Random Forest Impelementation in CUDA

Ankur Yadav (CS18M063)
Shabhaz Husain (CS18M049)
Rahul Vashisht (CS18D006)
Mentored by Abinash Patra

IIT Madras

May 1, 2019

Overview

Introduction

- Random Forest is a supervised classification algorithm.
- It creates multiple number of decision trees and uses majority votes for classification.
- To model the decision tree for random forests a sample of data is chosen with repetition.
- This reduces the variance of overall algorithm (in comparison to one decision tree).

Existing Implementation

- We are using Iterative Dichotomizer3 (ID3) algorithm to create the decision trees.
- There is no serial code available for random forest using ID3 algorithm in C language. We have Implemented serial version of random forest ID3.
- There is no parallel code available for random forest using ID3 algorithm in cudaC language. We have Implemented parallel version of random forest ID3.

Results Overview

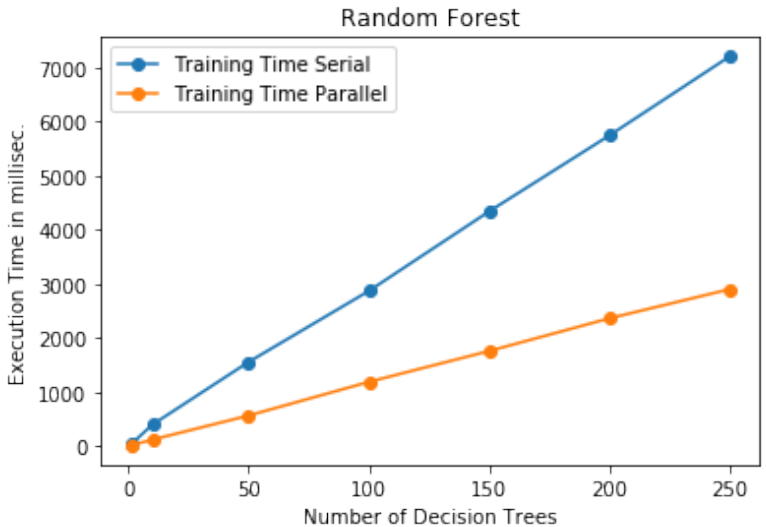
- We have used shared memory , atomic operation,parallel reduce operation (Max,Sum) etc.
- We have used the task level parallelism for cuda code where number of blocks is equal to number of attributes and number of threads in a block is equal to number of datapoints.
- We have parallelized the following task using cuda kernels such as getCardinallity, getInfoGain of an atribute and data.
- We observed the best speedup for number of decision trees is equal to 10 is 3.3.
- Our main observation is that use of task level parallelism for the small task is not good.

Profiling

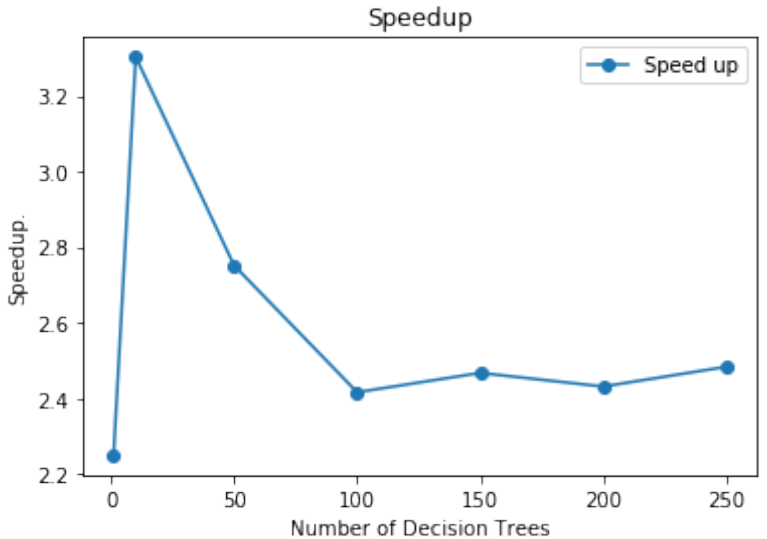
- Serial code in c is not required memorycopy during function call we can simlpy pass as an arguement, so for small task it's faster than gpu.
- The comparison of Serial code with parallel code is shown in the figure (a).

```
cs18m063@gpumaster-machine:~$ gstat -o
hello world
==16== NVPROF is profiling process 16, command: ./pi
==16== Profiling application: ./pi
==16== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   59.79%  91.648ms      4834  18.959us  1.5360us  130.98us  getInfoGains(int*, int*, int, float*, int*, int*)
                   19.02%  29.147ms      4834  6.0290us  5.0560us  41.856us  getInfoGainOfData(int*, int, int*, int*)
                   12.71%  19.478ms      9668  2.0140us  1.9520us  2.8800us  [CUDA memcpy DtoH]
                    8.43%  12.924ms     14512     890ns     832ns  9.2480us  [CUDA memcpy HtoD]
                    0.04%  55.200us        10  5.5200us  5.3120us  6.4000us  getCardinality(int*, int*)
                    0.01%  20.192us        10  2.0190us  1.9840us  2.0800us  [CUDA memset]
      API calls:   23.10%  332.76ms         8  41.595ms     786ns  332.75ms  cudaEventCreate
                   17.51%  252.22ms     24180  10.430us  1.9830us  4.1039ms  cudaEventSynchronize
                   16.17%  232.96ms     19346  12.041us  6.3810us  678.20us  cudaMemcpy
                    9.27%  133.52ms     48360  2.7610us  1.7310us  491.11us  cudaEventRecord
                    7.94%  114.31ms      9678  11.811us  8.2430us  664.84us  cudaLaunch
                    7.63%  109.83ms     14502  7.5730us  4.2800us  662.34us  cudaFree
                    7.01%  100.90ms     14504  6.9560us  3.8760us  713.00us  cudaMalloc
                    6.42%  92.417ms      4834  19.118us  14.340us  669.62us  cudaMemcpyFromSymbol
                    3.86%  55.583ms     24180  2.2980us  1.6900us  682.67us  cudaEventElapsedTime
                    0.77%  11.108ms     48360     229ns     110ns  660.74us  cudaSetupArgument
                    0.21%  3.0690ms      9678     317ns     178ns  7.2880us  cudaConfigureCall
                    0.07%  1.0580ms        94  11.255us     287ns  414.59us  cuDeviceGetAttribute
                    0.03%  386.01us         1  386.01us  386.01us  386.01us  cuDeviceTotalMem
                    0.01%  102.51us         1  102.51us  102.51us  102.51us  cuDeviceGetName
                    0.01%  97.406us        10  9.7400us  5.6960us  17.350us  cudaMemset
                    0.00%  3.1440us         2  1.5720us     525ns  2.6190us  cuDeviceGetCount
                    0.00%     925ns         2     462ns     358ns     567ns  cuDeviceGet
cs18m063@gpumaster-machine:~$
```

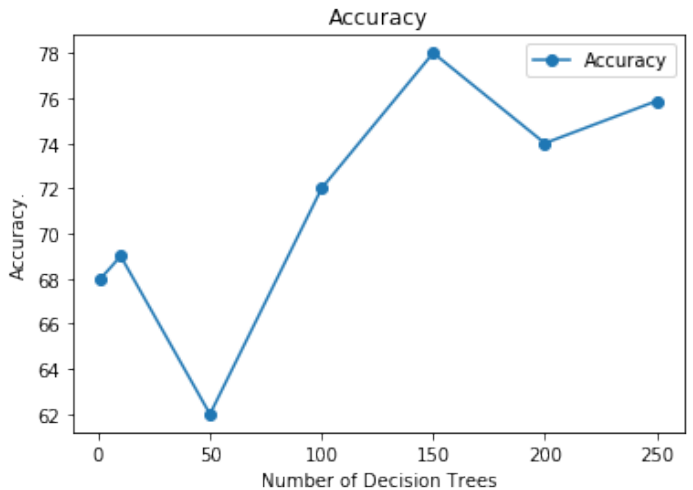
Observations and Results



(a) Training time : Serial vs Parallel



(b) Speed up



(c) Test Accuracy

Continue...

- As the number of decision tree the time taken increases linearly.
- We have achieved speed up of 3.30.
- The test data accuracy increases as number of decision tree increases.
- Cudamemcpy and some API calls(cudaEventCreate, cudaEventSynchronize, cudaFree) taking more time than kernel execution.

Team Contribution

Ankur Yadav	Serial and optimized Parallel Code
Shahbaz Husain	Serial,Naive Parallel Code with parallel reduce operations
Rahul Vashisht	Serial and Optimized Parallel Code

Learnings and Future Works

- We have considered pure decision trees for classification, the classification algorithm can be further be improved using the pruning of decision trees.
- We can achieve further speed up by using the hybrid method which combines task level parallelism with data level parallelism.
- The tradeoff between task level parallelism and data level paralelism in a hybrid technique can also be studied.

Thank You