Implement Parallelization on Decision Tree and Random Forest

PP-f18 Final Report

黃則睿
0516230
資工系網多組
raymond860909@gmail.com

顏劭庭
0516301
資工系資工組 A
samuelyen.cs05@g2.nctu.edu.tw

陳筱蕙
0516202
資工系網多組
crystal.cs05@nctu.edu.

Abstract

We choose the topic because of the fact that machine learning conducted for general purpose has become an inevitable trend. Random Forest Classifier is one of the most popular model among these machine models However, constructing a Random Forest Classifier model is a time-consuming process. As a result, we want to apply several parallelization methods to the construction of Random Forest Classifier. In this report, we'll explain what is Random Forest Classifier and how do we parallelize it by different model such as Pthread, MPI and OpenMP.

Introduction of Basic Conception

Machine learning has become one of the popular techniques in computer science. While computing power is keeping increasing in nearly exponential rate, the requiring resources of those techniques still grow much faster than hardware does. In this case, how optimize the program to fully utilize the hardware become an important question, especially for ML, which requires large amount of computing resources to make it works.

Before explaining how to parallelize the whole program, I would like to talk a little bit about what is Decision Tree Classifier and what is Random Forest Classifier.

Decision Tree Classifier is a tool that helps you predict which class the data belongs to. In usual, given a data set, we'll divide them into two parts, that is, training data and testing data. We use training data to construct the Decision Tree Classifier and use testing data to test the validity, for example, the precision of our model.

Let's take iris data set for example, each entry of dataset consists of four floating point attribute and one categorical class. The following is an entry of iris dataset. (4.4,2.9,1.4,0.2,Iris-setosa) Assume that there are total k entries in a dataset. Basically we'll take 0.5*k entries as training data and 0.5*k entries as testing data.

The procedure of training the model is described under. Frist, sort the data by each feature and compute the entropy and information gain of that attribute. Second, choose the feature with highest information gain and compute a proper threshold for that feature. Third, divide the data into two parts, that is, check if a specified feature of a n entry satisify the threshold or not. Fourth, pass the entries that satisify the condition to the left child of current node and pass the entries that do not satisfy the condition to the right child of current node. Fifth, recursively go to first step until all of the data has the same class.

I would like to explain a little bit about entropy which is mentioned in the procedure of constructing Decision Tree. The following formula stands for entropy with dataset S.

p(x) means the portion of number of elements with that belongs to class x. The whole formula requires tons of calculations especially in continuous data. While dealing with continuous data, we have to use every number exist in dataset as the threshold

of p and calculate and store the corresponding

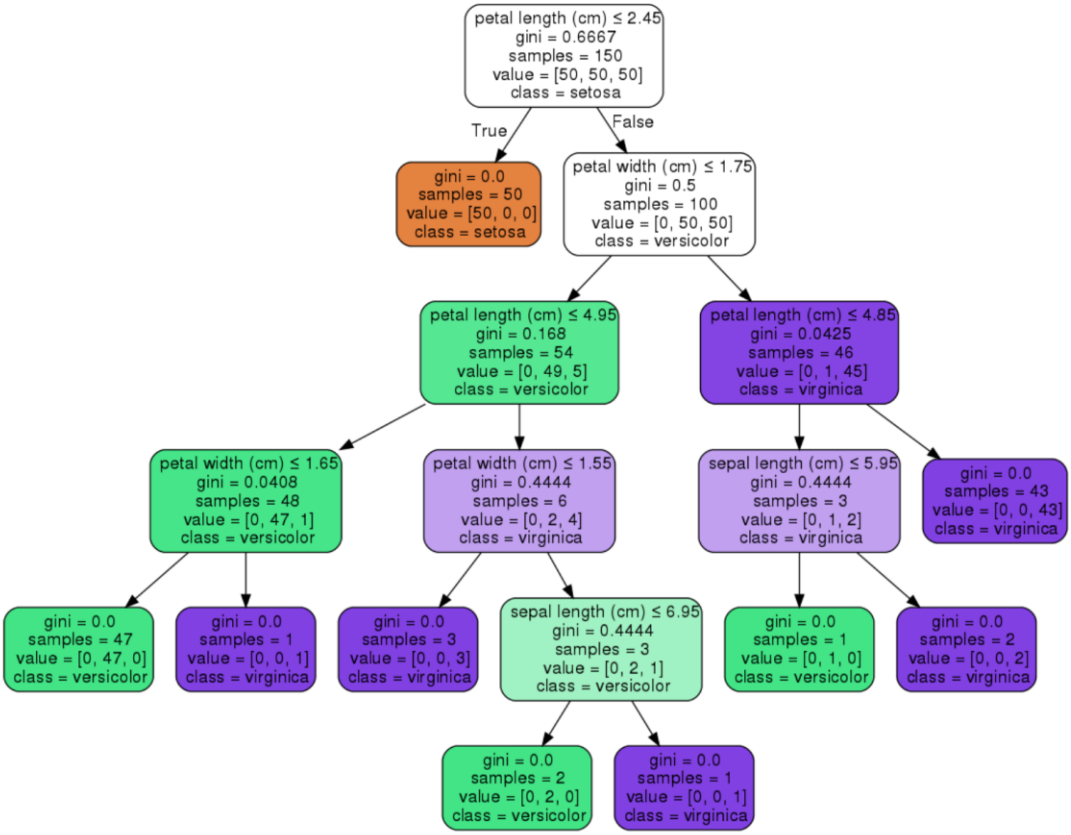
$$\mathrm{H}(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

entropy.

$$IG(S, A) = \mathrm{H}(S) - \sum_{t \in T} p(t)\mathrm{H}(t) = \mathrm{H}(S) - \mathrm{H}(S|A).$$

The formula above represents information gain. We'll use the entropy calculated before to evaluate the information gain of current dataset. In ID3 algorithm, we'll choose the feature and threshold with maximum information gain and minimum entropy and use that feature with corresponding threshold to generate the node of Decision Tree Classifier.

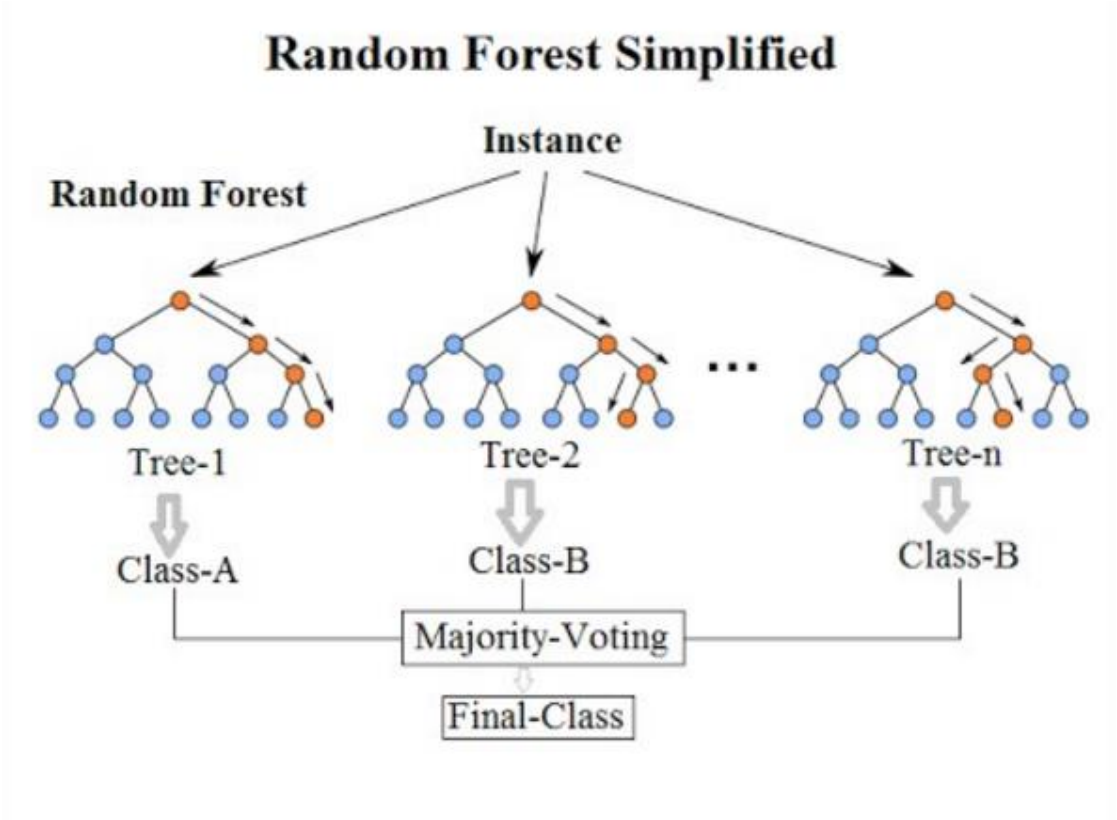
The time complexity of Decision Tree Classifier is roughly O(log(n) * n^3).

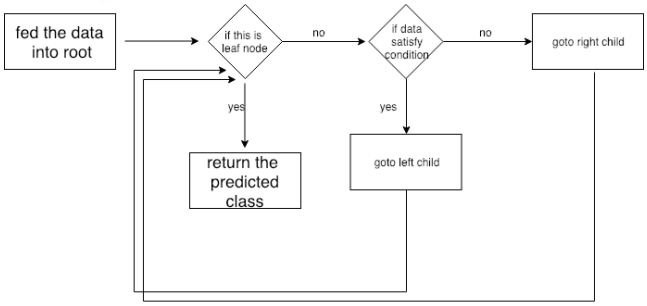


The following picture is a Decision Tree Classifier with the whole Iris dataset as the training data.

At the time you want to validate your model or you want to classify the unknown data, the procedure is described as follow.

First, The testing data is fed into the root of Decision Tree Classifier. Second, if it is currently in the non-leaf node, we would check if that data satisify the condition stored in the node.If the data satisifies the condition, then go to the left child of current node and go back to second step. If the data does not satisify the condition, then go to the right child of current node and go back to second step. However, if it's currently in a leaf node, then it would return the predicted class held in the leaf node.





The flowchart above shows the whole procedure.

Random Forest Classifier is an extension of Decision Tree Classifier. We would like to apply random sampling on our dataset feature selection in order to construct our Random Forest Classifier. After doing some research on Random Forest Classifier. We were acknowledged that proper number of Decision Trees is usually placed around 50 to 100. As a result, we decided to construct 64 Decision Tree Classifiers for it is the multiple of 2, 4, 8 and16.

At the time you want to test something on your Random Forest model, you have to feed the testing data to all of the Decision Trees in the Random Forest Classifier, and poll the classes returned by those Decision Tree Classifiers.

Random Forest Classifier provides us with a better way of prediction the class of a data. For it may eliminates the useless attributes or noisy data, making the model has better accuracy.

In our project, we are going to implement a simple universal program with ID3 algorithm and Random Forest algorithm for most kinds of datasets in UCI Machine Learning Repository. Next, we will try to implement different parallelization methods including Pthread, OpenMP, MPI and other parallelization techniques that will be introduced in this course. We estimate that several part of algorithms can be parallelized, such as feature selecting in trees, testifying, and constructing trees with different features enable at the same time. The final results

are expected to show significant difference between serial programs and parallel programs, and We will further compare different techniques of parallelization to make an ideal universal classifier in final.

Experimental methodlogy :

We use the PPlinux server provided by TAs to test and evaluate the performance of our project. We use two datasets to evaluate the performance of our models with different ways of parallelization. The two datasets are abalone dataset and nursery dataset. The abalone dataset consists of 4177 instances with 8 floating-point attributes, and the nursery dataset consists of 12960 instances with 8 categorical attributes. We run the program with a specific parallelization model for three times on each dataset and compute the average execution time.

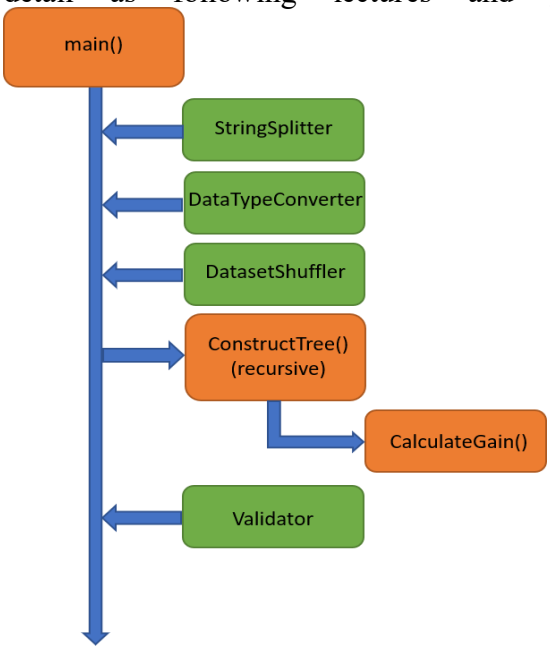
The PPlinux server has an Intel Xeon Gold 6126 CPU with 12 cores 24 threads and has 16GB ram. However, with our commands are actually being executed on a virtual machine, we only have four cores for each server to use.

The output format of our program is shown as follow. We do print out the execution time, progress and accuracy to make it easier for debugging.

```
[yenst@pplinux1 UniversalClassifier]$ ./build.sh
[yenst@pplinux1 UniversalClassifier]$ ./main.out abalone.data
################################################################################
#                                                                              #
#   U   U NN  N IIIII V   V EEEEE RRRR   SSSS  AAA  L                        #
#   U   U N N N   I   V   V E____ R   R S___  A   A L                        #
#   U   U N  NN   I    V V  E     RRRR      S AAAAA L                        #
#    UUU  N   N IIIII   V   EEEEE R   R SSSS  A   A LLLLL                    #
#                                                                              #
#                 CCCC L      AAA   SSSS  SSSS IIIII FFFFF IIIII EEEEE RRRR   #
#                C     L     A   A S___  S___    I   F       I   E____ R   R  #
#                C     L     AAAAA     S     S   I   FFFF    I   E    RRRR   #
#                 CCCC LLLLL A   A SSSS  SSSS  IIIII F     IIIII EEEEE R   R  #
#                                                                              #
################################################################################
#     Input File: abalone.data
#     Splitting input.
#        > Dataset contains 4177 data and 8 features each
#        > Execution time: 0.01972 sec
#     Convert all features into double.
#        > Execution time: 0.025056 sec
#     Shuffling dataset.
#     Constructing random forest with 64 trees and randomly select sqrt(#) features each.
#------------------------------------------------------------------
#        > Execution time: 64.9773 sec
#     Validating.
#        > Execution time: 0.039503 sec
#     Prediction result:
#     Classfication +-0 (for classfication): 24.8923 %
#     Classfication +-1 (for regression-like dataset): 63.1403 %
#     Classfication +-2 (for regression-like dataset): 78.9852 %
#     Total execution time: 65.0622 sec
```

Proposed Solution

First, we are going to construct a program with original concept of decision tree. It consists of several parts to implement different function of decision tree constructing and correctness analyzing. We are going to introduce them in detail as following lectures and graphs.

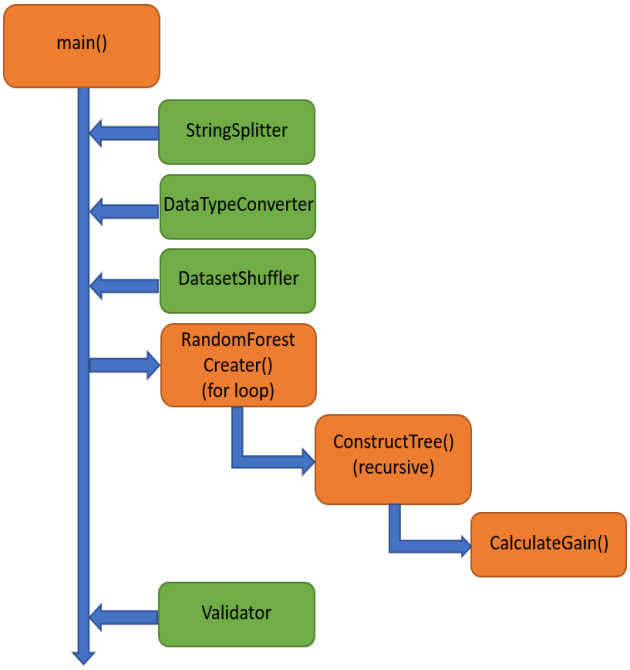


As the graph showing above, there are three parts of program conducting data preprocessing. We construct those parts each as objects to implement these functions. In String splitter, we input the dataset line by line and store into an 2D structured buffer. In Data Type Converter, We Transform all kinds of data into double type. To implement this feature, We use C++ atod function inside try and catch block to handle those who are enable to cast directly to double and those who are not. For those uncastable type of data, we use dictionary to mapping different data into unique float numbers. The reason of doing this procedure is because that the threshold of value should be used in data splitting process and tree node creating process. In Data Shuffler, we use the c++ shuffler with static seed in order to check the correctness of program between serial program and other parallel programs with different methods.

Next part of program below those objects is recursive code. In this part, we calculating the entropy regarding to the whole dataset's classes, the way to calculate entropy has been introduced at introduction part. Then, we assume all the possible way to separate the dataset into two parts with different target features and it's thresholds, and calculate the gain value to evaluate which one has the best performance of guess each dataset's belonging class. As the result, the dataset is separated by the best threshold, and the node of binary tree has been created. All of these processes will be done recursively at both of sub-dataset, until all instances in dataset are same and need not to separate it again, a leaf of the tree is created.

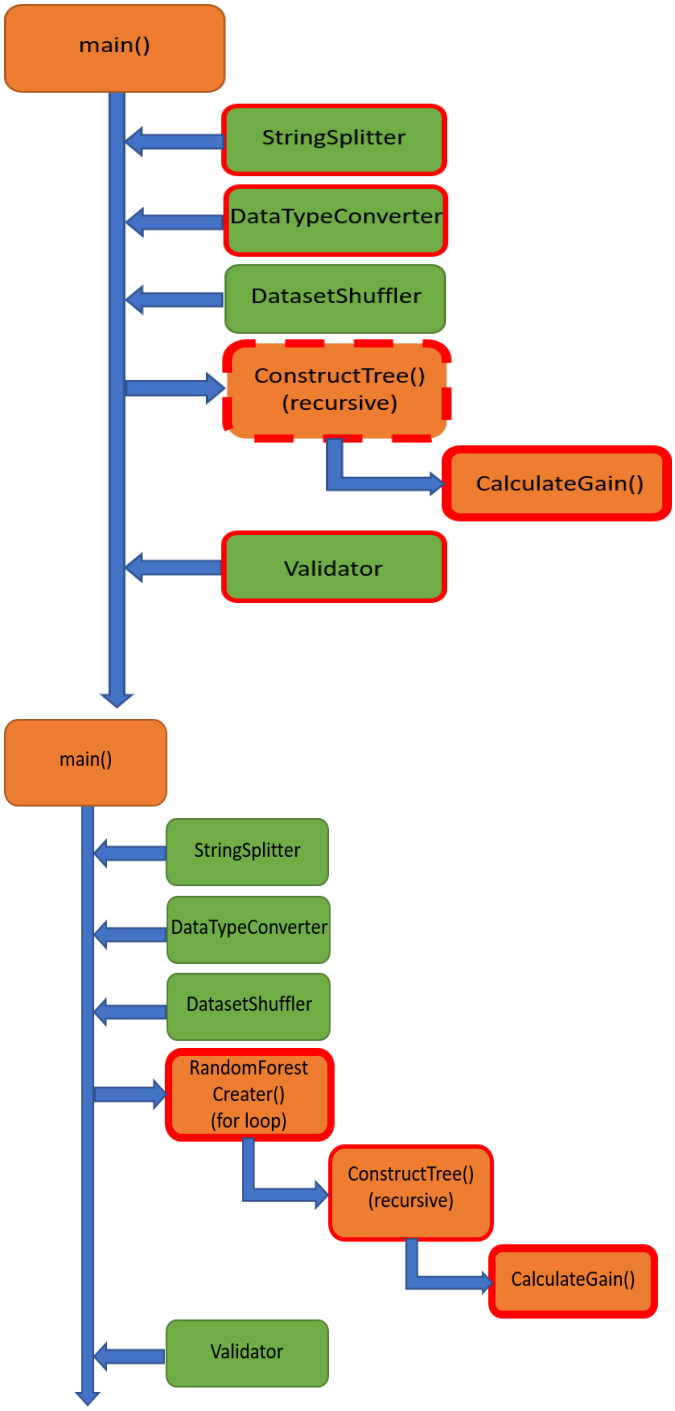
The last Part is Validation. After the tree is created, we input each instance in another dataset, and visit the tree from root to the leaf. With the linked-list data structure as we create, each instance will be testified by the node's threshold, and decide which child node to visit. As the leaf is reached, we compare the predicted result with the instance's actual class to determine the correctness of prediction, and summarize all the result to get the final accuracy of the decision tree.

Second, We are going to modify the original serial code to be random forest validator. To do this, the additional code has been included as follow graph:



We simply add Random Forest Creator function in order to construct multiple trees with loop at the model creating stage, and we slightly modify the validator part and let each instance visit every binary tree respectively, and conduct voting in final.

As these two serial programs are built, we have to figure out which part of the program could be implemented with parallelization. Briefly we can estimate that there are several parts of program can be parallelized in single decision tree program and random forest program:



As we can see in data preprocessing part in both programs, String Splitter and Datatype Converter are parallelizable because of it's simple process and independent characteristic. However, We will not take this part into account due to the fact that it does not have significant difference between serial version and parallel version.

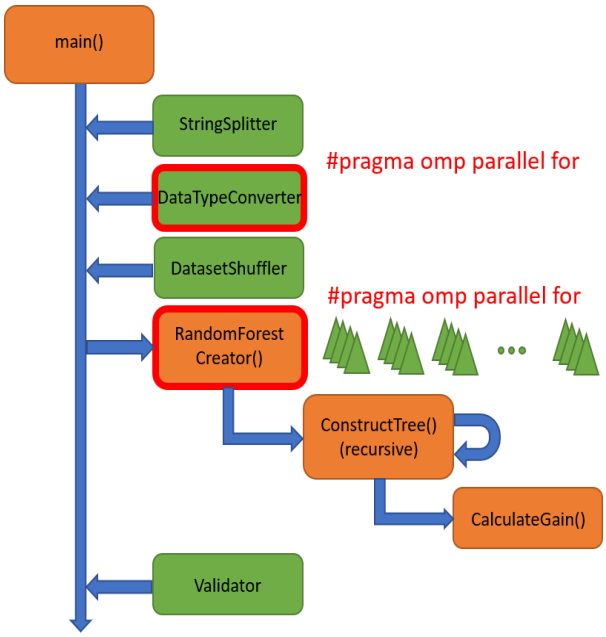
With regard to single tree program, there are two parts of program which have considerable difference when implement parallelization respectively. If we conduct parallelization in Construct Tree Part, the method will be undoubtedly applied on sub-tree creation when a root has been split into two sides, which create another two threads to run the child tree creation process. On the other hand, if we conduct parallelization on Calculate Gain part, the method will be applied at gain calculation on each threshold value. To test their performance, We conducted a simple test that use the same dataset and compare their time consuming. The detail of the test and result will be shown at next section.

To testify the best performance of different parallel ways in random forest classifier, we choose two parts of program to implement parallelization: Random Forest Creator, and Construct Tree, that have been mention in last paragraph. The root parallelization will be applied if we implement parallization on Random Forest Creator because that we can simply unroll the for loop of tree creation and change it into parallelization. Also, we testified their performance in random forest classifier to decide which part of program is suitable to further conduct comparison between the use of different parallelization techniques. Finally, we can conclude that root parallelization may be the best way to implement, an we will show the detailed reason in next section.

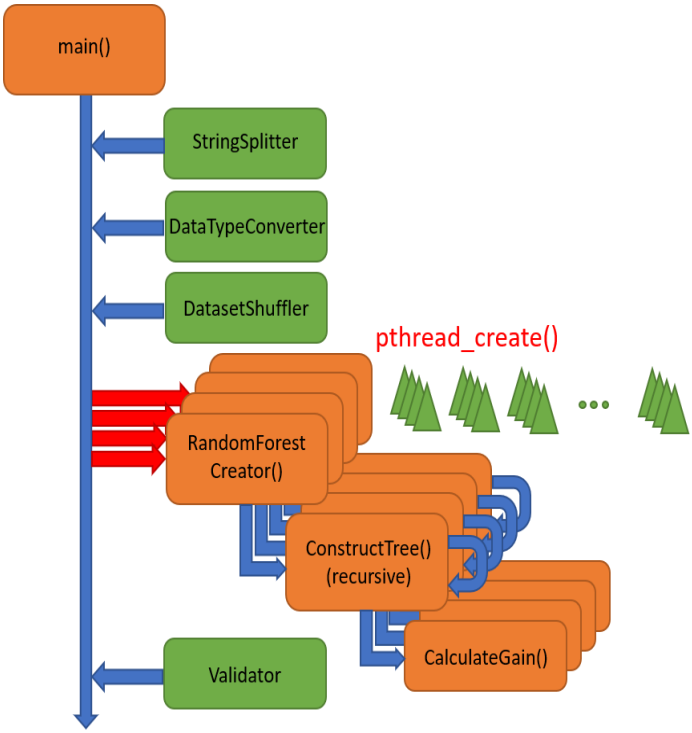
Next, we are going to figure out how to implement different method at the same part of program to evaluate their performance. First, we will discuss about the method that using openMP.

Due to the fact that we can simply add "#pragma omp parallel for" above the for loop statement, and the compiler will automatically create threads to

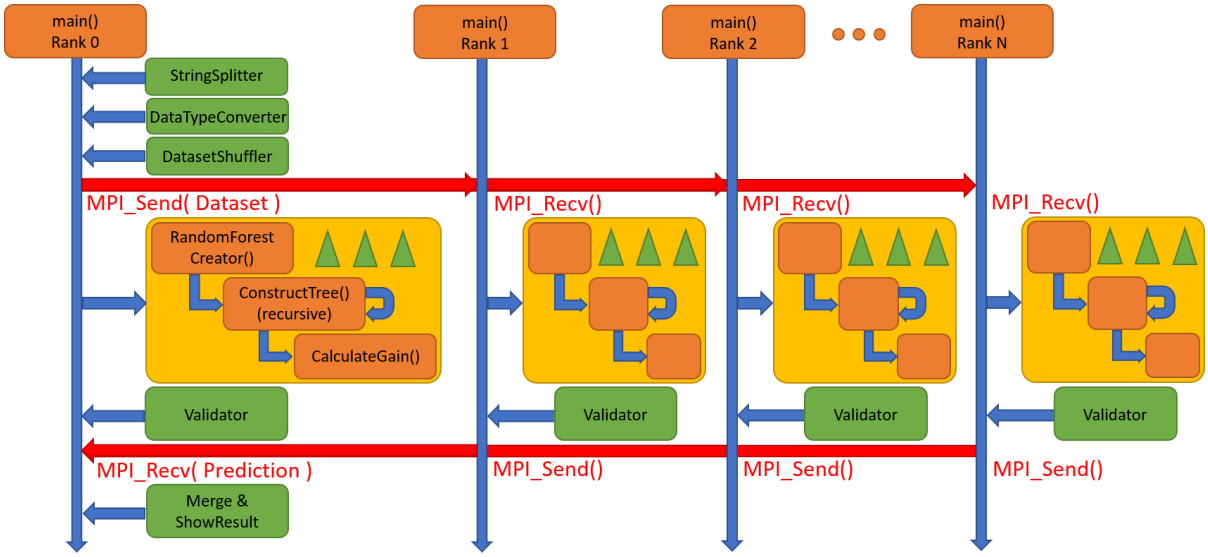
complete the task independently. The other way of parallelization is to add "#pragma omp parallel" with brackets that include the part that we want to parallelize. Thus, we use this technique to implement the program as follow structure:



In addition to OpenMP, we also manually implement Pthread at random forest creator program. Compare to OpenMP, implementing Pthread does not simple as OpenMP, we need to package our function to a function pointer to let the pthread_create() function being called correctly. But in general, the structure of Pthread parallelization code does not have significant different compare to the OpenMP version. The Pthread version of program structure shows as follows:



Lastly, we introduce the third method of parallelization that is quite different with those other techniques: MPI. This tool is developed for the computing units which are not strictly bound in high speed interface. Conversely, each unit was barely connected by internet. By other means, however, MPI provides multi device computing environment to let us utilize our experiment hardware. In order implement MPI program, we must construct a program that can be run by different rank of devices and conduct different process respectively. Besides, the program must include data transmission procedure that can share data from one to another, such as the sharing of dataset and the sharing of vote results. In conclusion, the structure of MPI program is different from other programs. As visualization, the programs is constructed as following structure:



Experimental Results

As we have mention at the previous section, we testify different parts of program to determine which is the best for us to conduct parallelization and further compare each performance of parallelization tool. At single decision part, we using OpenMP and Pthread at to parts of program to estimate its performance, we input 2 datasets with different sizes to test it's time consumption, after 10 times of running and average the result, the result shows at follow table:

OpenMP	nursery	abalone
Serial	30.1646	4.5315
CalculateGain()	10.2204	1.3587
ConstructTree()	30.0050	4.5097
Both	15.5176	3.1925

Pthread	nursery	abalone
Serial	17.2774	3.4555
CalculateGain()	12.2365	11.936
ConstructTree()	17.4890	3.4897

We can summary that Parallelizing Calculate Gain function can have significantly faster result. On the other hand, we also conclude the reason of unable to get significant speed up on Parallelizing Construct Tree function. When a root of tree is being visited, there is no parallelization during the calculation of biggest dataset, which consume the most time in tree construction; When the grand child creation is processed, too many threads is created to serve each node, which cause the system overhead.

Same as previous experiment, we also testify the different parts of random forest program. We compare two programs that parallelize Calculate Gain function at one and Random Forest Creator function at the other. The test method is same as previous work and the result is determined as follow:

OpenMP	nursery	abalone
Serial	17.2774	3.4555
CalculateGain()	12.2365	11.936
RandomForestCreator()	17.4890	3.4897

The result shows that both ways to parallelize the program are faster than serial, but paralleling Random Forest Creator function has slightly better performance. We concluded the reason of the showing fact is that parallelizing Calculate Gain function has less serial part of program, compare to the other one. On the other hand, when applying another solution, each thread creates its own forest,

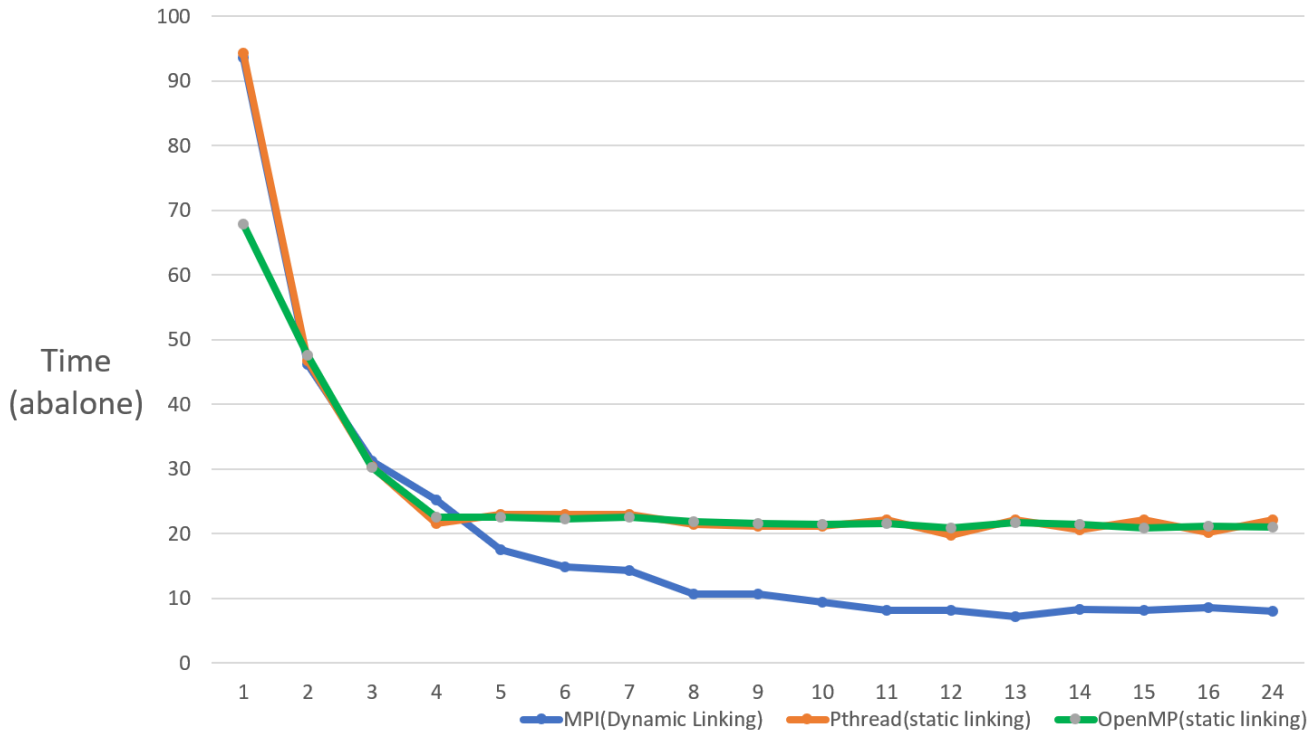
which provide the bigger scale parallelization instead of fragmented parallel program. In conclusion, we are about to compare each parallelization techniques that used in Create Random Forest function.

Lastly, we run each version of programs with manually assigning the number of threads to observe the changes of time consuming. The experiment result shows as following graph. As we can see, the all of them has the significant time reduce when the number of threads is increasing but less or equal to 4 threads. Fewer performance difference does the graph shows. However, due to the fact that our MPI environment is constructed with three devices, the effect of increasing threads can be further accomplished.

After doing some research of this topic on google, we found that programmers tend to parallelize random forest with R language. There doesn't exist the case that parallelize random forest by C++ programming language, so it could be pretty challenging for there doesn't have much information for us to use.

In R language, there are lots of package like multicombine, H20, doMC to help you implementing Random Forest Classifier in a parallel way. In python, pyrallel provides you with an easy way to develop your project in parallel procedure.

However, methods mentioned focused on the construction of Random Forest Classifier. It makes us curious that if we can parallelize a single Decision Tree instead of the procedure of constructing the whole forest.



Related Works

As a result, we finish both parts, parallelization on Random Forest Creation and parallelization on

a single Decision Tree. That's a special part of our project.

Conclusion

Constructing random forest with lots of decision trees can be a good topic for parallel programming with the process of constructing a single decision tree is independent to others. In this paper, we have been determined the best place to implement parallelization, and the performance of each parallelization tools. We fully expect that there will have more other applications of parallelization and let our research become helpful to future researchers.

Github:
https://github.com/RaymondHuang210129/Parallel-Programming-Project-Universal-Decision-Tree-Classifier.git

References

[1] UCI, UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets.html.
[2] Tom T Mitchell (Ed.). 2017. *Machine Learning* (1st. ed.).
[3] Meduim, Random Forest Simple Explanation, https://bit.ly/2znDYCd
[4] scikit-learn, Decision Trees, https://scikit-learn.org/stable/modules/tree.html
[5] https://chtseng.wordpress.com/2017/02/10/%E6%B1%BA%E7%AD%96%E6%A8%B9-decision-trees/