High Performance Computing – 2025/2026

MsC in Informatics – ESTiG/IPB

Practical Work 2: **Random Forests Acceleration with MPI**

Goal: To accelerate a sequential implementation of the Random Forests method using MPI.

Preliminary Note:
This work comes with a ZIP archive with related resources (randforest-serial.zip).

Introduction:

“Random Forests […] build an "ensemble" (a group) of many individual decision trees to make more accurate predictions for both classification (e.g., spam/not spam) and regression (e.g., predicting house prices) tasks, by using random subsets of data and features to train each tree, then combining their results through voting or averaging for a robust final answer. They reduce the variance and overfitting issues of single decision trees, making them stable and reliable algorithms. How They Work: 1) Bootstrap Aggregation (Bagging): Each tree in the forest is trained on a random subset of the original training data (sampled with replacement); 2) Feature Randomness: At each split point in a tree, only a random subset of features (variables) is considered, not all of them, which decorrelates the trees; 3) Ensemble Voting/Averaging: in Classification, the final prediction is determined by majority vote (the class predicted by most trees); in Regression, the final prediction is the average of the predictions from all the individual trees.” [this summary was generated by AI]

Details and Methodology:

In this work you are given a serial implementation of the Random Forests method in C and you are required to develop two versions: an optimized serial version, and a parallel version based on the latter and built on the MPI programming model. In the end, there should be a clear incremental performance improvement, with the MPI version being faster (when using at least 2 tasks) than the optimized serial version, and this one being faster than the original provided serial version.

The serial version provided is the one available in [2], with several modifications and corrections. The code supports several Random Forests hyper-parameters [3] that affect its overall execution time and accuracy: a) k_folds or simply k (default 5) – number of times the model is trained and evaluated; each time the dataset is split into k subsets (or folds); one fold is held as a validation set while k-1 folds are used as training sets; b) n_estimators (default is 3) - number of trees in the random forest model; c) max_depth (default is 7) - maximum depth of a tree in the model; the number of splits that each decision tree is allowed to make; if the number of splits is too low, the model underfits the data and if it is too high the model overfits; common max_depth values are 3, 5, or 7; max_features (default is 3) - the number of dataset “columns” that are shown to each decision tree (the specific features that are passed to each decision tree can vary between each decision tree).

An adapted version of the [4] dataset is also provided as an ASCII file in CSV format. The last column is the label, and the previous are the features. The features pertain to medical images used to diagnose breast cancer, with a binary classification: label = 0 = Malign, label = 1 = Benign.

Note that the Random Forests implementation provided supports the search for the best hyper-parameters, but that will not be used in this work (you will use the hyper-parameter values specified in the next sections). Also, the implementation only supports classification in two classes (binary).

The work involves several stages; some may be executed on your personal computer, others must be executed in one or more worker nodes of the IPB cluster, and others may be executed on either systems; read the specific instructions for each stage regarding the required execution environment.

If your group number is G, your worker nodes are rocks-node-i and rocks-node-j, where i=G mod 8 and j = (i+1 mod 8). Ensure no one else is using your nodes when doing profiling or benchmarking.

All requests in red correspond to tasks that should be documented in your report. Provide each answer within a subsection with the same numbering of the question (e.g., b1, b2,…, c1, and so on).

a) Compiling and Testing the Original Serial Code

[this stage may be done on you personal computer, in the cluster frontend or in an worker node]

The companion ZIP archive contains a project file (*Makefile*), the source code of the serial version (*main.c* and the files in the *eval*, *model* and *utils* folders), and a the *wdbc.csv* dataset. Do not clone the code from the original git repository [2] – several files present in the git repository are absent from the ZIP file, once they are not needed for this work; also, the source code in the ZIP file was modified in order to correct several compilation warnings and implement certain enhancements.

Extract the ZIP archive, compile the code with make and execute the program with its default hyper-parameters, by issuing ./random-forest wdbc.csv --seed 0 , where wdbc.csv is the dataset and --seed 0 specifies 0 as the random number generator seed. In the next sections, you must always execute the program this way, only changing the hyper-parameters (in *main.c*) or compilation options (in the *Makefile*) as specifically instructed. With the default hyper-parameters and this program options, the program will run very fast (no more than 2s), yielding an accuracy of 93%.

b) Profiling the Original Serial Version

[this stage must be done on the rocks-node-i node assigned to your group; this is mandatory]

In this task you are required to use gprofng [4] (non-GUI version); to discover the options that you need to use, read the documentation on gprofng provided in the Virtual e-learning system.

Ensure the definitions in the *Makefile* are adequate to generate an executable suitable for profiling. Edit the *main.c* file and set k_folds, n_estimators and max_features all to 10. Regenerate the random-forest executable by issuing make clean; make and run it through gprofng to capture the profiling data. The profiling should take approximately 65 seconds.

b1) Provide the command used to run random-forest through gprofng.

After the profiling finishes, use again gprofng to display the name of the functions that have been executed, plus their respective **inclusive** CPU times (this should be the sorting criteria of the list).

b2) Provide the previous command and its output (for functions taking >30% of the the CPU time).

Analyze the inclusive CPU times. Verify if there are functions unrelated to the Random Forests algorithm whose execution time clearly stands out (for instance, related with memory management).

b3) State the “anomalous” hot-spots identified in the serial code.

c) Development, Profiling (*) and Benchmarking (**) of the Optimized Serial Version
[(*) and (**) must be done on the rocks-node-i node assigned to your group; this is mandatory]

Copy the randforest-serial folder into a new randforest-serial-opt folder. In the latter, modify the code to minimize the impact of the anomalous hot-spots and produce the optimized serial version.

c1) Provide a brief description of the changes implemented in the optimized serial code.

Execute and profile the new optimized version under the same conditions of the original serial version. Important: make sure the new version provides the same accuracy as the old version. Also, use again gprofng to show the functions executed, sorted by their respective **inclusive** CPU times.

c2) Provide the execution time of the optimized version and the output of the gprofng command (only for functions taking >30% of the CPU time).

Confirm that the anomalous hot-spots have disappeared or become irrelevant (regarding CPU-time). Now, use again gprofng, this time to display the call tree.

c3) Provide the previous command and its output (for functions taking >30% of the CPU time).

Study the code and identify the highest function(s) in the calltree that seem to be paralelizable.

c4) State the function(s) identified and explain why they are parallelizable. Also, provide the weight (% of total execution time) of the highest function in the calltree you decided to parallelize (note: if the weight is 100%, assume instead 99.99% so that Amdahl’s Law can be applied).

Important: if you were unable to produce an optimized serial version, you may proceed selecting, in the original serial version, a parallelizable hotspot function, directly related to the Random Forests algorithm, and work on that basis in the following sections; of course, this will affect your grade.

d) Applying Amdahl's Law

Considering the aggregated weight (%) of the code that you chose to parallelize, use Amdahl's Law to calculate the theoretical speedup (S_T) of the parallelization for N=2,4,6,8,10,12,14,16 homogeneous MPI tasks. Calculate also the corresponding theoretical efficiencies (E_T).

d1) Provide the next table with the missing values (?) of the theoretical speedups:

N	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
S_T	?	?	?	?	?	?	?	?

Table 1 - Theoretical Speedups

d2) Based on the Table 1 values, provide the next table with the theoretical efficiencies:

N	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
E_T **(%)**	?	?	?	?	?	?	?	?

Table 2 - Theoretical Efficiency (in percentage)

d3) Provide the theoretical speedup limit (and also present its calculation).

e) Parallelization

Copy the randforest-serial-opt folder into a new randforest-par-mpi folder. The latter is where modifications are to be done, in order to produce and MPI-based parallel version. Ensure the definitions in the *Makefile* are adequate to generate an executable suitable for benchmarking. Create also a hostfile with the proper node definitions, to use to run your application with MPI. Considering the specific values of *i* and *j* for your group, the file should contain 2 lines like these:

```
rocks-node-i slots=8 max-slots=8
rocks-node-j slots=8 max-slots=8
```

e.1) Development

[this stage may be done on you personal computer or in the cluster (frontend or worker)]

Decide how to divide the work (the hot-spot function(s) that you choose to parallelize) by tasks and how to synchronize them. Balance the work by tasks as uniformly as possible, and minimize their synchronization points. This will favor performance.

Consider the possibility of using collective operations (including `MPI_IN_PLACE` variants), derived data types, and other techniques to improve performance and code legibility. Minimize the synchronization points, and always verify the correctness of the results as you progress in coding. When possible/appropriate, make the main task (rank 0) to also participate as a worker task.

To make development and testing faster, during development (and for debugging purposes) you may use the same hyper-parameters as before (k_folds, n_estimators and max_features set to 10).

Start with 2 tasks and compare the accuracy with the one of the previous serial versions, to ensure the results are the same; only if they match, should you increment the number of MPI tasks used.

e1) Explain your parallelization strategy, and the MPI functions and techniques applied.

e.2) Benchmarking

[this stage must be done on the worker nodes assigned to your group; this is mandatory]

Set the hyper-parameters k_folds, n_estimators and max_features to 20 in the optimized serial version. In node rocks-node-i run this version 3 times, with a pause of 5s in-between each run. Register the execution times (in s), get its average (in s), and the relative standard deviation (absolute standard deviation divided by the average). All values must be truncated to the tenths.

Reapply the same hyper-parameter and the same evaluation methodology in the MPI version, for a number of tasks *N*=2,4,6,8,10,12,14,16 (3x8 executions overall). Important: if the relative standard deviation for a certain *N>=1* is >10%, discard the outlier execution time(s) and repeat the execution for that *N* until you have 3 execution times that ensure a relative standard deviation <= 10%.

e2) Provide the following table, filled with the average real execution time ($\overline{T}_R$, in seconds), and the relative standard deviation *(RSD*, in %), for each number of tasks (*N*). Also, provide a chart for $\overline{T}_R$.

N	**1**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
$\overline{T}_R$ ***(s)***	?	?	?	?	?	?	?	?	?
RSD (%)	?	?	?	?	?	?	?	?	?

Table 3 - Real Exec. Times of the Optimized Serial Version (N=1), and MPI Parallel Version (N>1)

e3) Based on Table 3, provide Table 4 filled with the real speedups (S_R), and the real parallel efficiency (E_R, in %). Explain how you calculated S_R and E_R. Also, provide a chart for S_R and E_R.

N	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
S_R	?	?	?	?	?	?	?	?
E_R ***(%)***	?	?	?	?	?	?	?	?

Table 4 - Real Speedups and Real Efficiencies of the MPI Parallel Version

e4) Based on Table 2 and 4, provide Table 5 filled with the ratio E_R/E_T (%) (this ratio tells how close is the parallel implementation to the Amdahl's Law prediction). Also, provide a chart for this ratio.

N	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
E_R/E_t	?	?	?	?	?	?	?	?

Table 5 - Closeness of the real to ideal efficiency of the MPI Parallel Version

f) Discussion

f1) Discuss the results achieved. How close are they to the expectations ? If far from Amdahl's predictions, what could be the reason(s))? What other optimizations could still improve the times ?

g) Groups

Students must keep the groups already formed. Any group change requires teacher authorization.

i) Deadline and Submission

The report (PDF) and code (all zipped) should be submitted in Virtual until **January 11th 2026**.

i) Plagiarism and Usage of AI Tools: If detected, Plagiarism or use of AI tools for coding dictates the work annulment. The report should not also be written with the aid of AI tools, and the new code that needs to be developed should not be generated by AI tools. This is your work, not other's.

j) References

[1] https://en.wikipedia.org/wiki/Random_forest
[2] https://github.com/SermetPekin/random-forests-c
[3] https://towardsdatascience.com/mastering-random-forests-a-comprehensive-guide-51307c129cb1/
[4] https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic