Poster C-34

Parallelizing Computationally Intensive Bootstrap Phylogenetic Analyses

Authors:
Ricardo Gonzalez-Mendez (*Department of Radiological Sciences, University of Puerto Rico School of Medicine*)
Hugh B. Nicholas (*NRBSC, Pittsburgh Supercomputing Center*)
Alexander J. Ropelewski (*NRBSC, Pittsburgh Supercomputing Center*)

Short Abstract: Computationally intensive programs in PHYLIP have been parallelized using MPI, enabling large-scale phylogenetic studies that rely on bootstrapping to be performed in a moderate amount of time. The methodology used to paralyze PHYLIP programs and the performance of the PHYLIP protein distance method on several parallel architectures is discussed.

Long Abstract:
Computationally intensive programs in the PHYLIP package have been parallelized using MPI, enabling large-scale phylogenetic studies that rely on bootstrapping to be performed in a moderate amount of time. This paper discusses the methodology used to paralyze the PHYLIP programs and reports the performance of the PHYLIP protein distance method on several unique parallel architectures.

In order to determine the reliability of a phylogeny and to make inferences about the evolutionary history derived from the tree the main methods are percent support for tree nodes using resampling methods, i.e. bootstrap percentages. These resampling procedures use substantial computer resources and require significant turn around time even for the more modest traditional datasets found in molecular phylogenetics. Parallel versions of the software used to carry out the resampling calculations become a critical need.

We have modified the PHYLIP package, a set of integrated programs that allows the construction of evolutionary trees from sets of homologous sequences. It includes distance matrix methods, parsimony methods, maximum-likelihood methods to search for phylogenetic trees using both heuristic and exact algorithms. It also allows for resampling techniques that include several types of bootstrapping, jackknifing, and permutation of characters. It also allows for consensus trees to be reconstructed from the resampled data analyses by use of strict consensus or several variants of the more usual majority rule consensus.

IMPLIMENTATION

We wanted to create a parallel implementation that:
• Required minimal changes to the PHYLIP source code, but still allowed a substantial speedup in the processing of large bootstrapped datasets.
• Made use of the MPI message-passing library which would enable the parallel implementation to work on a wide variety of computer architectures including Clusters, and both shared and distributed memory multiprocessors.

• Used a methodology could be transferred to other computationally intensive routines in the PHYLIP Package

To keep the code consistent with the serial version, we developed data primitives in MPI to distribute parameters and input file data to the processors. While the program is running, each node accesses its own input file and writes the data to its own output file. Prior to termination the output files are collected into a single output file. The general parallelism is as follows:
Distribute Stdin file to all processors
Distribute Input file to all processor
For (i = 1; i<number_of_datasets/number of processors; i++) {
Read unique dataset from local input file into global variables
Perform parsimony, distance, or maximum likelihood calculation
Write results to local output file}
Collect results from processors into single file
PERFORMANCE:

The PROTDIST code in the PHYLIP package was benchmarked using a 1000 replication dataset from a 32 species alignment 242 residues long on several different parallel architectures at the Pittsburgh Supercomputing Center. The first platform, Jonas is an SMP machine consisting of 64 1.15 Ghz EV7 processors with 256 Gbytes of shared memory with 6 Tbytes of local disk space. The second platform,. Lemieux, comprises 750 Compaq Alphaserver ES45 nodes. Each computational node contains four 1-GHz Alpha EV6.8CB processors with 4 Gbytes of memory. A Quadrics interconnection network connects the nodes. The timings and scaling for the dataset on these machines are shown in the table below.

Processors Jonas Elapsed Lemieux Elapsed Jonas Scaling Lemieux Scaling
1 (serial) 3505 4084 1 100% 1 100%
2 1768 2058 1.98 99% 1.98 99%
4 891 1035 3.93 98% 3.95 99%
8 453 535 7.74 97% 7.63 95%
16 234 292 14.98 94% 13.99 87%
32 140 177 25.04 78% 23.07 72%
64 n/a 137 n/a n/a 29.81 46%

The parallel performance scales well to a moderate number of processors. There are two factors that appear to affect the scaling of the code. The first factor is the problem size. The second factor is the disk IO performance of the system. Problem size affects scaling in three ways. First, the number of replicates selected directly affects scaling as the dataset size represents the total amount of work to be distributed among the processors. Second, datasets of sequences that are relatively long will have more work and require more time to compute than sequences that are relatively short. The third scaling factor is the number of sequences in the dataset. Larger sets of sequences will require relatively more computational time than smaller datasets. We have carried out a bootstrap calculation using 5,000 replicates in a data set of proteins that included 89 proteins and an alignment of 350 columns (including gaps), a run that took over 245 minutes using 16 AMD Opteron processors. The code will not scale well when processors are issuing more IO requests than

the IO system can handle. This effect was noted on both Jonas and Lemieux. On the 16 processor Jonas run it took approximately 15 elapsed seconds to read in and distribute the data to the node processors and to collect the output or about 6% of the total elapsed time. On the 32 processor Jonas run IO took approximately 28 seconds or about 20% of the total elapsed time. On Lemieux, the 16 processor run took 43 seconds or 15% of the total elapsed time, the 32 processor run took 48 seconds or 27% of the total elapsed time, and the 64 processor run took 73 seconds or approximately 52% of the total elapsed time for the IO processes.

In the future the use of computational grids and or agent-based implementations will likely be pursued. These may improve our ability to maximize use of computational resources or approach even larger problems or data sets. We are also interested in exploring the use of parallel versions of existing programs and combine those with our parallel resampling computations to gain further improvements in the use ML and Bayesian MCMC methods to study molecular evolution.