The ExaML v3.0.X Manual

July 17 2017

Alexandros Stamatakis
Heidelberg Institute for Theoretical Studies

Introduction

Exascale Maximum Likelihood (ExaML) is a code for phylogenetic inference using MPI. This code implements the popular RAxML search algorithm for maximum likelihood based inference of phylogenetic trees.
ExaML is a strapped-down light-weight version of RAxML for phylogenetic inference on huge datasets. 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. ExaML only implements the CAT and GAMMA models of rate heterogeneity for binary, DNA, and protein data.
ExaML uses a radically new MPI parallelization approach that yields improved parallel efficiency, in particular on partitioned multi-gene or whole-genome datasets. It also implements a new load balancing algorithm that yields better parallel efficiency.
It is up to 4 times faster than its predecessor RAxML-Light [1] and scales to a larger number of processors.

Please cite the following paper when using ExaML:

> A. Kozlov, A.J. Aberer, A. Stamatakis: “ExaML Version 3: A Tool for Phylogenomic Analyses on Supercomputer”. In *Bioinformatics* (2015) doi: 10.1093/bioinformatics/btv184
>
> Advance on-line publication link: http://bioinformatics.oxfordjournals.org/content/early/2015/03/28/bioinformatics.btv184.abstract?keytype=ref&ijkey=xZdxpOZMjzxWlv9

Similar to RaxML-Light whose development has been discontinued, ExaML also implements checkpointing, SSE3, AVX vectorization and memory saving

techniques. The usage philosophy is similar to RAxML, albeit it only implements a subset of RAxML features. It will be very helpful to read the RAxML manual, before getting started with ExaML.

Support is provided via the RAxML google group:
https://groups.google.com/forum/?hl=de&fromgroups#!forum/raxml.
Please refrain from sending emails with questions directly to the authors of the code! If you ask your questions on the google group you can first search it for similar questions and then post your question. The reply will then be visible and searchable for everybody and make it much easier for us to provide good and timely support.

ExaML has been used, for instance, in the following two science papers:

> http://www.sciencemag.org/content/346/6215/1320.abstract
> http://www.sciencemag.org/content/346/6210/763

[1] A. Stamatakis, A.J. Aberer, C. Goll, S.A. Smith, S.A. Berger, F. Izquierdo-Carrasco: "RAxML-Light: A Tool for computing TeraByte Phylogenies", *Bioinformatics* http://bioinformatics.oxfordjournals.org/content/28/15/2064.short

Compiling the code

There is actually a suite of tools required to accomplish the task of inferring large phylogenies.

When you download the code there will be four relevant directories:

```
examl
manual
parser
testData
```

Let's first compile the parse-examl by typing:

```
cd parser
make -f Makefile.SSE3.gcc
```

This should produce an executable called `parse-examl`

Next, let's compile ExaML. Note that, you will require a MPI compiler, usually called `mpicc` to accomplish this task (if you don't know what this is ask your local geeks; the OpenMPI MPI implementation is usually relatively easy to install: http://www.open-mpi.de/).

Type:

```
cd ../examl/
```

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

→ this will compile the SSE3 vectorized version of the code, the binary will be called `examl`

If you have a recently purchased system you can also try to compile the AVX-ion vectorized version of ExaML (for performance and further details please refer to the on-line supplement of [1]). Whenever you can, please use the AVX version of the code which is faster!

first remove the object files created when you compiled the SSE3 version of `examl`:

```
rm *.o
```

then compile the AVX version by typing:

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

this should produce a binary called: `examl-AVX`

We recently also implemented a hybrid MPI-OpenMP version of ExaML for x86 CPUs, that sometimes scales better when using a huge number of cores (see Section “Production Level Analyses”), this can be compiled by using the appropriate Makefiles. For instance, typing:

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

will produce a binary called: `examl-OMP-AVX`

Usage Example & Command Line Options

I will walk you through a simple example of how to use the code on my laptop.

The Parser

Initially we will have to transform an alignment file in relaxed phylip format (as used for RAxML) into a binary file format that can be read by ExaML. The main reason for this is to allow ExaML to read this file faster and not waste any valuable parallel computing time for this simple pre-processing task. This might seem to be a bit complicated but saves a lot of CPU idle time when starting ExaML with a huge number of cores. To better understand this you can have a look at the Wikipedia site for Amdahl's law: http://en.wikipedia.org/wiki/Amdahl%27s_law

Help for the command line arguments of the `parse-examl` can be obtained by typing:

```
./parse-examl -h

parse-examl
     -s sequenceFileName
     -n outputFileName
     -m substitutionModel
     [-c]
     [-q]
     [-h]

     -m   Type of Data:

          For DNA data use:    DNA
          For AA data use:     PROT
          For Binary data use: BIN

     -c   disable site pattern compression

     -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 RAxML manual.
     -h   Display this help message.
```

Now let's use the parse-examl to transform the test file into a binary file by typing:

```
./parse-examl -s ../testData/49 -m DNA -n 49.unpartitioned
```

So we are transforming an unpartitioned phylip file with 49 DNA sequences into a binary file. The parser directory will now contain a file called:

```
49.unpartitioned.binary
```

that can be read by ExaML.

The parser will also print the following useful output to screen:

```
gappyness: 0.074048
Pattern compression: ON

Alignment has 51 completely undetermined sites that will be
automatically removed from the binary alignment file

Your alignment has 628 unique patterns

Under CAT the memory required by ExaML for storing CLVs and tip
vectors will be
```

```
1015476 bytes
991 kiloBytes
0 MegaBytes
0 GigaBytes

Under GAMMA the memory required by ExaML for storing CLVs and
tip vectors will be
3969588 bytes
3876 kiloBytes
3 MegaBytes
0 GigaBytes

Please note that, these are just the memory requirements for
doing likelihood calculations!
To be on the safe side, we recommend that you execute ExaML on a
system with twice that memory.

Binary and compressed alignment file written to file
49.unpartitioned.binary

Parsing completed, exiting now ...
```

So let's see why this is useful information. First of all, the proportion of gaps/missing data is interesting (it's 7.4% here), since this can be used as a guide to determine if it makes sense to use the `-S` memory saving option for gappy alignments in ExaML. Roughly speaking, using that option in ExaML makes sense if the gappyness is above 30%, i.e., above 0.3.
Next, the parser tells you that there are 51 alignment sites in the alignment that consist of fully undetermined characters (`N, ?, -,` etc.) and contain no signal. This is a bot worrisome, since something may have gone wrong during alignment assembly. Nonetheless, the parser will automatically remove these sites.
Next, the parser tells you how many distinct site patterns the alignment has which is important for figuring out the computational requirements in terms of memory usage.
Finally, and this is very useful, the parser tells you how much memory will be required by ExaML to store the conditional likelihood arrays for this alignment under the per site rate (PSR) and GAMMA models of rate heterogeneity. These numbers allow you to determine how much memory the system you are going to use for tree inference with ExaML needs to have. Here it tells us that it requires 3MB, to be on the safe side let's multiply this with 2, hence we need a system with at least 6MB RAM. This can easily be executed on my laptop :-)

Note that, if you have an alignment that requires, for instance, 500GB of RAM it can run on 50 cores with 1GB RAM per core, since ExaML distributes the memory requirements evenly over all cores.

If we want to partition the data, we will have to pass a standard RAxML partition file (see RAxML manual for a very detailed explanation) to `parse-examl`, e.g:

```
./parse-examl -s ../testData/49 -q ../testData/49.model -m DNA
    -n 49.partitioned
```

This will generate a file called

```
49.partitioned.binary
```

WARNING: Note that, every time you change the partition scheme, you will have to re-generate a binary alignment file that encodes the new partitioning scheme!
A common question is how one can tell the parser to use a specific protein substitution model, e.g., `WAGF` (WAG with empirical base frequencies) or to optimize the base frequencies for DNA data (`DNAX`) via a Maximum Likelihood estimate when there is only one single partition. In this case, you will have to specify a one line partition file, spanning the entire aligment. Assume your alignment has 1000 sites, then, the respective line in the protein partition file would look like this:

```
WAGF, p1=1-1000
```

and for DNA data with a ML estimate of base frequencies:

```
DNAX, p1=1-1000
```

Generating Starting Trees

Now, we are almost ready to start an ExaML run. However, you also need to provide a starting tree to ExaML by using, for instance the `parsimonator` (see http://sco.h-its.org/exelixis/web/software/parsimonator/index.html) code or standard RAxML.

By using the following command you can generate a randomized stepwise addition order parsimony tree with RAxML:

```
raxmlHPC-AVX -y -m GTRCAT -p 12345 -s ../testData/49 -n
    StartingTree
```

Note that, if possible you should use the `AVX` version of RAxML since the parsimony calculations have been optimized with `SSE3` and `AVX` vector intrinsics and the `AVX` version is the fastest. In addition, when generating several parsimony trees make sure to pass different random seeds via `-p`, otherwise you will always obtain the same tree!

To generate a complete random starting tree type:

```
raxmlHPC-AVX -y -d -m GTRCAT -p 12345 -s ../testData/49 -n
    RandomStartingTree
```

As above, for generating several random starting trees make sure to pass different random seeds via `-p`, otherwise you will always obtain the same tree!

ExaML

What can ExaML compute?

As already mentioned, ExaML is a strapped-down version of the standard RAxML distribution. It is meant to be used in combination with standard RAxML or the `parsimonator` (also available at www.exelixis-lab.org/software.html) program for analyzing very large trees under parsimony. As such, the only three things ExaML can do is to infer trees under Maximum Likelihood, given a pre-computed (e.g., with standard RAxML or the `parsimonator`) starting tree, to evaluate a set of given, fixed trees under Maximum Likelihood, or to sample quartets in a similar way as RAxML.
It can not do bootstraps, searches on multiple starting trees, or compute starting trees on its own. This requires some scripting. The rationale is that all these pre-computation steps can be done on a smaller server and will therefore not consume valuable CPU time on the large clusters and supercomputers that ExaML has been designed for.

But let's get started with ExaML now. We have included a starting tree in the `testData` directory, such that we can now run ExaML.

For this we will have to change into the ExaML directory by typing:

```
cd ../examl
```

On-line help regarding ExaML command line options can be obtained by typing:

```
./examl -h
```

which will yield the following output:

```
examl|examl-AVX
     -s binarySequenceFileName
     -n outputFileNames
     -m rateHeterogeneityModel
     -t userStartingTree|-R binaryCheckpointFile|-g constraintTree
     -p randomNumberSeed
     [-a]
     [-B numberOfMLtreesToSave]
     [-c numberOfCategories]
     [-D]
     [-e likelihoodEpsilon]
     [-f d|e|E|o|q]
     [-h]
     [-i initialRearrangementSetting]
     [-I quartetCheckpointInterval]
```

```
[-M]
[-r randomQuartetNumber]
[-S]
[-v]
[-w outputDirectory]
[-Y quartetGroupingFileName]
[--auto-prot=ml|bic|aic|aicc]
```

`-a` use the median for the discrete approximation of the GAMMA model of rate heterogeneity

DEFAULT: OFF

Comment: typically using the median instead of the average yields slightly better likelihood values, without increasing the number of parameters. Hence, enabling this option is recommended.

`-B` specify the number of best ML trees to save and print to file

Comment: This can be used to not only save the final best-scoring ML tree, but also trees that were encountered during the tree search.

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

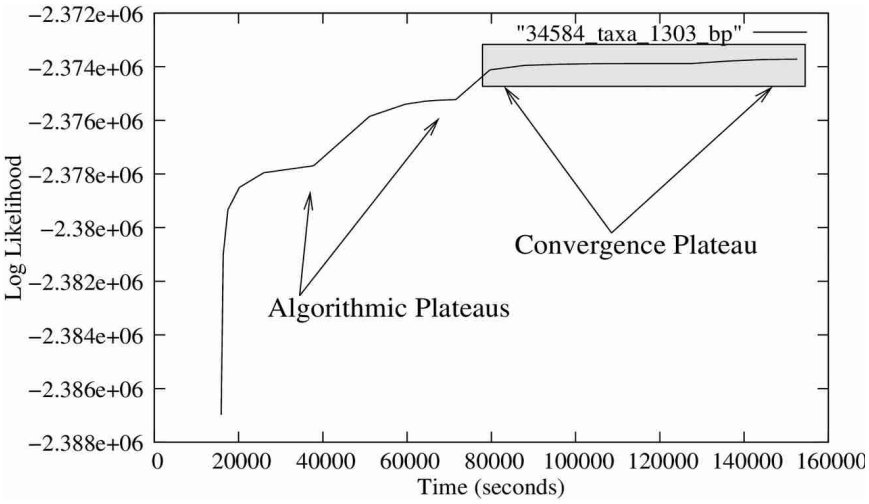
DEFAULT: 25

Comment: It is probably best to leave this unchanged.
A common user mistake is to think that this can also be used to set the number of discrete rate categories the GAMMA model of rate heterogeneity uses. This is wrong! Under GAMMA ExaML will always use 4 discrete rate categories, this setting can not be changed!

`-D` ML search convergence criterion. This will break off ML searches if the relative Robinson-Foulds distance between the trees obtained from two consecutive lazy SPR cycles is smaller or equal to 1%. Usage recommended for very large datasets in terms of taxa.
On trees with more than 500 taxa this will yield execution time improvements of approximately 50% while yielding only slightly worse trees.

DEFAULT: OFF

Comment: For performance details of this option, please see the following book chapter: A. Stamatakis: "Phylogenetic Search Algorithms for Maximum Likelihood". In M. Elloumi, A.Y. Zomaya, editors. Algorithms in Computational Biology: techniques, Approaches and Applications, 547-577, John Wiley and Sons, 2011.
An example 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.



`-e` `set model optimization precision in log likelihood units for final optimization of model parameters`

`DEFAULT: 0.1`

Comment: Analogous to the correspondig standard RAxML option.

`-f` `select algorithm:`

`"-f d": new rapid hill-climbing`

`DEFAULT: ON`

`"-f e": compute the likelihood of a bunch of trees passed via -t . This option will do a quick and dirty optimization without re-optimizng the model parameters for each tree.`

`"-f E": compute the likelihood of a bunch of trees passed via -t this option will do a thorough optimization that re-optimizes the model parameters for each tree.`

Comment: Note that, checkpointing is also implemented for the `-f e` and `-f E` options!

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

DEFAULT for "-f": new rapid hill climbing

Comment: For very broad phylogenomic datasets we recommend that users also test the performance of the `-f o` option. The performance differences between -f o and -f d are described in the following paper: A. Stamatakis, F. Blagojevic, C.D. Antonopoulos, D.S. Nikolopoulos: "Exploring new Search Algorithms and Hardware for Phylogenetics: RAxML meets the IBM Cell". In Journal of VLSI Signal Processing Systems, 48(3):271-286, 2007.

"-f q": fast quartet calculator

Comment: Analogous to the respective RAxML option for sampling quartets. If neither `-r` nor `-Y` is specified, this will sample all possible quartets from a given alignment and tree. The tree is only required to estimate ML model parameters for quartet calculations on it. Unlike RAxML, quartet calculations in ExaML are not parallelized over quartets, but the likelihood calculations for each quartet are conducted in parallel. As a consequence this will only scale on broad phylogenomic alignments, just like ExaML tree searches as well. An advantage is that model parameter optimization and quartet evaluation can be conducted in one single run. Checkpointing is also implemented.

-g Pass a multi-furcating constraint tree to ExaML. The tree needs to contain all taxa of the alignment!
When using this option you also need to specify a random number seed via "-p" !

Comment: Mostly analogous to the corresponding standard RAxML option. However, in contrast to RAxML, the constraint tree needs to contain all taxa of the alignment, that is, ExaML can not add taxa that are not contained in the constraint tree. The random number seed is required because multi-furcations will initially be resolved randomly and then refined via a Maximum Likelihood search.

-h Display this help message.

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

Comment: Analogous to the corresponding standard RAxML option.

```
-I    Set after how many quartet evaluations a new checkpoint
      will be printed.

      DEFAULT: 1000
```

Comment: Writing checkpoints too frequently can impact performance. However, performance would also decrease if I had implemented a given time interval for writing checkpoints since this would require frequent communication between all process. You may want to tune this such that a checkpoint is written every 1-2 hours or so.

```
-m    Model of rate heterogeneity

      select "-m PSR" for the per-site rate category model (this
      used to be called CAT in RAxML)

      select "-m GAMMA" for the gamma model of rate heterogeneity
      with 4 discrete rates
```

Comment: The selected model of rate heterogeneity will be applied to all partitions; one can not assign `PSR` to some partitions and `GAMMA` to some others. Also note that, on large phylogenomic datasets with an insuficiient number of taxa (below roughly 100) `PSR` may yield suboptimal results. We therefore recommend to always also do some initial tree searches under `GAMMA`.

```
-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
```

Comment: Analogous to the corresponding standard RAxML option.

```
-n    Specifies the name of the output file.
```

Comment: Analogous to the corresponding standard RAxML option.

```
-p    Specify a random number seed, required in conjunction with
      the "-g" option for constraint trees
```

Comment: Analogous to the corresponding standard RAxML option.

```
-r   Pass the number of quartets to randomly sub-sample from
     the possible number of quartets for the given taxon set.
     Only works in combination with -f q !
```

Comment: Analogous to the corresponding RAxML option. Note that we are essentially drawing quartets at random with replacement here.

```
-R   read in a binary checkpoint file called
     ExaML_binaryCheckpoint.RUN_ID_number
```

Comment: This can be used to continue a tree search that was interrupted, for instance, if ExaML runs longer than the CPU time limit (typically 24 or 48 hours) on the cluster you are using. Note that, if you restart with `-R` you can omit specifying `-t` because the starting tree is not required anymore and the intermediate tree has been stored in the checkpoint file.
Note that, checkpoints generated by previous ExaML versions may be be incompatible with the current version. ExaML will check the version number that is stored in the checkpoint and exit with an error in such cases.
Also, make sure that a run starting from a checkpoint uses the same command line arguments as the run that produced the checkpoint. If the command line arguments don't match, ExaML will also exit with an error.

```
-s   Specify the name of the BINARY alignment data file
     generated by the parser component
```

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

Comment: Equivalent to the standard RAxML `-U` option. Corresponding performance results are described in the following paper http://www.biomedcentral.com/1471-2105/12/470

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

Comment: To avoid tree formatting issues we highly recommend that you generate starting trees either with `parsimonator` or standard RAxML. ExaML is guaranteed to properly parse these trees. If you are re-starting ExaML from a checkpoint file, you don't need to specify the starting tree once again.

```
-v   Display version information

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

     DEFAULT: current directory
```

Comment: Analogous to the corresponding standard RAxML option.

```
-Y   Pass a quartet grouping file name defining four groups from
     which to draw quartets
     The file input format must contain 4 groups in the
     following form:
     (Chicken, Human, Loach), (Cow, Carp), (Mouse, Rat, Seal), (Whale, Frog);
     Only works in combination with -f q !
```

Comment: Analogous to the corresponding standard RAxML option. The required file format is exactly the same!

```
--auto-prot=ml|bic|aic|aicc When using automatic protein model
     selection you can chose the criterion for selecting these
     models.
     ExaML will test all available prot subst. models except for
     LG4M, LG4X and GTR-based models, with and without empirical
     base frequencies.
     You can chose between ML score based selection and the BIC,
     AIC, and AICc criteria.

     DEFAULT: ml
```

Comment: Analogous to the corresponding standard RAxML option.

Now, given that we know all command line options of `examl` and that we have used `parse-examl` and RAxML or `parsimonator` to generate binary alignment files and starting trees, we are ready to start an ExaML run:

To execute a simple run using two processors we type:

```
mpirun.openmpi -np 2 ./examl-AVX -t ../testData/49.tree -m GAMMA
     -s ../parser/49.unpartitioned.binary -n T1
```

This run will produce the following three plain-text output files:

`ExaML_modelFile.T1` contains the model parameters (alpha, GTR rates, base frequencies, tree length) for each partition of

	the final tree
`ExaML_result.T1`	contains the ML tree in Newick format
`ExaML_log.T1`	contains a log of the log likelihoods encountered for different trees during the tree search
`ExaML_info.T1`	contains execution information, similar to the RAxML_info file

Then, the run also produces a set of binary checkpoint files numbered like this:

```
ExaML_binaryCheckpoint.T1_0
ExaML_binaryCheckpoint.T1_1
…
ExaML_binaryCheckpoint.T1_13
```

The higher the number at the end, the later the checkpoint was written! Thus, if your ExaML run was interrupted for some reason, you have too look for the checkpoint file with the highest number and re-start from there. Assume, that the latest checkpoint written, was `ExaML_binaryCheckpoint.T1_5` you can re-start ExaML from this checkpoint as follows:

```
mpirun.openmpi -np 2 ./examl-AVX
     -s ../parser/49.unpartitioned.binary
     -R ExaML_binaryCheckpoint.T1_5 -m GAMMA -n T1_RESTART
```

Note that, here you do not need to specify the starting tree via `-t` again, specifying the checkpoint file name via `-R` suffices.

Now, for a tree inference on a partitioned dataset we type:

```
mpirun.openmpi -np 2 ./examl-AVX  -t ../testData/49.tree
     -m GAMMA -s ../parser/49.partitioned.binary -n T2
```

This will produce the same output files as above.

For better performance do not forget to experiment with the `-S` option if your dataset has a large fraction of missing data! The fraction of missing data is indicated by the parser component.

Now, assume that you have conducted, for instance, several ML searches under the Per site model of rate heterogeneity (`-m PSR`) and want to evaluate these trees now under the gamma model of rate heterogeneity.

You will first need to concatenate all ML trees you want to evaluate in a single file, using, for instance the Linux `cat` commad:

```
cat ExaML_result.* > treeSet
```

Then you can invoke ExaML either as follows:

```
mpirun.openmpi -np 2 ./examl-AVX
     -s ../parser/49.unpartitioned.binary -t treeSet -f e
     -m GAMMA -n T3
```

This will produce several output files:

```
ExaML_TreeFile.T3
```

This file contains the evaluated trees with branch lengths in the same order as the input tree file treeSet.

Then, assuming that the input tree set contained two trees, ExaML will produce two additional files:

```
ExaML_modelFile.T3.0
ExaML_modelFile.T3.1
```

These files contain information about the estimated model parameters (alpha, GTR rates, tree length, base frequencies) for each partition of the dataset. A separate file is generated for each input tree.

Alternatively, you can invoke ExaML as follows to evaluate trees:

```
mpirun.openmpi -np 2 ./examl-AVX
     -s ../parser/49.unpartitioned.binary -t treeSet -f E
     -m GAMMA -n T4
```

The only difference between the `-f e` and `-f E` options is that with `-f E` the model parameters for each tree will be estimated completely from scratch each time, whereas under `-f e` they are just optimized on the first tree in the tree set. Then, `-f e` will, for all remaining trees only optimize their branch lengths. This is a trade-off between speed and accuracy, since `-f E` will run much longer than `-f e`.

Production Level Analyses

How many cores shall I use?

This generally depends on the type of data and model of rate heterogeneity you deploy as well as on the hardware you are using. As a rough guide, we propose the following.

DNA under GAMMA:Chose the number of cores such that each core will work on about 3,000-4,000 alignment patterns. Keep in mind that the number of distinct alignment patterns that is usually smaller than the number of alignment sites is reported to you by the parser component!

Protein data under GAMMA: Chose the number of cores such that each core will

work on about 1,000 alignment patterns.

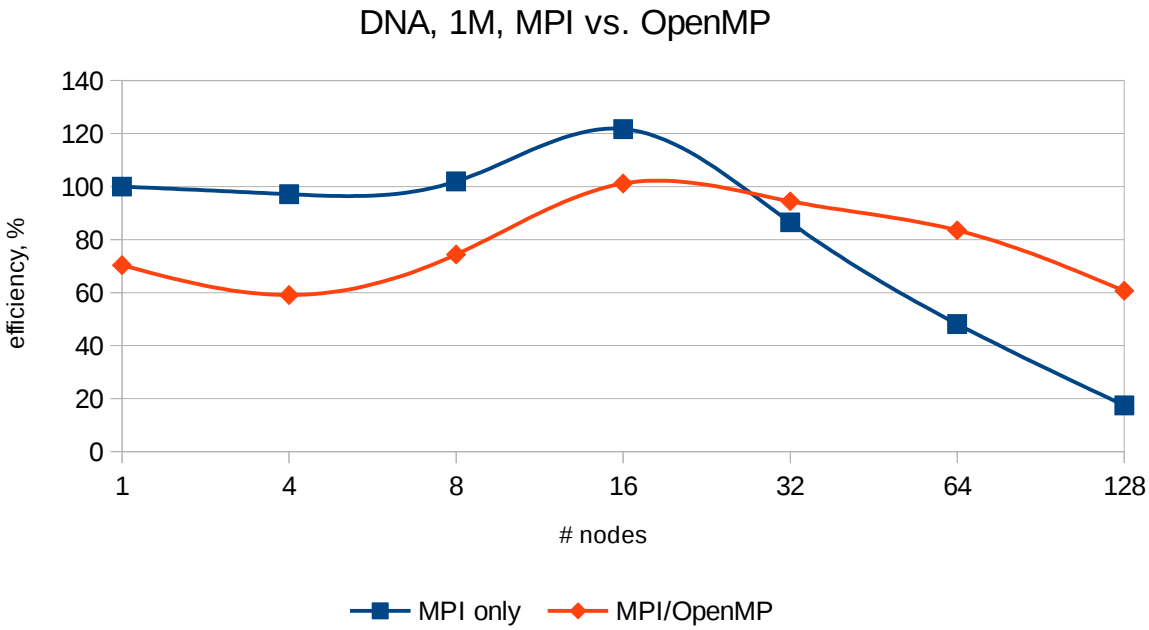
For analyses under CAT each core should work on around 12,000-16,000 patterns for DNA and 4,000 patterns for protein data.

When shall I use the hybrid MPI-OpenMP version on x86 systems?

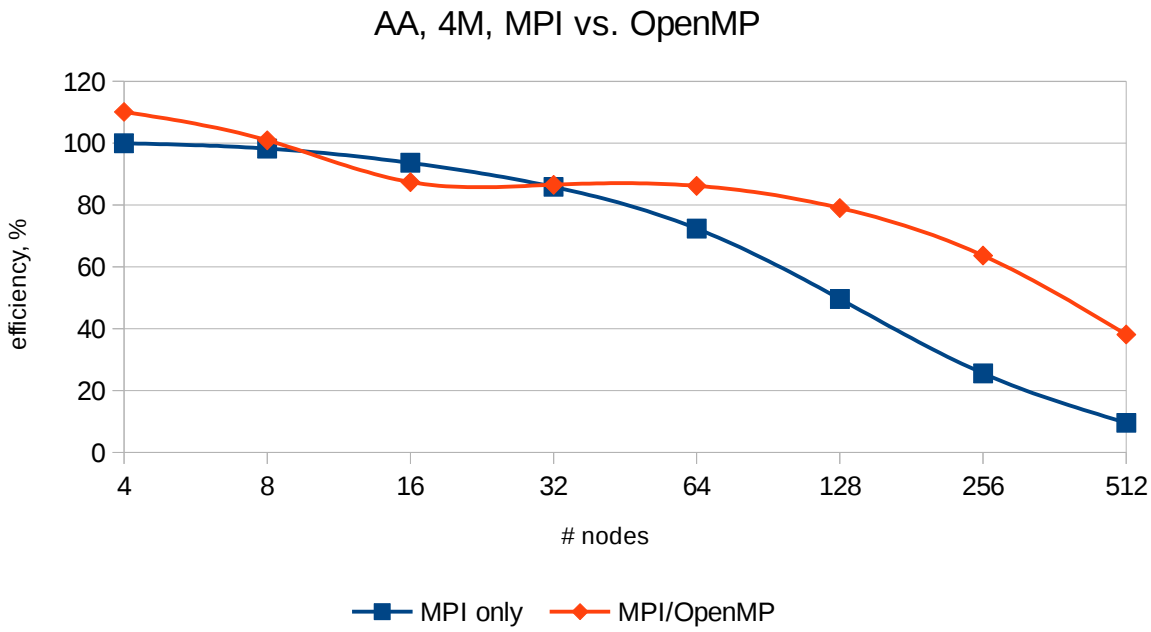
In general you should use the hybrid code when using a huge number of cores, since it will provide improved parallel efficiency. We have included two parallel efficiency plots below for a dataset with 1 million DNA patterns (Plot 1.) and 4 million amino acid patterns (Plot 2.).

It becomes pretty clear from these plots, that there is a clear cutoff point where the hybrid MPI-OpenMP version (red line) is more efficient than the pure MPI version (blue line). Also note that, for protein data the hybrid version is also slightly faster when only using 4 nodes.

Finally note that, the memory saving option (`-S`) is not available for the hybrid MPI-OpenMP version of ExaML.



Plot 1. Parallel efficiency on the SuperMuc supercomputer for 1 million DNA patterns over the number of nodes used (each node has 16 cores).



Plot 2. Parallel efficiency on the SuperMuc supercomputer for 4 million amino acid patterns over the number of nodes used (each node has 16 cores).

How do I conduct a full production-level analysis?

Here, we will quickly summarize our experiences and insights in running ExaML on the two large phylogenomic datasets (birds & insects) that were both published in *Science* in 2014.

Generally, broad phylogenomic datasets behave a bit differently than normal datasets in terms of the tree search, since there is just so much data. For the birds we had around 50 entire genomes, while for the insects we had about 140 transcriptomes.

Caution is advised with starting trees. Because there is so much data, using randomized stepwise addition, to generate a parsimony tree can frequently yield identical starting trees, despite different random addition orders.

Thus, once you have generated a set of 20-40 parsimony starting trees you should first check how diverse they really are. This can be done with the standard RAxML option `-f r` that can compute the Robinson-Foulds distances between all trees in a tree set and will also tell you how many distinct tree topologies are in that tree set.

Irrespective of this, you should also always generate 20-40 completely random starting trees (for instance, using standard RAxML), to make sure that the search algorithm is not biased in any way by the starting trees.

Before, running the complete set of ML searches we also advise that you should

check the behavior of the GAMMA and PSR models of rate heterogeneity. This can be done by running some preliminary analyses. We usually took a set of 10 starting trees (5 random trees and 5 parsimony starting trees) and conducted a total of 20 ML searches, 10 under PSR and 10 under GAMMA.

In a second step, we then calculated the scores of all 20 inferred trees under GAMMA with ExaML using `-f e` or `-f E`. This allows to determine if GAMMA- or PSR-based inferences yield the best results. Note that, caution is advised when using PSR on datasets with roughly less than 100 taxa, because in such cases there might not be enough data available to reliably infer the per-site rates.

With respect to bootstraps, we used the a posteriori Bootstopping criterion (see http://online.liebertpub.com/doi/abs/10.1089/cmb.2009.0179) implemented in standard RAxML to determine how many bootstrap replicates are sufficient.

This is done as follows: We use ExaML to compute an initial set of say 50 bootstrap trees. We then concatenate them into a single file and then use standard RAxML with the `-I autoMRE` option (for details, see RAxML manual) to determine if the support values have stabilized. If they have not, we run another, say, 50 BS replicates, concatenate all 100 BS trees into a single file and do an a posteriori bootstop test again. We repeat this procedure until we have enough replicates.

Note that, on such large phylogenomic datasets, due to the strong signal, doing 50 or 100 replicates is usually sufficient. Thus, in order not to waste valuable CPU time we highly recommend to use bootstopping.

Running ExaML on the Intel MIC

Since version 3.x ExaML also offers (partial) support for running on Intel MIC/Intel Xeon Phi-based clusters. In the following we will outline how to compile the code and what the limitations are.

Compiling

Please first set your MPI/MIC environment (ask your sys-admin if unsure) and then type:

```
make -f Makefile.AVX.gcc
make -f Makefile.MIC.icc clean
make -f Makefile.MIC.icc
```

This will create *two* executables for both, the host (=CPU) and MIC. They are called `examl-AVX` and `examl-MIC`, respectively.

Running on the MIC

Initially, use `parse-examl` to generate a binary alignment file as described before.

You might want to allocate MPI ranks both, on host CPUs and MICs (hybrid mode) or just on the MICs, depending on your configuration.

Here is an example command line for running ExaML in hybrid mode (16 CPU cores + 2 MIC cards):

```
mpiexec -host myhost-ib -n 16 /scratch/examl-AVX -n mictest
          -s /scratch/mictest.binary -t /scratch/start.tre
          -m GAMMA -w /scratch :
          -host myhost-mic0 -n 30 -env OMP_NUM_THREADS 4
          -env KMP_AFFINITY "granularity=fine,balanced"
          /scratch/examl-MIC -n mictest
          -s /scratch/mictest.binary -t /scratch/start.tre
          -m GAMMA -w /scratch :
          -host myhost-mic1 -n 30 -env OMP_NUM_THREADS 4
          -env KMP_AFFINITY "granularity=fine,balanced"
          /scratch/examl-MIC -n mictest
          -s /scratch/mictest.binary -t /scratch/start.tre
          -m GAMMA -w /scratch
```

Here, we use 1 MPI rank per core on the host CPUs. Also, on each MIC card, we start 30 ranks with 4 OpenMP threads each. This generates 120 threads in total or 2 threads per MIC core. For better readability we show the commands for starting ExaML on the CPUs in green, one the first MIC card in blue and on the second in red.

Changing the ratio of CPU versus MIC ranks allows to fine-tune the optimal load balance (and thus parallel efficiency) for the specific hardware configuration *and* dataset at hand.

The `KMP_AFFINITY` environment variable is specific to OpenMP and is used for mapping threads to processors. Please refer to the respective Intel documentation web-pages for further details.

Limitations

The following ExaML options *are supported* on the MIC:

- DNA and AA data
- GAMMA model of rate heterogeneity
- multiple partitions
- all AA substitution matrices supported by ExaML, including LG4M & LG4X

The following ExaML options are currently *not supported* on the MIC:

- binary data

- PSR model of rate heterogeneity (it does not fit the architecture well, a mapping would thus be inefficient and too complex)
- memory saving for gappy alignments (–s option)

Memory capacity:

Compared to traditional CPUs, MIC cards have significantly less memory available per core. This can cause problems for memory-intensive ML computations. Thus, you should plan your runs carefully (using the information generated by the parser) and split your run over multiple cards, if needed.

Performance:

ExaML-MIC performs best on alignments with a large number of sites and few taxa. The latter limitation is due to the limited on-card memory of the MICs (see above). Therefore, you might need to use multiple cards if the number of taxa is large.

For further details, please refer to:

http://www.hicomb.org/papers/HICOMB2014-04.pdf

and the on-line supplement of the ExaML version 3 *Bioinformatics* paper:

http://bioinformatics.oxfordjournals.org/content/suppl/2015/03/28/btv184.DC1/supplement.pdf

Frequently Asked Questions

Q: How can I do bootstraps with ExaML?
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 49 -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
```

Important: if you use a partitioned dataset, you will need to also pass the partition file of the original alignment to the above RAxML command via -q to generate correct BS replicate MSAs, but also, correct new partition files, (works as of RAxML version 8.2.5). The command line and output will look like this:

```
./raxmlHPC-AVX -# 100 -b 12345 -f j -m GTRCAT -s 49 -q part
                        -n REPS_2
```

This will produce the following output:

```
A partitioned model file with model assignments for bootstrap alignments
is printed to file /home/stamatak/Desktop/GIT/raxml-hpc/standard-RAxML/part.BS0
IMPORTANT: You MUST use this new model file and NOT the original one when running RAxML and ExaML on these bootstrapped alignments!

Printing replicate 0 to /home/stamatak/Desktop/GIT/raxml-hpc/standard-RAxML/49.BS0


A partitioned model file with model assignments for bootstrap alignments
is printed to file /home/stamatak/Desktop/GIT/raxml-hpc/standard-RAxML/part.BS1
IMPORTANT: You MUST use this new model file and NOT the original one when running RAxML and ExaML on these bootstrapped alignments!

Printing replicate 1 to /home/stamatak/Desktop/GIT/raxml-hpc/standard-RAxML/49.BS1

...

A partitioned model file with model assignments for bootstrap alignments
is printed to file /home/stamatak/Desktop/GIT/raxml-hpc/standard-RAxML/part.BS99
IMPORTANT: You MUST use this new model file and NOT the original one when running RAxML and ExaML on these bootstrapped alignments!

Printing replicate 99 to /home/stamatak/Desktop/GIT/raxml-hpc/standard-RAxML/49.BS99
```

Then, you would use standard RAxML again (and the corresponding partition file for the BS replicate!) 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
   $command = "./raxmlHPC-AVX -y -s ".$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, there is no PThreads support for parsimony calculations in RAxML, hence you should use the sequential version.

You will also need to run the parser again for each BS replicate MSA including the corresponding partition file for each BS replicate.

Once you have generated the starting trees and binary alignment files for each replicate you can then launch ExaML 100 times to compute the 100 Bootstrap trees.

Q: What is a large dataset that would be appropriate for ExaML?
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 or even full transcriptomes or genomes.

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

Q: What are the largest trees that can/have be computed with ExaML?
A: In terms of #taxa a tree with almost 120,000 taxa and a couple of genes was computed with the ExaML predecessor RAxML-Light. This can run nicely on a single multi-core node with 48 cores and 128GB of memory under the PSR model. In terms of broad phylogenomic datasets we have analyzed about 50 complete genomes (bird paper in *Science*) and 140 transcriptomes (insect paper in *Science*).