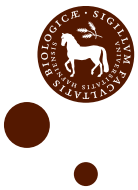
Making Woody Parallel

Applying Parallel Computing to decision trees

Hugh McGrade wbr412@alumni.ku.dk
Mads Obitsø scr411@alumni.ku.dk
Titus Robroek robroek@di.ku.dk
-
Department of Computer Science DIKU

November 8, 2018



Introduction

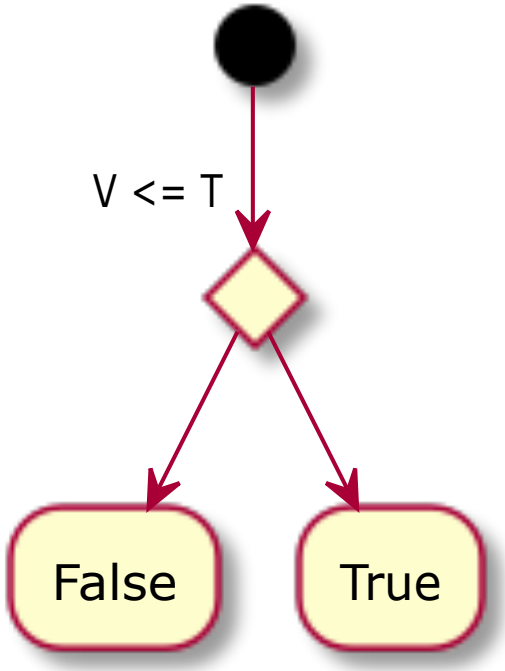
- What is Woody?
- Decision trees and Machine Learning

Working with Woody

Decision tree evaluation

A decision tree consists of nodes. Non-leaf nodes consist of a conditional check, where:

- V: Feature
  value checked for this node
- T: Threshold value of the
  node that is checked against



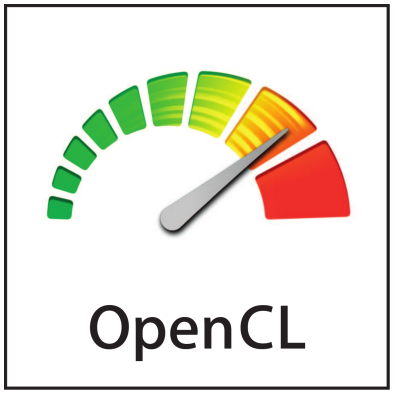
Working with Woody

Extracting from Woody

1. Python library run against a dataset
2. Split data into training and test set
3. Fit on training set to create a forest of trees
4. Run predictions with test set on this forest

Working with Woody

Interoperation of Woody and Futhark



Run as a library or run stand-alone?

Working with Woody

Tree and test data encoding for Futhark

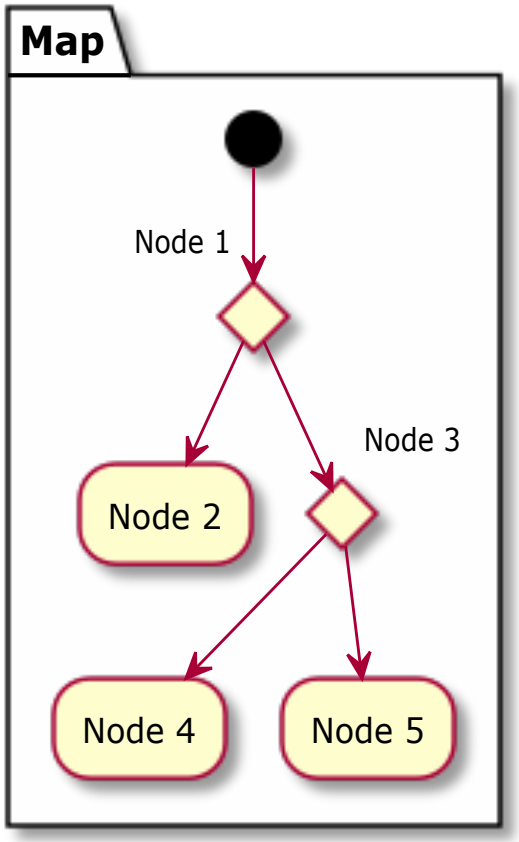
In order to pass the tree and test data from Woody to Futhark, we encoded each as a series of flat arrays.

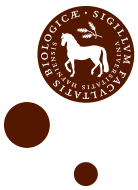
- `treeLeftid`
- `treeRightid`
- `treeFeature`
- `treeThres_or_leaf`

- `Xtest`
- `nXtest`
- `dXtest`

Writing Futhark

Basic Futhark implementation





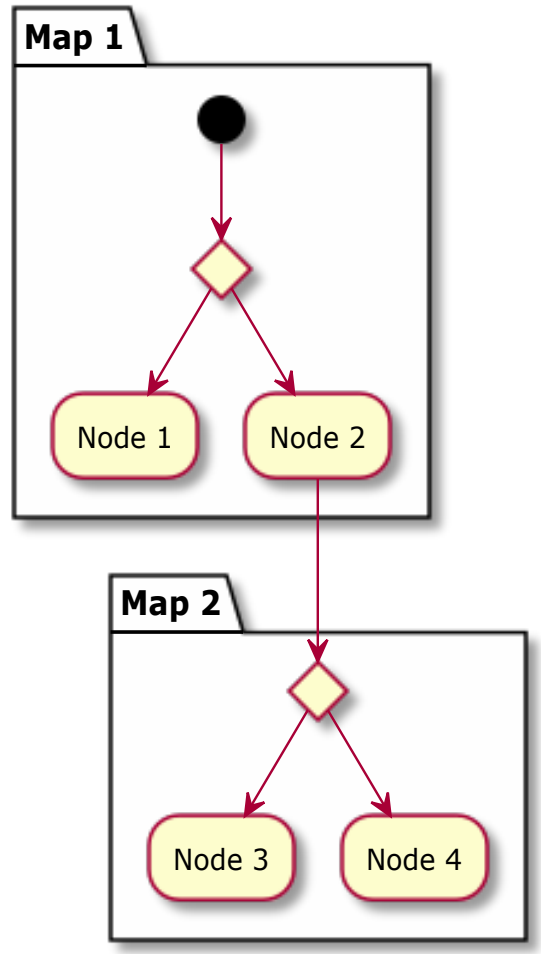
Writing Futhark

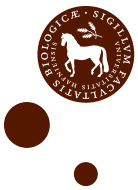
Basic Futhark implementation


```
unsafe map (\ i ->
let idx = if dindices > 0 then indices[i] else i
let row_start = idx * dXtest
in loop node_id = TREE_ROOT_ID
      while treeLeftid[node_id] != TREE_CHILD_ID_NOT_SET do
            if Xtest[row_start + treeFeature[node_id]] <= treeThres_or_leaf[
                  node_id]
              then treeLeftid[node_id]
              else treeRightid[node_id]
           ) (iota n_preds)
```

Writing Futhark

Layered Futhark implementation

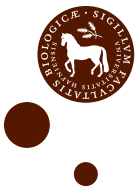




Writing Futhark

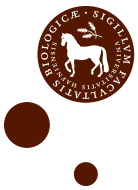
Layered Futhark implementation


```
loop node_array for row in iota(depth) do
    unsafe map (\ (node_id, data_row_start) ->
                  if (treeLeftid[node_id] != 0)
                     then (if Xtest[data_row_start + treeFeature[node_id]] <=
                               treeThres_or_leaf[node_id]
                           then treeLeftid[node_id]
                           else treeRightid[node_id])
                     else node_id)
                     (zip node_array data_row_starts)
```



Writing Futhark
Filtering Futhark implementation

- Layered provides a structured way of iterating over layers
- Many passes end early
- These are still considered by the maps



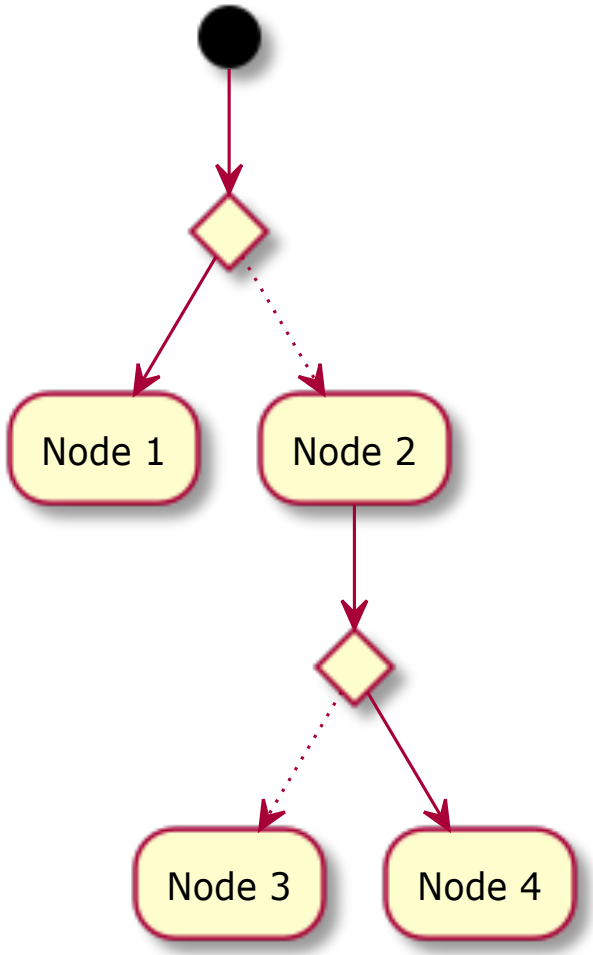
Writing Futhark

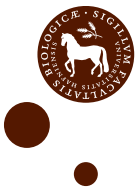
Filtering Futhark implementation


```
let node_array = replicate n_preds 0
let is_not_leaf = (\ (node_id, _) -> treeLeftid[node_id] != 0)
let is_leaf = (\ (node_id, _) -> treeLeftid[node_id] == 0)
let next_node = (\ (node_id, data_row_start) ->
                 ((if (is_not_leaf (node_id, data_row_start)) then (if Xtest[
                       data_row_start + treeFeature[node_id]] <= treeThres_or_leaf[
                       node_id] then treeLeftid[node_id] else treeRightid[node_id])
                       else node_id), data_row_start))
let nodes = zip node_array data_row_starts
let leaves = []
let (_, leaves) = loop (nodes, leaves) for row in iota(depth) do
                  let new_nodes = (unsafe map next_node nodes)
                  in (unsafe filter is_not_leaf new_nodes,
                      leaves ++ (unsafe filter is_leaf new_nodes))
                  let result = map (\ (a, _) -> a) (radix_sort_by_key (\ (_, a) -> a)
                        i32.num_bits i32.get_bit leaves)
in result
```

Writing Futhark

Treesolver Precompute

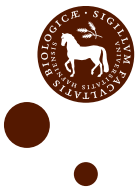




Writing Futhark

Treesolver Precompute


```
let next_node
  (row: []f64)
  ((left, right, feature, thres) : (i32, i32, i32, f64)) : i32 =
  if row[feature] <= thres then left else right

let make_next_tree
   (tree: [](i32, i32, i32, f64))
   (row : []f64) : []i32 =
   map (next_node row) tree

let traverse
   (next_nodes: []i32): i32 =
   let (last, current) = (0, next_nodes[0])
   let (result, _) = loop (last, current) while current != 0 do (current, next_nodes
         [current])
   in result

let nodes = zip4 treeLeftid treeRightid treeFeature treeThres_or_leaf
let rows = unflatten nXtest dXtest Xtest

let next_nodes = unsafe map (make_next_tree nodes) rows

in unsafe map traverse next_nodes
```



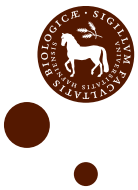
Writing Futhark

Treesolve Matrix


```
let repeated_criteria = flatten (replicate nXtest treeFeature)
let repeated_offsets = flatten (map (\ i -> replicate treelength i) (steps 0 nXtest
    dXtest))
let flcr = map2 (+) repeated_offsets repeated_criteria
let scattered_features = unsafe map (\ i -> Xtest[i]) flcr
let threshold_result = map2 (<=) scattered_features (flatten (replicate nXtest
    treeThres_or_leaf))
let left_or_right = (\ b l r -> if b then l else r)
let repeatedLeft = flatten (replicate nXtest treeLeftid)
let repeatedRight = flatten (replicate nXtest treeRightid)
let directions = map3 left_or_right threshold_result repeatedLeft repeatedRight

in unsafe map traverse (unflatten nXtest treelength directions)
```

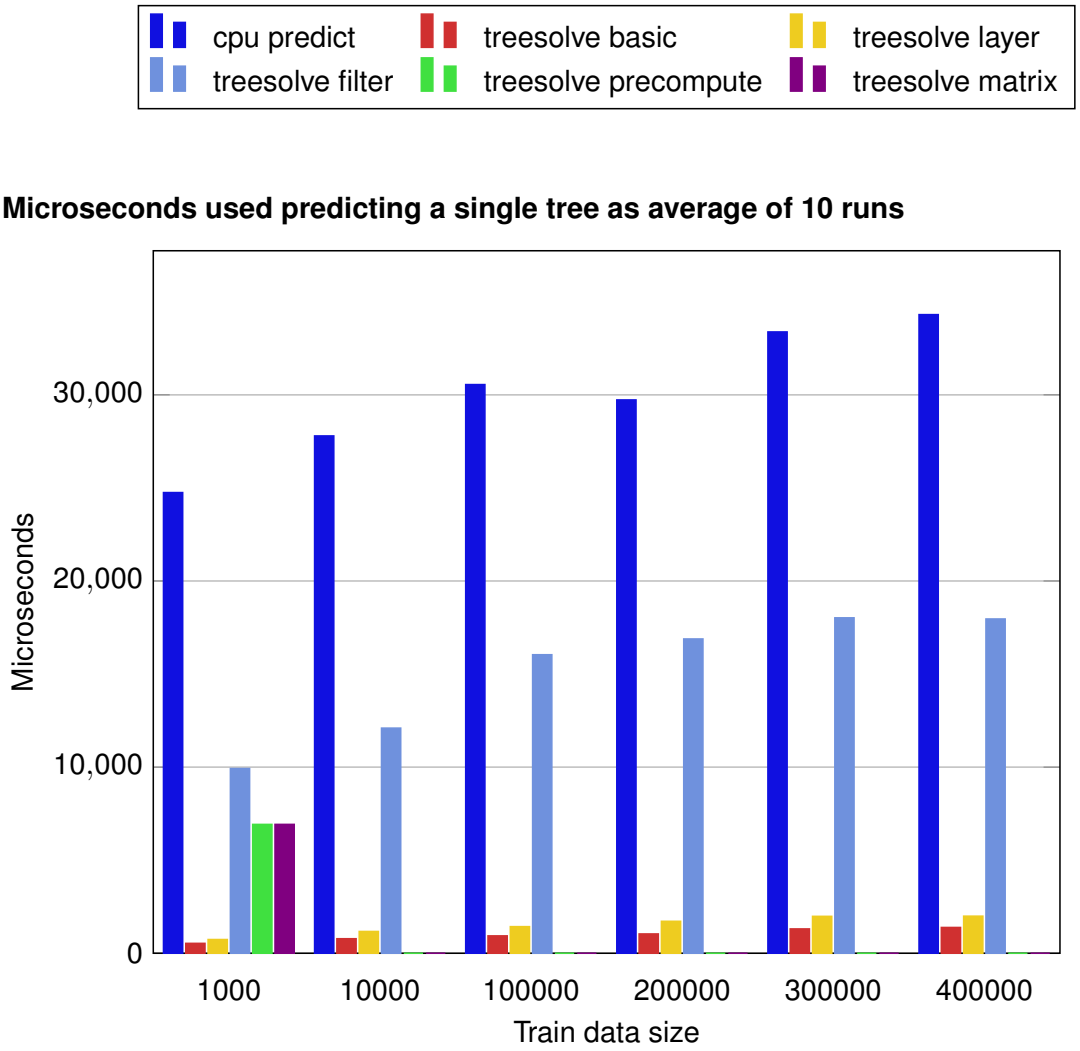
Experiment
Experimental Setup

To evaluate our Futhark implementation, we performed various comparisons with the woody implementation.
–Perhaps add an extra slide with an overview of the framework as an image? Itemize the tests–



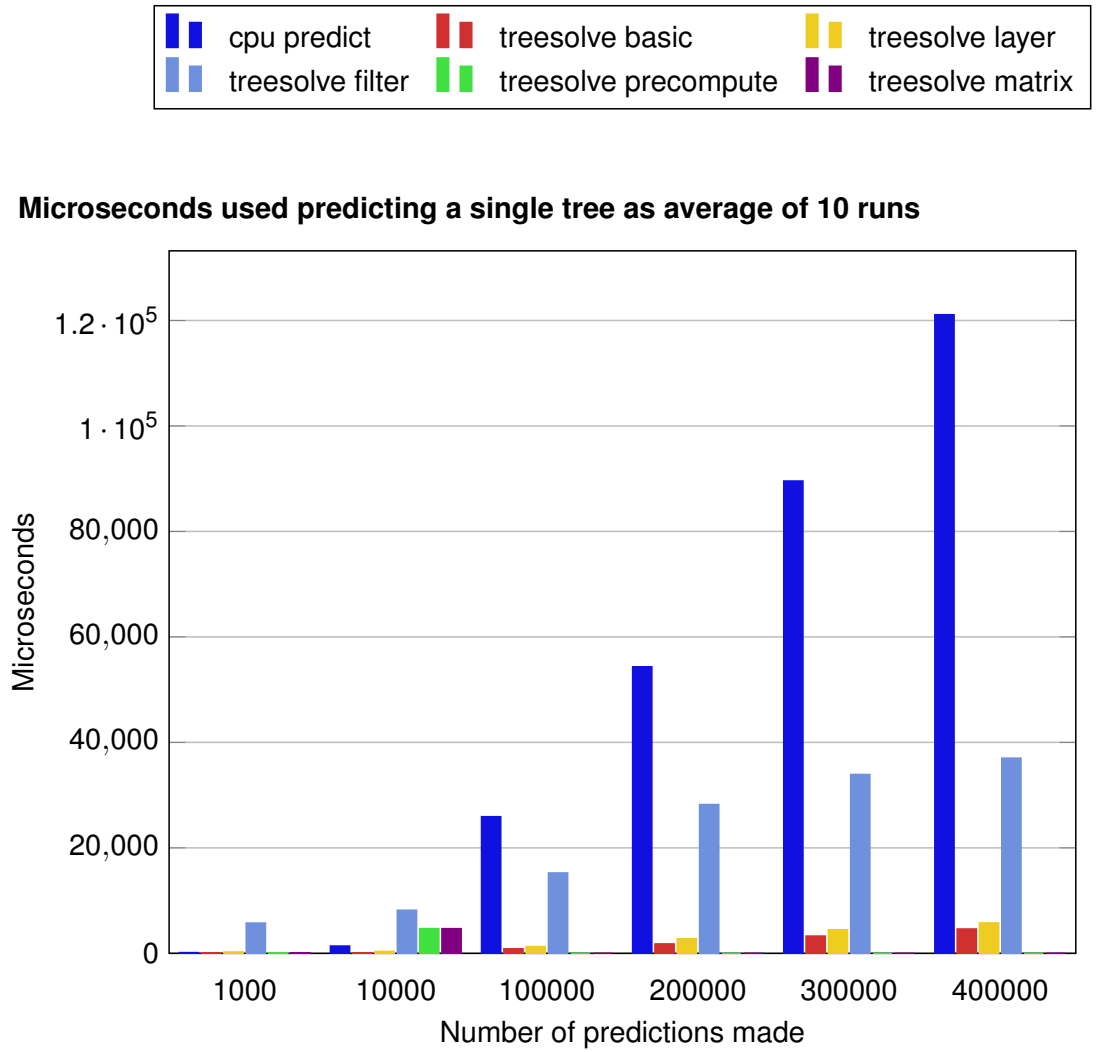
Experiment

Results: Varying train data size



Experiment

Results: Varying number of predictions

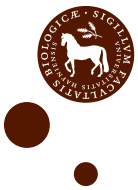


Experiment
Evaluation

- Basic and layer are fast.
- GPU code scales better with bigger tests

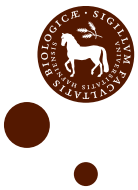
Experiment
Future Research

- Matrix and Precompute might be promising
- GPU code scales better with bigger tests



Conclusion

We have proposed a number of approaches to parallelising the evaluation of decision trees using Futhark. Our findings show that as a whole this parallelisation is promising for the performance of decision tree evaluation on large datasets.



Belgiu, Mariana, and Lucian Drăguţ. ”Random forest in remote sensing: A review of applications and future directions.” ISPRS Journal of Photogrammetry and Remote Sensing 114 (2016): 24-31.

Gieseke, Fabian, and Christian Igel. ”Training Big Random Forests with Little Resources.” arXiv preprint arXiv:1802.06394 (2018).