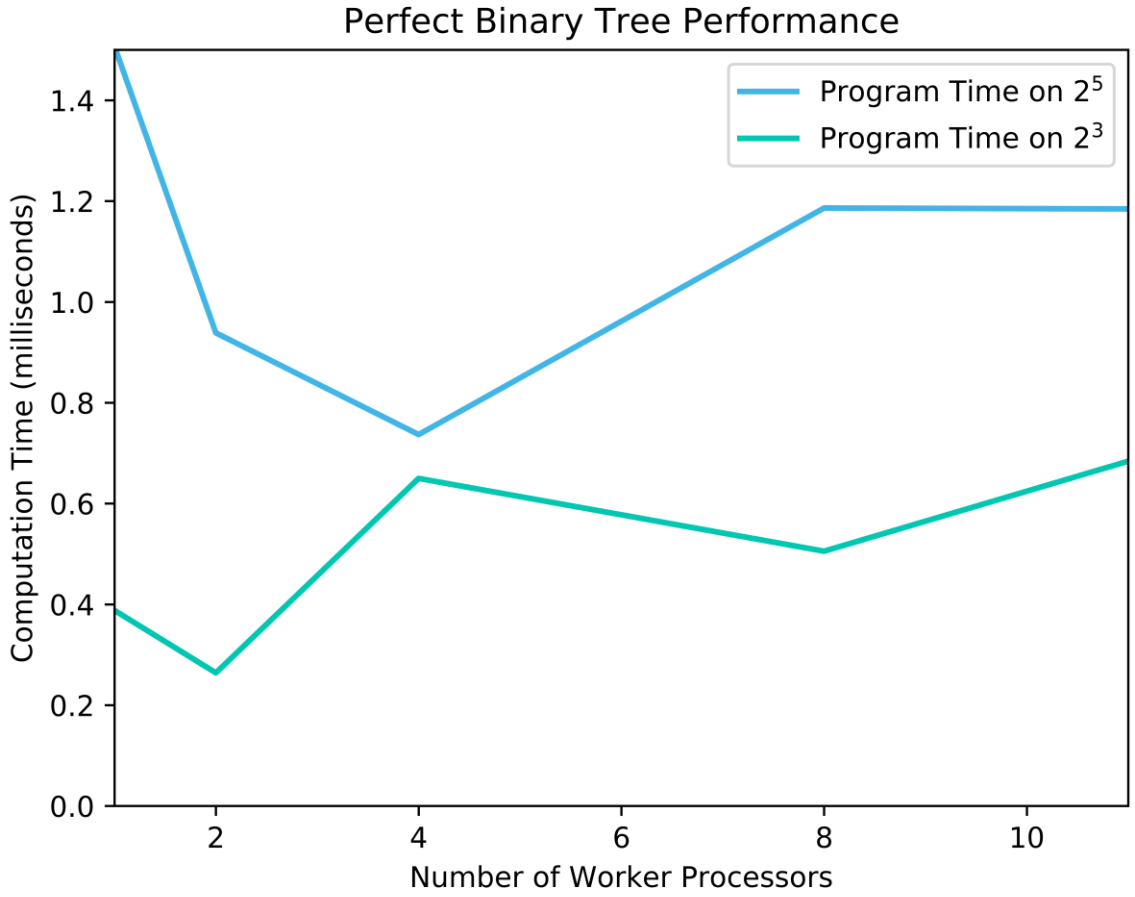
Parallel Speedup

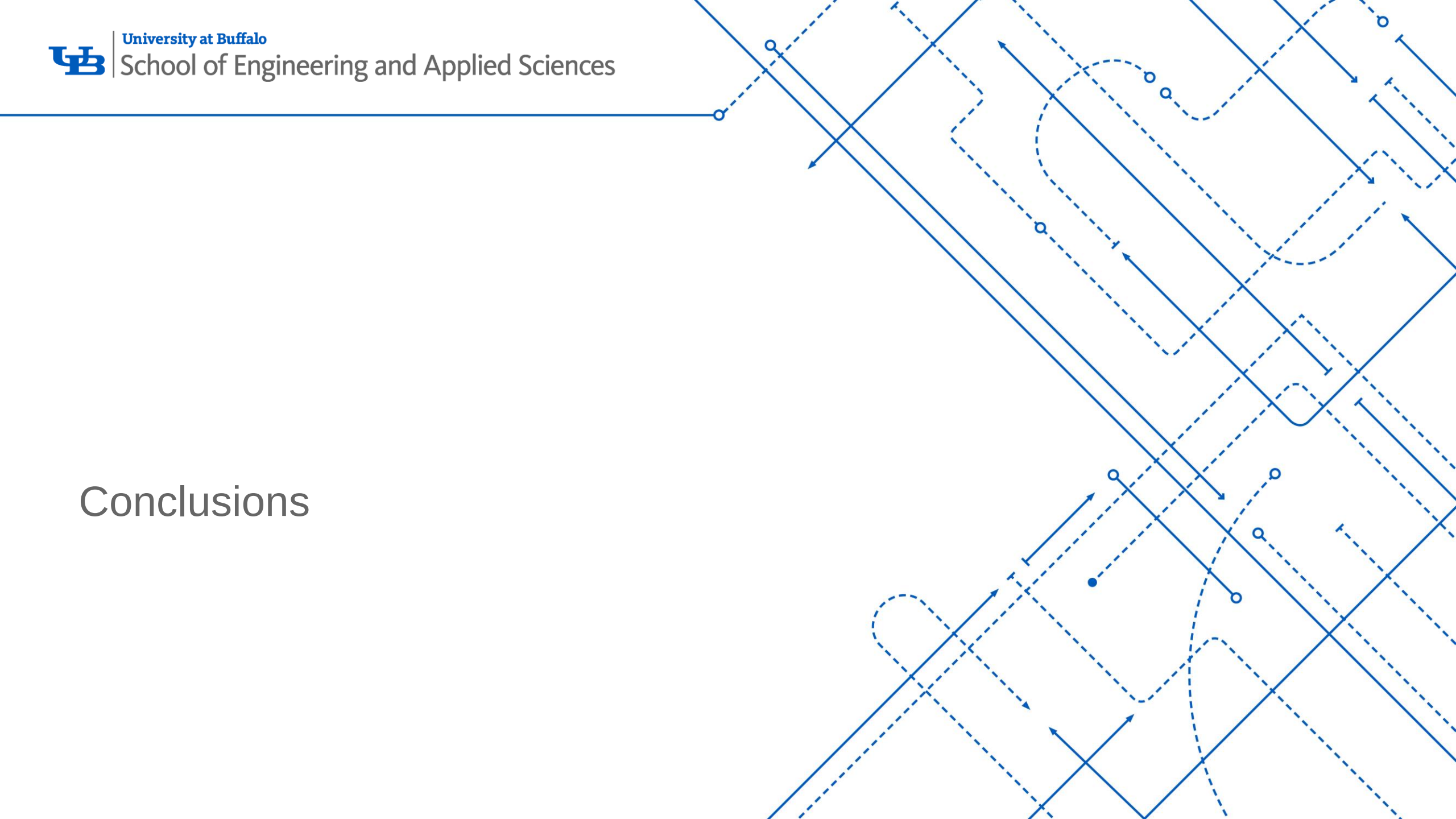
- Optimal speedup would be equal to the number of worker processors
- $O(1)$ overhead when sending a task to and $O\left(log_k^2 N\right)$ when receiving a result
- Inevitable queueing delays when multiple processors try to return simultaneously



Overhead Overtaking

- Even in cases where there are more tasks than processes, overhead can mean that fewer processors yield higher performance





Conclusions

Expectations Met and Unmet

- Sequential performance was as predicted
- Parallel speedup takes a large hit due to overhead, in part from having a single coordinator and frequent communication needs
- Improvement over simple divide and conquer in unbalanced datasets is linear to the reduction in nodes as expected

Future Improvements

- Reduce the $O(log_k^2 N)$ receiving overhead if at all possible
- Cluster individual processor tasks into multiple nodes instead of single nodes to reduce communication overhead, and reduce duplicate processing
- Dynamically create tree topology to reduce $O(Nlog_k N)$ initialization overhead in master processor