RAxML-Light v1.0.6 Manual
February 2012

Alexandros Stamatakis
Scientific Computing Group
Heidelberg Institute for Theoretical Studies
Alexandros.Stamatakis@h-its.org
raxml@h-its.org

What is RAxML-Light?

RAxML-Light is a strapped-down light-weight version of RAxML for inference of very large trees. It can only execute some very basic functions and is intended for computer-savvy users that can write little perl-scripts and have experience using queue submission scripts for clusters.
RAxML-Light only implements the CAT and GAMMA models of rate heterogeneity for DNA and protein data.
Issues with CAT-based branch lengths have been fixed, since the per-site rate categories are re-adjusted in such a way that the mean substitution rate is 1.0. Extensive tests (see [Izquierdo-Carrasco2011]) have shown that the CAT-based branch lengths are highly correlated with GAMMA-based branch lengths (Pearson correlation coefficient > 0.99), albeit the absolute values may be different.
Remember that, branch lengths on Maximum Likelihood (ML) trees are always just relative branch lengths.

What's new in RAxML-Light?

RAxML-Light has some new features that allow for reconstruction of huge trees.

1. It offers a fine-grain parallelization of the likelihood function with Pthreads for shared memory architectures and MPI (Message Passing Interface) for distributed memory architectures with low latency interconnects (such as Infiniband or Myrinet or, e.g., the dedicated interconnects on the IBM BlueGene systems).
2. It offers a light-weight checkpointing and restart capability. That is, typical HPC clusters usually have execution time limits of 24 hours, 48 hours or at best 1 week. This is problematic because inferences on large trees usually take much longer. Hence, a method to save the state of the search and re-start the program is required. This is directly implemented in software within RAxML-Light to save time when writing checkpoints and re-starting the program.
3. A special memory saving option **-S** that allows you to save memory and computations (albeit the program does not necessarily get faster) on gappy multi-gene alignments. For a large and very gappy (90% gaps) multi-gene alignment (about 120,000 taxa and 10 genes) the -S option yielded a memory consumption reduction from 70GB to only 19GB. Note that, results (log likelihood scores) may be slightly different, because I had to fiddle around a bit with the models to make this work.
4. Another memory saving option **-r** that allows to save memory by recomputing ancestral probability vectors on-the-fly instead of storing them in RAM. Also, the **-r** and **-S** options can be combined with each other.
5. A new protein model function called AUTO that will automatically select the best protein substitution matrix (w.r.t. to the likelihood) during the tree search.
6. Option to parse and write as well as read input alignments as binary data files. This can save a lot of time on huge datasets (e.g., a PHYLIP alignment file with a size of 27GB) because reading binary input data will be substantially faster and the parsing only need to be done once.
7. Improved parallel load balance for large (many partitions/genes) partitioned analyses
8. AVX-/FMA-vectorized versions of the likelihood function.

Version History:

New as of version 1.0.3:

- a little bug fix :-)

New as of version 1.0.4:

- a little bug fix for the restart from checkpoint option. This will not affect previous results.
- The so-called search convergence criterion (-D option) from the standard RAxML 728 version has been re-introduced (including restart capability) for tree searches on extremely large trees.

New as of version 1.0.5:

- implementation of GAMMA model of rate heterogeneity
- implementation of **-S** memory saving option for GAMMA-based model
- when re-started from a checkpoint the program will also report the total accumulated execution time, since the initial invocation
- Fixed some small bugs to be able to compute trees that require 1 TB of memory for storing the likelihood vectors (analyzing a DNA alignment with 1,481 taxa and 20,000,000 sites)
- Ability to parse, write, and read the input alignment as binary file (new **-G** and **-B** options)

New as of version 1.0.6:

- implementation of **-r** memory saving option
- implementation of **-Q** parallel load balance option
- implementation of AVX- & FMA-vectorized likelihood function implementations

Availability

All RAxML versions (and other code we are developing) are distributed via github at:
https://github.com/stamatak

Please check this site regularly for updates!

How do I compile RAxML-Light?

The distribution comes with six Makefiles for the sequential, Pthreads, and MPI-based versions. The sequential, Pthreads and MPI versions either use SSE3 128-bit wide SIMD (Single Instruction Multiple Data) or AVX 256-bit wide SIMD instructions which are offered by all recent AMD and Intel x86 architectures. The default compiler is gcc, but you may experiment with Intel icc or the Portland PGI compiler (although those compilers have not been tested by me). The parallel MPI version works with Intel icc and gcc and so far I have tested the MVAPICH2 (gcc and icc) and OpenMPI (gcc) compilers.

To compile the sequential versions, type:

```
make -f Makefile.SSE3.gcc
make -f Makefile.AVX.gcc
```

This will generate executables called `raxmlLight/raxmlLight-AVX`. If you want to compile the Pthreads version next, first type "`rm *.o`" in your terminal to remove the object files generated for compiling and linking the sequential program and type:

```
make -f Makefile.SSE3.PTHREADS.gcc
make -f Makefile.AVX.PTHREADS.gcc
```

This will produce executables called raxmlLight-Pthreads and raxmlLight-Pthreads-AVX respectively

Then, to compile the MPI version, first type "`rm *.o`" again and then type:

```
make -f Makefile.SSE3.MPI
make -f Makefile.AVX.MPI
```

which will generate executables called raxmlLight-MPI/raxmlLight-MPI-AVX. For this you will need to have a MPI compiler (called mpicc) installed on your system/cluster. If you don't know what a MPI compiler is, just talk to your local geek.
For testing purposes under Ubuntu Linux, it is probably easiest to install OpenMPI.
On our local cluster here at HITS using the 64-bit icc compiler v11.1 and MVAPICH2 yielded the best performance. It is really worth playing around with different compiler and MPI implementations, because this can yield performance differences of up to 30%.

WARNING: For performance reasons the number of partitions is hard-coded in the MPI version of the code. This is admittedly not very elegant, but we had to sacrifice flexibility for performance.
Thus, when running RaxML-Light, you may sometimes see it abort with the following or a similar error message:

raxmlLight-MPI: fineGrainMpi.c: startFineGrainMpi: Assertion `tr->NumberOfModels == 2' failed.

In such a case you will have to edit source file **axml.h**, and amend the following line:

#define NUM_BRANCHES 1
to read:

#define NUM_BRANCHES X

where X is the number of partitions in the dataset you want to analyze. Thereafter, please re-compile the MPI code and everything should work.

What can RAxML-Light compute?

As already mentioned, RAxML-Light is a strapped-down version of the standard RAxML distribution (currently version 7.2.8). It is meant to be used in combination with standard RAxML or the `parsimonator` (also available on the RAxML software page) program for analyzing very large trees. As such, the only thing RAxML-Light can do is to infer trees under Maximum Likelihood, given a pre-computed (e.g., with standard RAxML or the parsimonator) starting tree or checkpoint file. It can not do bootstraps (this requires scripting), searches on multiple trees, compute starting trees on its own etc. All of those tasks require scripting!

Here are the RAxML-Light program options, many are similar to the standard RAxML options.
The new options are highlithed in red and explained in more detail:

[raxmlLight|raxmlLight-PTHREADS|raxmlLight-MPI|
raxmlLight-AVX|raxmlLight-PTHREADS-AVX|raxmlLight-MPI-AVX]
-s sequenceFileName | -G binarySequnceFile
-n outputFileName
-m substitutionModel
-t userStartingTree | -R checkpointFileName

Unlike in standard RAxML, the user has to provide a comprehensive (containing all taxa) bifurcating starting tree to RAxML-Light, if no checkpoint file is specified via -R.

[-B]
[-c numberOfCategories]
[-D]
[-e likelihoodEpsilon]
[-f d|o]
[-h]
[-i initialRearrangementSetting]
[-M]
[-o outGroupName1[,outGroupName2[,...]]]
[-P proteinModel]
[-q multipleModelFileName]
[-Q]
[-r recomputationFraction]
[-R binaryCheckpointFile]
[-S]
[-T numberOfThreads]
[-v]
[-w outputDirectory]

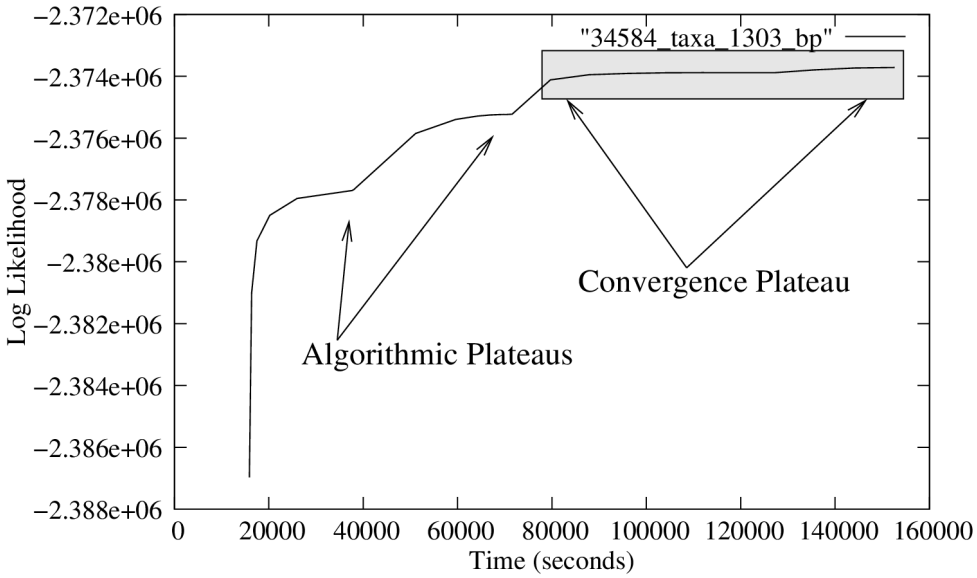
-B Parse phylip file and conduct pattern compression, then store the output in a binary file called sequenceFileName.binary that can be read via the "-G" option
ATTENTION: the "-B" option only works with the sequential version .

This is a new option as of v1.0.5. It will parse the input alignment, compute emprical base frequencies, parse the partitions, compress identical sites into site patterns, write the compressed alignment to a binary file and then exit. This can be used to accelerate parallel computations on huge datasets (e.g., phylip files with a size of 27GB) since all worker processes will not have to wait for the master to do this operation, prior to starting the actual tree search.

-c Specify number of distinct rate catgories for RAxML when modelOfEvolution is set to GTRCAT
Individual per-site rates are categorized into numberOfCategories rate categories to accelerate computations.

DEFAULT: 25

-D ML search convergence criterion. This will break off ML searches if the relative Robinson-Foulds topological distance between the pair of trees trees obtained from two consecutive lazy SPR cycles for improving the likelihood of the tree is smaller or equal to 1%.
Usage is highly recommended for very large datasets in terms of taxa. This option will avoid the program spending a large amount of processor time to obtain only slight imrpovements in the likelihood score.
This is outlined in the log likelihood score over execution time plot below for a single-gene dataset with 34,584 taxa. The -D option will help you to spare the time the program spends in the gray-shaded convergence plateau.



On trees with more than 500 taxa this will yield execution time improvements of approximately 50% while yielding only slightly worse trees. A detailed description and experiments with real data are provided in [Stamatakis2010].

DEFAULT: OFF

-e set model optimization precision in log likelihood units for final optimization of tree topology under GTRCAT

DEFAULT: 0.1

-f select algorithm:

"-f d": new rapid hill-climbing

DEFAULT: ON

"-f o": old and slower rapid hill-climbing without heuristic cutoff

DEFAULT for "-f": new rapid hill climbing

-G Read in a binary alignment file (instead of a text-based phylip file with "-s") that was previously generated with the "-B" option. This can substantially save time spent in input parsing for very large parallel runs .

This is a new option as of version 1.0.5 and highly recommended if you are using large datasets, because it will speed up the analysis pipeline substantially.

-h Display this help message.

-i Initial rearrangement setting for the subsequent application of topological changes phase

Recommendation: on datasets with more than 10,000 taxa you should set this values to -i 25 such that RAxML does not spend much time trying to estimate the best rearrangement radius. This is done when you don't specify -i. My empirical observation is that the estimate will always be 25 on very large datasets.

-m Model of Nucleotide or Amino Acid Substitution:

NUCLEOTIDES:

"-m GTRCAT": GTR + Optimization of substitution rates + Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency.

"-m GTRGAMMA": GTR + GAMMA model of rate heterogeneity. This uses 4 hard-coded discrete rates to discretize the GAMMA distribution.

AMINO ACIDS:

"-m PROTCATmatrixName[F]": specified AA matrix + Optimization of substitution rates + Optimization of site-specific evolutionary rates which are categorized into numberOfCategories distinct rate categories for greater computational efficiency.

"-m PROTGAMMAmatrixName[F]": specified AA matrix + GAMMA model of rate heterogeneity. This uses 4 hard-coded discrete rates to discretize the GAMMA dist ribution.

Available AA substitution models: DAYHOFF, DCMUT, JTT, MTREV, WAG, RTREV, CPREV, VT, BLOSUM62, MTMAM, LG, MTART, MTZOA, PMB, HIVB, HIVW, JTTDCMUT, FLU, AUTO, GTR .
With the optional "F" appendix you can specify if you want to use empirical base frequencies
Please note that for partitioned models you can specify the per-gene/per-partition AA model in the mixed model file (see standard RAxML manual for details). Also note that, if you estimate AA GTR parameters on a partitioned dataset, they will be linked (estimated jointly) across all partitions to avoid over-parametrization.
The AUTO "model" is a new option (as of RAxML-Light v1.0.2) that will automatically determine the best (with respect to the likelihood) protein substitution matrix. Essentially, every time RAxML re-estimates the model parameters during the tree search, it will loop over all protein substitution models (DAYHOFF, DCMUT, ..., FLU) except GTR and just use the model that yields the best likelihood.
***WARNING:** AUTO should currently not be used for partitioned datasets, we have not figured out yet what the best way to determine this is, especially when branch lengths are jointly estimated across partirtions. AUTO also can be called in two flavors AUTO and AUTOF (using empirical base frequencies instead of the pre-defined ones). Typically you would want to use AUTOF since the likelihood scores will be better.*

-M Switch on estimation of individual per-partition branch lengths. Only has effect when used in combination with "-q" Branch lengths for individual partitions will be printed to separate files . A weighted average of the branch lengths is computed by using the respective partition lengths

DEFAULT: OFF

-n Specifies the name of the output file.

-o Specify the name of a single outgrpoup or a comma-separated list of outgroups, eg "-o Rat" or "-o Rat,Mouse", in case that multiple outgroups are not monophyletic the first name in the list will be selected as outgroup, don't leave spaces between taxon names!

-P Specify the file name of a user-defined AA (Protein) substitution model. This file must contain 420 entries, the first 400 being the AA substitution rates (this must be a symmetric matrix) and the last 20 are the empirical base frequencies

-q Specify the file name which contains the assignment of models to alignment partitions for multiple models of substitution. For the syntax of this file please consult the standard RAxML manual.

-Q Enable a different load balancing scheme/algorithm which can yield substantial performance gains when the number of partitions >> the number of processors.

In particular under CAT this can lead to parallel performance improvements of over 50 per cent, for details please read the paper [Zhang2012] in the references.

DEFAULT: DISABLED

-r Specify the fraction of ancestral node vectors (0.100000 <= R < 1.000000) that will be allocated and stored in RAM to save memory/reduce the memory footprints of the analysis.

The ancestral probability vectors (also called conditional likelihood vectors) that are not stored in RAM will be recomputed on the fly, such that execution times will increase, but only to a relatively small extent. For details please read the paper [Izquierdo-Carrasco2012] in the references.

DEFAULT: DISABLED

-R Read in a binary checkpoint file called RAxML_binaryCheckpoint.RUN_ID_number .

This is a new option in RAxML-Light, instead of specifying a comprehensive starting tree via -t, you can use this option to restart a large-scale RAxML tree search that was interrupted, e.g., because you have used up your queue time. During the course of execution, RAxML will be writing files called RAxML_binaryCheckpoint.RUN_ID_number, where RUN_ID is the run name you specified via "-n RUN_ID" and "number" is simply the number of the checkpoint. Evidently, you would like to re-start RAxML-Light from the most recent checkpoint that was written.

-s Specify the name of the alignment data file in PHYLIP format

-S turn on memory saving option for gappy multi-gene alignments.

*This is a new option as of RAxML-Light v 1.0.2. For large and gappy datasets specify -S to save memory. This will produce slightly different likelihood values, may be a bit slower but can reduce memory consumption from 70GB to 19GB on very large and gappy datasets. For details please read the paper [Izquierdo-Carrasco2011] in the references. This option can be combined with the **-r** option.*

DEFAULT: OFF

-t Specify a user starting tree file name in Newick format .

If you don't specify -t, make sure that you specify -R, i.e., restart from a checkpoint, otherwise, the program will crash!

-T PTHREADS VERSION ONLY! Specify the number of threads you want to run. Make sure to set "-T" to at most the number of CPUs you have on your machine, otherwise, there will be a huge performance decrease!

-v Display version information

-w FULL (!) path to the directory into which RAxML shall write its output files

DEFAULT: current directory

Usage Examples

Suppose we want to compute a ML tree on dataset dna.phy (included in this distribution).
Initially, we will need to generate a starting tree, for this we can use, e.g., standard RAxML to compute a randomized stepwise addition parsimony tree:

```
./raxmlHPC-SSE3 -y -m GTRCAT -s dna.phy -p 12345 -n startingTree
```

This will just generate a parsimony starting tree (using the specified random number seed 12345) called RAxML_parsimonyTree.startingTree.

Now, we can invoke raxmlLight to do a tree search on this tree by typing:

```
./raxmlLight -m GTRCAT -s dna.phy -t  RaxML_parsimonyTree.startingTree
     -n TreeInference
```

The terminal output will look like this:

```
This is RAxML-Light version 1.0.0 released by Alexandros Stamatakis in February 2011.

Alignment has 358 distinct alignment patterns

Proportion of gaps and completely undetermined characters in this alignment: 2.21%

RAxML rapid hill-climbing mode

Using 1 distinct models/data partitions with joint branch length optimization

All free model parameters will be estimated by RAxML
ML estimate of 25 per site rate categories

Partition: 0
Alignment Patterns: 358
Name: No Name Provided
DataType: DNA
Substitution Matrix: GTR

RAxML was called as follows:

./raxmlLight -m GTRCAT -s dna.phy -t RAxML_parsimonyTree.startingTree -n TreeInference

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_0 likelihood: -4229.582317

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1 likelihood: -4229.582317

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_2 likelihood: -3992.577273

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_3 likelihood: -3991.211171

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_4 likelihood: -3991.196119

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_5 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_6 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_7 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_8 likelihood: -3991.193810

Overall Time for 1 Inference 1.772864
Likelihood   : -3991.193810

Final tree written to:                /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_result.TreeInference
Execution Log File written to:        /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_log.TreeInference
Execution information file written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_info.TreeInference
```

The interesting part here are the checkpoint files that are written by RAxML-Light. Suppose that our program is interrupted after the second checkpoint:
RAxML_binaryCheckpoint.TreeInference_1 we could restart the program like this:

```
./raxmlLight -m GTRCAT -n TreeInference_1 -s dna.phy
     -R RAxML_binaryCheckpoint.TreeInference_1
```

and get the following terminal output:

```
This is RAxML-Light version 1.0.0 released by Alexandros Stamatakis in February 2011.

Alignment has 358 distinct alignment patterns
```

Proportion of gaps and completely undetermined characters in this alignment: 2.21%

RAxML rapid hill-climbing mode

Using 1 distinct models/data partitions with joint branch length optimization

All free model parameters will be estimated by RAxML
ML estimate of 25 per site rate categories

Partition: 0
Alignment Patterns: 358
Name: No Name Provided
DataType: DNA
Substitution Matrix: GTR

RAxML was called as follows:

./raxmlLight -m GTRCAT -s dna.phy -R RAxML_binaryCheckpoint.TreeInference_1 -n TreeInference_1

RAxML Restart with likelihood: -4229.582316613224065804388374090194702148437500000000000

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_0 likelihood: -4229.582317

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_1 likelihood: -3992.577273

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_2 likelihood: -3992.577273

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_3 likelihood: -3991.211171

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_4 likelihood: -3991.196119

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_5 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_6 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_7 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_binaryCheckpoint.TreeInference_1_8 likelihood: -3991.193810

Overall Time for 1 Inference 1.521070
Likelihood : -3991.193810

Parsimony starting tree written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_parsimonyTree.TreeInference_1
Final tree written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_result.TreeInference_1
Execution Log File written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_log.TreeInference_1
Execution information file written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML_info.TreeInference_1

If we want to use the search convergence criterion, we would type:

```
./raxmlLight -m GTRCAT -s dna.phy -t  RaxML_parsimonyTree.startingTree
      -D -n TreeInference_CC
```

and obtain the following terminal output:

This is RAxML-Light version 1.0.4 released by Alexandros Stamatakis in May 2011.

Alignment has 358 distinct alignment patterns

Proportion of gaps and completely undetermined characters in this alignment: 2.21%

RAxML rapid hill-climbing mode

Using 1 distinct models/data partitions with joint branch length optimization

All free model parameters will be estimated by RAxML
ML estimate of 25 per site rate categories

Partition: 0
Alignment Patterns: 358
Name: No Name Provided
DataType: DNA
Substitution Matrix: GTR

RAxML was called as follows:

./raxmlLight -D -t RAxML_parsimonyTree.startingTree -s dna.phy -m GTRCAT -n TreeInference_CC

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_0 likelihood: -4229.582317

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_1 likelihood: -4229.582317

Best rearrangement radius: 5

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_2 likelihood: -3992.577273

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_3 likelihood: -3991.211171
ML search convergence criterion fast cycle 0->1 Relative Robinson-Foulds 0.142857

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_4 likelihood: -3991.196119

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_5 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_6 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_7 likelihood: -3991.193810

Checkpoint written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_binaryCheckpoint.TreeInference_CC_8 likelihood: -3991.193810

Overall Time for 1 Inference 1.590988
Likelihood : -3991.193810

Final tree written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_result.TreeInference_CC
Execution Log File written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_log.TreeInference_CC
Execution information file written to: /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.4/RAxML_info.TreeInference_CC

Running and restarting the parallel versions of RAxML-Light works in exactly the same way, e.g.,

```
./raxmlLight-PTHREADS -T 2 -m GTRCAT -s dna.phy -t RAxML_parsimonyTree.startingTree -n TreeInference
```

will execute the Pthreads version of RAxML-Light with two threads. Never ever run more threads than you have cores available on your system! This will lead to serious performance degradation!
For the MPI version this could look like this using the OpenMPI MPI implementation (note that MPI is just an interface specification and there exist various implementations of MPI such as MVAPICH, Intel MPI, OpenMPI etc.).

```
mpirun.openmpi -np 2 ./raxmlLight-MPI -T 2 -m GTRCAT -s dna.phy -t RAxML_parsimonyTree.startingTree -n TreeInference
```

This will start two MPI procsesses that will work simultaneously on computing the likelihood. Also note that, you should not oversubscribe your nodes, i.e., do not start more processes than there are physical cores available, because, once again, you will get a performance degradation. Also note that, the first part of this command “mpirun.openmpi -np 2” is installation-specific, that is, it depends on your local cluster and MPI installation. Here you should talk to the cluster admins, in order to figure out how to best run this.

In general, note that, RAxML-Light writes binary checkpoints, thus you can typically not run RAxML for some time on one computer architecture and then restart it on another, different one. The general assumption is that RAxML-Light will be re-started on the same processor type it was stopped before.

If you want to run the above example with the memory saving option you'd just type:

```
mpirun.openmpi -np 2 ./raxmlLight-MPI -T 2 -m GTRCAT -S -s dna.phy -t RAxML_parsimonyTree.startingTree -n TreeInference
```

Now, let's have a look at parsing and using binary alignment files.

To parse a phylip file and write it to a binary file you'd type:

```
./raxmlLight -m GTRCAT -s dna.phy -t RAxML_parsimonyTree.startingTree -B -n Parse
```

The output will look like this:

```
stamatak@exelixis:~/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.5$ ./raxmlLight -m GTRCAT -s dna.phy -t RAxML_parsimonyTree.startingTree -B -n Parse

Binary and compressed alignment file written to file /home/stamatak/Desktop/GIT/RAxML-LIGHT/RAxML-Light-1.0.5/dna.phy.binary

Parsing completed, exiting now ...
```

To infer a tree using the binary file you'd now type:

```
./raxmlLight -m GTRCAT -t RAxML_parsimonyTree.startingTree -G dna.phy.binary -n Inf
```

And get a standard tree inference output.

Keep in mind that:

1. **Parsing and binary file writing always needs to be done with the sequential version**
2. **If you are using a partition file, you must pass it to the parser, otherwise this will not be written/compressed correctly into a binary file!**

Frequently Asked Questions

Q: If I have a large shared-memory node, should I use MPI or pthreads?
A: It is probably better to use the MPI version, since the collective communication routines as implemented in MPI are better optimized than the Pthreads collective communication routines I implemented for the RAxML Pthreads version.

Q: How can I do bootstraps with RAxML-Light?
A: Here, you will once again have to use the standard RAxML version first and do some scripting. Initially you should use standard RAxML to generate bootstrap replicate files, by typing e.g.:

```
./raxmlHPC-SSE3 -# 100 -b 12345 -f j -m GTRCAT -s dna.phy -n REPS
```

This will generate 100 BS replicates as indicated in the terminal output:

```
Printing replicate 0 to /home/stamatak/Desktop/GIT/RAxML-LIGHT/dna.phy.BS0
Printing replicate 1 to /home/stamatak/Desktop/GIT/RAxML-LIGHT/dna.phy.BS1
Printing replicate 2 to /home/stamatak/Desktop/GIT/RAxML-LIGHT/dna.phy.BS2
Printing replicate 3 to /home/stamatak/Desktop/GIT/RAxML-LIGHT/dna.phy.BS3
.....
Printing replicate 98 to /home/stamatak/Desktop/GIT/RAxML-LIGHT/dna.phy.BS98
Printing replicate 99 to /home/stamatak/Desktop/GIT/RAxML-LIGHT/dna.phy.BS99
```

Then, you would use standard RAxML again to compute parsimony starting trees for each replicate e.g., via a respective perl script:

```
#base name of bootstrap replicate file names
$bsname = "dna.phy.BS";

#parsimony random number seed range
$range  = 1000000000;

# lopp over 100 bootstrap replicates
for($i = 0; $i < 100; $i++)
 {

   # generate a random number seed for the randomized stepwise addition parsimony tree building process
   $random_number = int(rand($range));

   # build the command line string
   $command = "./raxmlHPC-SSE3 -y -s ".$bsname.$i." -m GTRCAT-n T".$i." -p ".$random_number." \n";

   # execute the command
   system($command);
}
```

This will generate 100 parsimony starting trees called RAxML_parsimonyTree.T0 .... RAxML_parsimonyTree.T99. Note that, it won't make much sense to use the Pthreads version of standard RAxML to compute parsimony starting trees, because (i) it's fast (ii) the parallel efficiency of the Pthreads-based parsimony implementation sucks. Once you have the starting trees you can then launch, e.g., the Pthreads version of RAxML-Light (this is an example perl script that work son our cluster at HITS) as follows to compute the 100 Bootstrap trees:

```
#base name of bootstrap replicate file names
$bsname = "dna.phy.BS";

for($i = 0; $i < 100; $i++)
 {
   #open a queue submission file that we will populate with commands
   open (F, " >bsinf".$i);

   # the stuff below is cluster and installation-specific
   print F "#!/bin/bash\n";

   print F "#\$ -S /bin/bash\n";

   print F "#\$ -cwd\n";

   print F "#\$ -j y\n";

   print F "#\$ -pe impi48 48\n";

   print F "#\$ -q test-48.q\n";
```

```
  print F "# source module\n";

  print F ". /etc/profile.d/modules.sh\n";

  print F "module load sge gcc/4.3.4\n";

 # here we assemble the command line that will execute the Pthreads version of raxmlLight on
 # shared-memory nodes with 48 cores and 48 threads
 printf F "./raxmlLight-PTHREADS -T 48 -s ".$bsname.$i." -m GTRCAT -t RAxML_parsimonyTree.T".$i." -n BINF_".$i."\n";

  # done editing the file, now just close it
  close(F);

  # now we can automatically submit the job to the queing system :-)
  system("qsub bsinf".$i);
}
```

The above script will automatically submit 100 tree inference jobs using raxmlLight-PTHREADS to the batch queuing system on our cluster here at HITS. With some slight modifications, the above script should work on most typical cluster installations.

Q: What is a large dataset that would be appropriate for RAxML-Light?
A: Large datasets are either many-taxon datasets with more than 10,000 taxa and a couple of genes (e.g., 10-20 genes) or datasets with a couple of hundred taxa and 1000 genes.

Q: How do I cite RAxML-Light?
A: For the time being just cite it as:

A. Stamatakis, A.J. Aberer, C. Goll, S.A. Smith, S.A. Berger, F. Izquierdo-Carrasco: "RAxML-Light: A Tool for computing TeraByte Phylogenies", Heidelberg Institute for Theoretical Studies, Exelixis-Rapid-Research-Dissemination-Report-2012-3, March 2012.

Q: Under which license is RAxML-Light available?
A: It's available under GNU GPL version 3 or later.

Q: How does the MPI version work?
A: The MPI version just works in an analogous way as the Pthreads version, i.e., multiple MPI processes (in analogy to multiple threads) work concurrently on computing the likelihood on the same underlying tree topology. Instead of communicating and synchronizing computations via shared memory areas, the processes communicate using the Message Passing Interface over some low latency network. A low latency network is required because the processes need to communicate with each other several thousand times per second of execution time. However, they do not exchange much data with each other, so network bandwidth is not a problem. The MPI version is a de novo implementation of the fine-grain MPI parallelization concept for the phylogenetic likelihood function that was originally presented in this paper here [Ott2007]: http://wwwkramer.in.tum.de/exelixis/pubs/SC2007.pdf

Q: When using the Pthreads or MPI version, can I restart the program from a checkpoint using a different number of processes/threads than in the run that produced the checkpoint?
A: Yes, you can :-)

Q: Is there a script to automatically restart RAxML-Light on a cluster with a say 48-hour queue limit?
A: Yes, there is one. John Cazes from the Texas Advanced Computing Center has kindly implemented such scripts which are included in this distribution. They are called: **checkpoint_bash.sge** and **checkpoint_tcsh.sge** and will work on the TACC supercomputer. If you know that your job will require approximately a total of 10 48-hour jobs you can simply submit 10 such scripted jobs and the queuing system will handle the rest. Note that, those scripts will require some adaptation to your local system, if you are not an expert you should ask your local geeks. We have also included the adapted version of this script for our local cluster at HITS called: **checkpoint_bash_hits_cluster.sge**.

Q: What are the largest trees that can/have be computed with RAxML-Light?
A: In terms of #taxa a tree with almost 120,000 taxa and a couple of genes. This can run nicely on a single multi-core node with 48 cores and 128GB of memory under the CAT model. I have also analyzed a datasets with 1,481 taxa and 20,000,000 sites using 672 cores and almost 1TB of RAM under the CAT model.

REFERENCES

[Ott2007] M. Ott, J. Zola, S. Aluru, A. Stamatakis: “Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L”. In Proceedings of IEEE/ACM Supercomputing (SC2007) conference, Reno, Nevada, November 2007.

[Stamatakis2010] A. Stamatakis: "Phylogenetic Search Algorithms for Maximum Likelihood". In M. Elloumi, A.Y. Zomaya, editors. Algorithms in Computational Biology: techniques, Approaches and Applications, John Wiley and Sons, 2010.

[Izquierdo-Carrasco2011] F. Izquierdo-Carrasco, S.A. Smith, A. Stamatakis: "Algorithms, Data Structures, and Numerics for Likelihood-based Phylogenetic Inference of Huge Trees". BMC Bioinformatics 12:470 2011. OPEN ACCESS

[Izquierdo-Carrasco2012] F. Izquierdo-Carrasco, J. Gagneur, A. Stamatakis: "Trading Memory for Running Time in Phylogenetic Likelihood Computations", 2012 Bioinformatics conference, Vilamoura, Portugal. PDF

[Zhang2012] J. Zhang**,** A. Stamatakis: "The Multi-Processor Scheduling Problem in Phylogenetics". 11th IEEE HICOMB workshop (in conjunction with IPDPS 2012). PDF