RAxML Scalability Strategy

HW Reqs

Initially a 16 or 32-core shared memory machine with 128GB or better 256GB of main memory for testing, exclusive access needed as well as roor rights.

Overall Strategy

Extract, separate, simplify, and re-design individual parts of RAxML to make them easier to tune, adapt, parallelize experiment with → get rid of a lot of the overhead of a real world tool that can do a lot of other things that are not required here.

Assumptions

I will assume that the input data will be a multi-gene datasets comprising only a few genes of DNA data. According to the Soltises this will most likely be the case, but let's wait for the data assembly workshop.

Technical Work for Right-Away

Somebody should look at the mechanisms used in the GIT version for synchronizing threads and doing reduction operations. It can be done in a more scalable way I am sure, but I don't have time for this now.

Strategy

I want to separate input data parsing, Maximum Parsimony Computations and Maximum Likelihood computations into three different components. This can be successively done by stripping down and simplifying the current RAxML code.

Data Parser (Phase I): read in a RAxML-readable DNA alignment and partition file and spit out a binary file formatted as indicated below.
The parser shall re-order and compress alignment patterns as in RAxML and check for taxon name duplicates etc.
Rationale: all this input data parsing can take pretty long (Amdahl's law) on large datasets and requires copying around alignment data between buffers → just do it once and then re-use the binary file, may be extended to parallel I/O in the future, as such one may think of separate data description files (#taxa,#sites, partitions, taxon names) and a separate binary file containing the data in compressed form. Thereby processes could access the largest bulk of data in a strided form.

General Format:

1. Number of taxa
2. Number of site patterns

3. number of partitions
4. partition ranges: start-end
5. weight vector for compressed data (as long as site patterns)
6. For every taxon: taxon name + sequences
   1. Data additionally compressed as follows: every site can basically have 15 states (ambiguous DNA coding) hence we need 4 bits per alignment pattern → can store 8 nucleotides in one 32-bit word
   2. For MP this format should also exclude all non-parsimony informative sites, in the current implementation I am not recovering all non-informative sites.

MP Component (Phase II): MP basically conducts bit-wise operations on trees: Strap down data structures and the way parsimony vectors are stored to the absolute minimum (remove all the ML stuff), represent parsimony vectors as compressed bit vectors (see above) and not as in the current version as one integer per alignment site. Do SSE3 operations on these vectors to more rapidly compute MP scores on trees. Once again we only assume that we have to deal with DNA data here. Keep Pthreads-type of parallelization and refine it by testing more adequate synchronization and reduction operations. Finally, compression of identical sites into site patterns with weigths > 1 may not be feasible if we go for the bit-wise MP operation approach.

ML Component (Phase II): Strap down RAxML as for MP, only assume we need DNA data and the CAT model or some modification thereof, initially keep Pthreads type of parallelization, only assume that there is one type of operation to be done: tree search on a given input tree. A simple organization of the code will allow for easy experiments with novel search algorithms. Add ability to write checkpoints in SW → provide a mechanism to read in model parameters in addition to the alignment. Also may just use standard RAxML to optimize model params once for a given, reasonable, tree and then just keep re-using them over all inferences, i.e., just have model parameter optimization overhead once.
If we just have one type of algorithm and one type of data it will be very easy to do checkpointing. Perhaps optimize the three most time-consuming functions in Assembler? How to achieve better parallel scalability with this is still an open question, algorithmic ideas are available and will be tested.