The ExaML v3.0.X Manual

December 12 2014

Alexandros Stamatakis
Heidelberg Institute for Theoretical Studies

Contents:

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. It 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 papers when using ExaML:

A. Stamatakis, A.J. Aberer: “Novel Parallelization Schemes for large-scale likelihood-based phylogenetic inference”. Pages 1195-1204 in proceedings of IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), Boston, MA, May 2013.
http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6569896&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6569896

T. Flouri, A. Stamatakis, K. Kobert, A.J. Aberer: "The divisible load balance problem and its application to phylogenetic inference". In Proceedings of WABI 2014, Wroclaw, Poland, September 2014.
http://link.springer.com/chapter/10.1007%2F978-3-662-44753-6_16#page-1

Similar to RaxML-Light whose development has been discontinued, ExaML also implements checkpointing, SSE3, AVX vectorization and memory saving techniques. Thus, the usage philosophy is similar.ML are similar to RAxML, thus it might be helpful to also read the RAxML manual.

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 2012; doi: 10.1093/bioinformatics/bts309. Link: 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`

A Usage Example

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

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

      -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 worrysome, 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
```

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. TODO more details!

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]
     [-h]
     [-i initialRearrangementSetting]
```

```
[-M]
[-S]
[-v]
[-w outputDirectory]
[--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.

`-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.`

`"-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.

`-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: Analogous to the corresponding standard RAxML option. 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.

```
-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.

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

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

   -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

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

      -v      Display version information

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

              DEFAULT: current directory

       --auto-prot=ml|bic|aic|aicc When using automatic protein model
selection you can chose the criterion for selecting these models.
              RAxML 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
```

To execute a simple run using two processors we type:

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

The output files and checkpoint files are analogous to those of RAxML-Light.

For a tree inference on a the partitioned dataset we type:

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

For better performance do not forget to experiment with the `-S` option if your dataset has a large fraction of missing data!