BIOINFORMATICS

Vol. 00 no. 00 2015
Pages 1–8

Supplement to "ExaML Version 3: A Tool for Phylogenomic Analyses on Supercomputers"

Alexey M. Kozlov [1,*], Andre J. Aberer [1] and Alexandros Stamatakis [1,2*]

[1]Scientific Computing Group, Heidelberg Institute for Theoretical Studies
[2]Institute for Theoretical Informatics, Department of Computer Science, Karlsruhe Institute of Technology

Associate Editor: Jonathan Wren

1 INTRODUCTION

In the following we provide a detailed technical description of the optimizations applied to ExaML version 3 in order to improve scalability and parallel efficiency. In Section 2 we briefly describe the new data distribution algorithm. A detailed performance analysis is available in (Kobert *et al.*, 2014). Then, we outline the novel parallelization approach for the alignment I/O during the `ExaML` start-up phase (Section 3). In Section 4 we outline the substantial re-design of the `ExaML` MIC version that was required to produce a usable, production-level implementation for this hardware platform. Finally, we investigate scaling performance of `ExaML` on both CPU and MIC clusters in Section 5.

1.1 Supported Platforms

ExaML runs on x86-based clusters with low-latency interconnects such as Infiniband. It can also be executed on Intel MIC-based clusters as well as on hybrid x86/MIC clusters. It supports Linux and MAC operating systems and can be compiled with `icc`, `gcc` and `clang`. ExaML has been successfully tested with the following MPI implementations: Intel MPI, IBM MPI (on the SuperMUC system), OpenMPI, and MVAPICH2. Note however, that we expect ExaML to run with all up-to-date MPI implementations, since it only uses basic MPI functions such as `MPI_Allreduce()` and `MPI_Scatterv()`.

ExaML does not provide support for systems based on IBM processors (e.g., IBM BlueGene/Q, etc.) yet, since this would require an appropriate and labor-intensive adaptation of the current SSE3/AVX core likelihood function vectorization. Moreover, systems based on IBM processors account for less than 20% of accumulated performance share in the November 2014 TOP500 list (see `http://www.top500.org/`). In addition, the vast majority of smaller-scale university clusters has x86 processors. Finally, we do not provide GPU support because of rather disappointing (less than a factor of two in the best case) performance with our kernel (Izquierdo-Carrasco *et al.*, 2013). Instead, we focus on the Intel MIC accelerator. Note that, the performance share of the TOP500 systems using the Intel MIC accelerators has increased from about 5% in the June 2014 list to approximately 30% in the November 2014 list.

*to whom correspondence should be addressed

2 NOVEL LOAD DISTRIBUTION ALGORITHM

Attaining an optimal data and/or load distribution represents a central challenge in high performance computing. Load balance algorithms strive to distribute computational load equally among processes such that unproductive overheads (waiting for other processors to finish their tasks) are minimized. In the phylogenetic setting, data and hence, load distribution is performed by assigning alignments patterns to processors, since we have to calculate the likelihood for each alignment pattern to compute the overall likelihood of a tree. Furthermore, each data partition for which a process evaluates alignment patterns requires certain pre-computations (e.g., eigenvector/eigenvalue decompositions and matrix exponentiations). Thus, if the alignment is partitioned (parts of the alignment such as genes, for instance, are assigned distinct models of evolution), load distribution in `ExaML` can be formulated as a bicriterion problem: (I) we have to distribute alignment patterns such that every processor has the same number of patterns (the difference between the minimum and maximum number of patterns assigned to processes should be ≤ 1) and (II) we have to assign partitions (partially) to processes, such that the maximum number of partitions (partially) assigned to a process is minimal. The latter restriction is due to the fact that each partition comes with a constant pre-computation overhead, regardless of how many alignment patterns it comprises.

Previous versions of `ExaML` offered cyclic data distribution, that distributed the patterns of each partition equally among processes (optimal w.r.t. criterion I). Alternatively, `ExaML` offered a scheduling algorithm that assigned full partitions to processors (Zhang and Stamatakis, 2012) (optimal w.r.t. criterion II). An optimal load distribution algorithm that satisfies both criteria is $\mathcal{NP}$-hard (Kobert *et al.*, 2014). Yet, there exists a linearithmic-time approximation that is guaranteed to miss the optimal solution w.r.t. criterion II by at most 1, while also guarantying the optimality w.r.t. criterion I (Kobert *et al.*, 2014). In other words, a *nearly equal* distribution of both partitions *and* alignment patterns among processes can be achieved.

We briefly outline the version of this algorithm that we implemented in `ExaML` version 3. Before starting the distribution algorithm, we calculate the desired (equal) load l per process (i.e., number of characters to be assigned to a process). If the number of processes is not a multiple of the number of characters, then some processes will acquire an additional character. However, we disregard this border case in the following description. Initially,

we sort partitions by their number of patterns in descending order. Then, we assign entire partitions to processes in a round-robin manner until for the first time we can not assign more partitions to a process, since the overall load of a process would exceed l. Having completed this phase, we are guaranteed that the number of partitions assigned to processes does not differ by more than 1 among processes. We now divide processes into a queue Q_{less} of processes with the minimum number of partitions and a queue Q_{more} of processes that already have one partition more (than the minimum) assigned to them. In the final phase, the algorithm partially assigns partitions to processes in a manner that satisfies both criteria. We continue the assignment (in descending order) with the partition that could not be fully assigned previously. In each iteration, we first try to dequeue a process from Q_{more} and check whether we can fill up its load via a partial assignment using the current partition. If this is not possible, we dequeue a process p from Q_{less} and try to fill up its load using as many characters of the current partition as we still have available. If p still has less than l characters, we requeue p into Q_{more} and finalize it in subsequent iterations. For a proof of correctness (w.r.t. satisfying criteria I and II) and a performance comparison to the data distribution strategies previously employed in `ExaML`, see (Kobert *et al.*, 2014).

3 STARTUP TIME OPTIMIZATION

Description. Because of Amdahl's law, sequential program phases can have a severe impact on the efficiency of parallel codes. For `ExaML`, the startup phase represents a performance bottleneck, since a high number of processes (potentially several thousands) have to read the input alignment file (potentially several GB) in the shortest amount of time possible.

`ExaML` provides a parser component that creates a pre-processed binary file. At the beginning of the startup phase (for an illustration see **Fig.** 1), processes first extract global information (e.g., taxon names, alignment lengths) and partition borders from this binary file. This information is needed to compute a partition to processor assignment (see **Sect.** 2). Given the partition assignment, each processor can allocate and extract alignment site patterns and associated site pattern weights (using `fseek`/`fread` operations). The `ExaML` parser writes the data of each partition in a taxon by taxon order to the binary file. Thus, if an entire partition is assigned to a single process, we only require a single `fread` operation to read this partition's data.

In previous versions of `ExaML`, each alignment row (each sequence) was read entirely by each process. Then, the base-pairs of the sequence the process would perform calculations upon were permanently stored in RAM. Alternatively, the alignment could also be read via a compressed stream using the `gzip` library. Thus, in previous versions, the per-processor memory requirements during the startup phase depended on the size of the alignment row. In contrast to this, in the novel implementation, each processor only requires memory proportional to the part of the alignment that is assigned to this processor. Note that for very large whole-genome alignments being analyzed with thousands of processors, the previous implementation that required reading-in and temporarily storing an entire sequence at each core led to per-core memory shortages.

Evaluation. We evaluated the new startup and I/O implementation using a simulated DNA alignment comprising 200 taxa and 100,000,000 bp (for information on availability, see Aberer *et al.* 2014). The binary alignment representation requires 19 GB of disk space, compressed versions of the alignment require between 3.4 and 3.8 GB depending on the partitioning scheme of the alignment. We measured the runtimes of the startup phase for an unpartitioned dataset and partitioned datasets with 500 and 1,000 partitions, respectively. Furthermore, runtimes were measured for all three discussed options (previous implementation, compressed implementation, new implementation) using 4, 8, 16, 24, 48, or 96 processes. We performed runtime measurements on 48-core compute nodes equipped with 4 AMD Opteron processors (each 12 cores). Thus, two compute nodes are required for the run with 96 processes. We averaged times over 4 measurements for each data point.

For our test dataset, we observe that, the novel `fseek+fread` implementation is at least one order of magnitude faster (see **Fig.** 2) than the previous variants. In the best case (500 partitions), the novel implementation invoked with 96 processes is 57$\times$ faster than the compressed variant of the previous implementation.

For previous implementations the total amount of data to be read increases with the number of processes and thus we observe a parallel slow-down. In contrast to this, the novel implementation scales well for partitioned alignments when the number of compute nodes is increased from 1 to 2. For instance, when using 48 processes on the dataset with 1,000 partitions the startup requires 33.6 seconds, while the startup takes only 18.7 seconds for 96 processes. Scaling properties are worse for the unpartitioned case, since partitions can not be read using only one `fread` operation. Unpartitioned datasets require a `fseek+fread` for each taxon.

Finally, we observe that reads of compressed streams generally improve start-up times. However, combining compressed streams with the `fseek` / `fread` approach introduced here is not straightforward.

4 OPTIMIZATIONS FOR THE INTEL MIC

Previously, we presented an initial, experimental version of the `ExaML` code with basic support for the Intel Xeon Phi coprocessors (Kozlov *et al.*, 2014). As of `ExaML` 3.0, this functionality has been extended and integrated into the production level version of the software. The current section provides details about implementation and performance of `ExaML` on the Intel MIC (called `ExaML-MIC` henceforth).

4.1 Implementation details

4.1.1 Supported functionality Currently, most of the core `ExaML` functionality is available on the Intel MIC, in particular:

- support for nucleotide (DNA) and protein (AA) data
- Γ model of rate heterogeneity
- partitioned (multi-gene) datasets
- all AA substitution matrices supported by the CPU version, including the LG4M and LG4X models

At the same time, the following features are not yet supported:

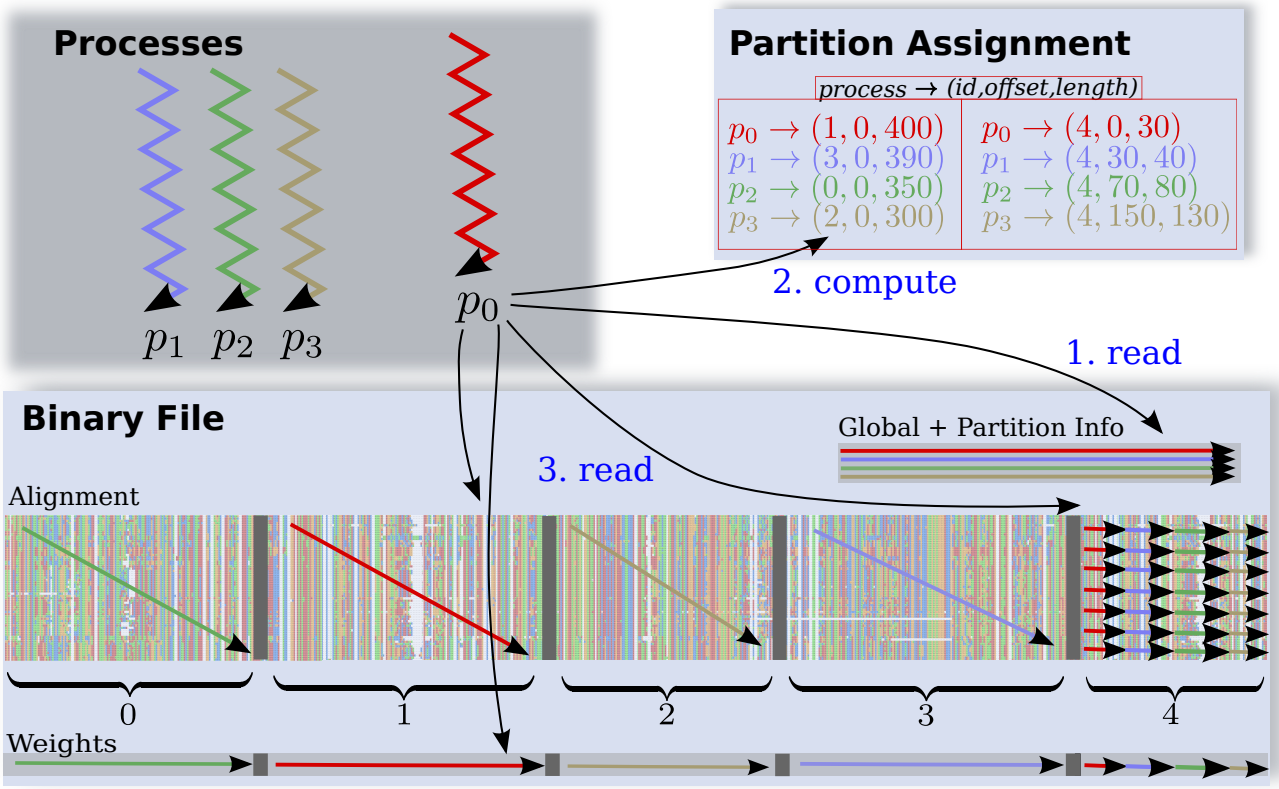


Fig. 1: `ExaML` startup phase (using 4 processes) illustrated for an alignment comprising 5 partitions. First, all processes read global information and partition definitions. Next, each process computes an assignment of partitions (or parts thereof) to processes. Subsequently, each process uses this information to read only data (i.e., alignment patterns and weights) assigned to it.

- support for binary characters (presence/absence data)
- PSR (per-site rate) model of rate heterogeneity (Stamatakis, 2006)
- memory saving by omitting calculations on missing data (`-S` option)

4.1.2 Likelihood kernel optimizations The evaluation of the Phylogenetic Likelihood Function (PLF) represents the main computational bottleneck for `ExaML` as well as for other likelihood-based (ML or Bayesian) tree inference codes. Therefore, optimizing the corresponding functions (called *PLF kernels* henceforth) is *the* primary goal for any efficient implementation and adaptation to a new hardware platform. In our previous work, we applied several MIC-specific optimization techniques to speedup the PLF kernels for DNA data (for a detailed discussion, see Kozlov *et al.*, 2014). In `ExaML` 3.0, we now also optimized the PLF kernels for protein data, using the same underlying principles.

4.1.3 New hybrid parallelization approach Originally, `ExaML` provided MPI-based parallelization only. So in order to exploit intra-node parallelism, multiple MPI processes had to be started (e.g., 1 process per CPU core). However, as our tests have shown, this approach does not fit the Xeon Phi well, since hundreds of MPI processes per card would need to be started. To circumvent this problem, we initially implemented an *ad hoc* OpenMP solution, where the main loop over alignment site patterns was parallelized in each kernel individually (Kozlov *et al.*, 2014). Despite being better than the pure MPI approach, the parallel efficiency of this initial solution deteriorates with an increasing number of partitions. Note that analyses with hundreds or thousands of partitions represent the standard `ExaML` use case. The are several reasons for the above inefficiency:

1. *Excessive amount of synchronization*: For practical reasons, each partition is processed via a separate kernel call. Consequently, for p partitions there will be p distinct `for`-loops over site patterns. Moreover, since every thread on the MIC is calculating per-site likelihoods for every partition, synchronization after each loop/partition is required.
2. *Sequential overhead*: As mentioned before, each partition requires some constant amount of computations (see Section 2). For instance, the P matrix exponentiation and the pre-computation of conditional likelihoods at the tips fall into this category. Since these computations are conducted outside the parallelized `for`-loops, they are executed sequentially and deteriorate performance according to Amdahl's law.

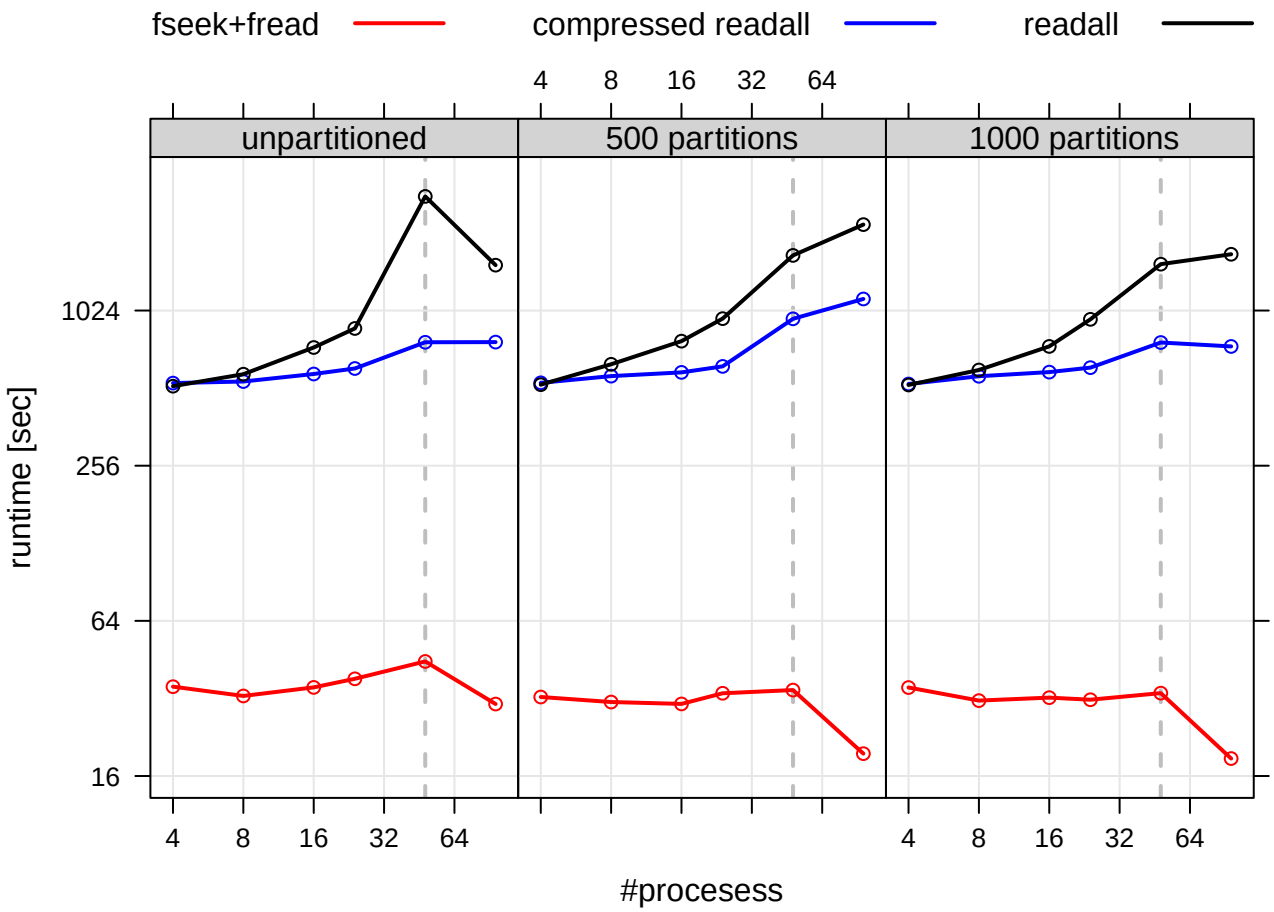


Fig. 2: Startup time evaluation on a DNA alignment comprising 200 taxa and 100,000,000 bp that is either (i) unpartitioned, (ii) divided into 500 partitions or (iii) divided into 1,000 partitions. Runtimes are depicted for a number of processes $\in [4, 96]$, where the instance with 96 processes is the only case, where processes reside on two distinct compute nodes connected by Infiniband. The red line represents the novel `fseek+fread` implementation in `ExaML` 3.0, the black line represents the previous approach of reading an entire alignment row at once and the blue line indicates the a variation of the previous implementation where `ExaML` reads a compressed alignment file.

3. *Reduced data locality*: Due to the OpenMP loop-based parallelization employed, there is no fixed assignment of alignment sites patterns to specific threads, so cache efficiency decreases because of lack of data locality.

Because of these shortcomings, we designed a novel OpenMP parallelization approach from scratch for `ExaML` 3.0 (see **Fig.** 3). In particular, we use the algorithm described in **Sect.** 2 again to distribute partitions and alignment site patterns not only among MPI processes, but also (in a second step) among OpenMP threads. In other words, we now perform two-level load balancing. Initially, partitions (or regions thereof) are assigned to MPI processes. Then, the partitions (or parts thereof) of a MPI process are assigned to individual threads within this process using the same algorithm once more. Thereby, we attain a fixed thread-to-alignment pattern assignment and improve data locality.

To deal with the remaining two issues, we introduce a two-phase PLF calculation:

- In phase 1, we parallelize over partitions: all partitions are distributed evenly among threads, and each thread performs the constant part of computational work (mentioned above and in Section 2) for the partition(s) assigned to it.
- In phase 2, we parallelize over sites: each thread performs PLF computations on its individual part of the alignment.

With this approach, we only require *two* synchronization barriers (one after *Phase 1* and one after *Phase 2*) to perform the PLF computation. Thus, the amount of barriers required is independent of the number of partitions. In addition, work is now evenly distributed among the MIC threads in both phases which eliminates the sequential bottleneck.

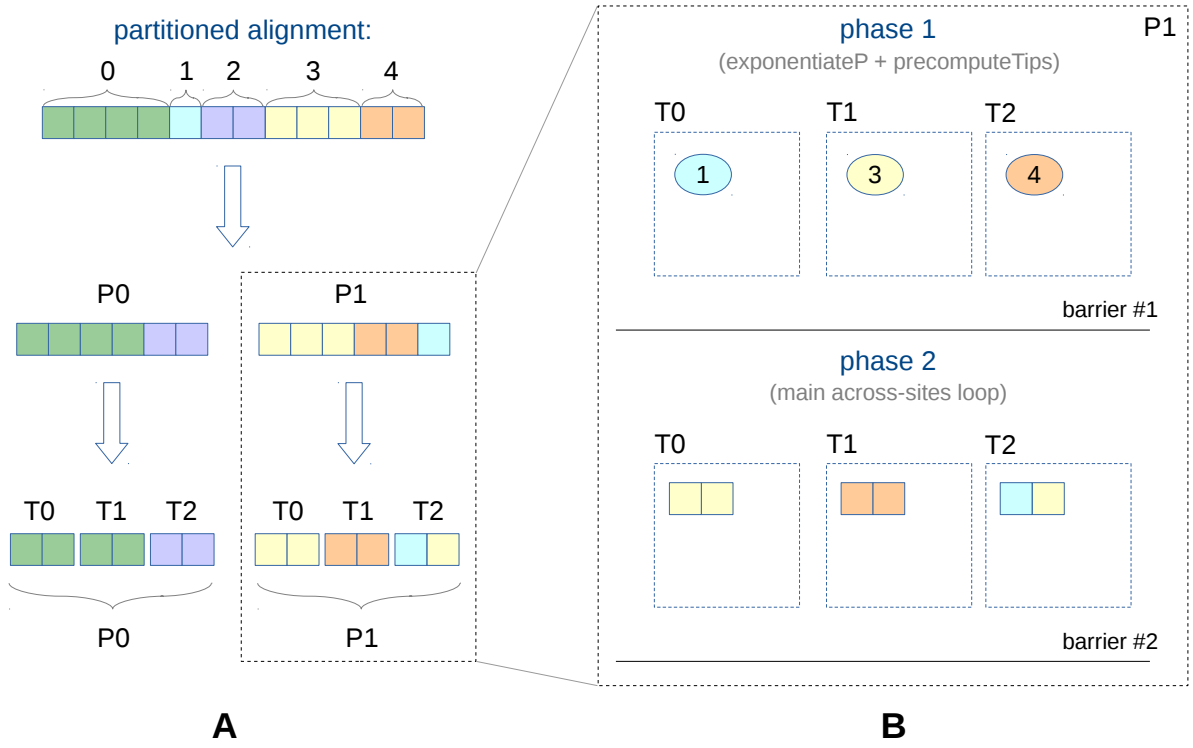


Fig. 3: Hybrid MPI/OpenMP parallelization scheme: **A**. Partitions are distributed among MPI processes, and then among OpenMP threads. **B**. The two-phase PLF evaluation procedure allows for better load balance and less synchronization overhead.

4.2 Performance evaluation

4.2.1 Experimental setup We used *Indelible v1.03* (Fletcher and Yang, 2009) to simulate 10 alignments (5 DNA, 5 AA) with representative dimensions as observed for empirical datasets (see **Table** 1 for details):

- `short`: single-gene alignment (e.g., classical 16S rRNA analysis),
- `medium`: multi-gene alignment with a moderate number of genes,
- `large`: whole-genome alignment or concatenation of hundreds of genes.

Due to the memory constraints on the MIC coprocessors, `large` alignments comprise only 20 taxa, whereas `medium` and `short` alignments consist of 200 and 2000 taxa, respectively.

For the `medium` and `large` alignments, we tested 2 partitioning schemes:

- unpartitioned (AA) or partitioned by codon positions (DNA),
- fine-grained partitioning schemes analogous to schemes generated by automatic partitioning tools (e.g., *PartitionFinder*, Lanfear *et al.*, 2012).

We performed all test runs on the SuperMIC cluster, which is part of the SuperMUC supercomputer (Leibniz Rechenzentrum, Garching, Germany). On SuperMIC, each compute node is equipped with 2 Ivy-Bridge host processors (Xeon E5-2650 v2, 2 $\times$ 8 cores @ 2.6 GHz) and 2 Intel MIC coprocessors (Xeon Phi 5110P, 2 $\times$ 60 cores @ 1.1 GHz). The nodes are connected via Mellanox Infiniband FDR14 using Mellanox OFED 2.2. At the time of performance testing, the software stack included:

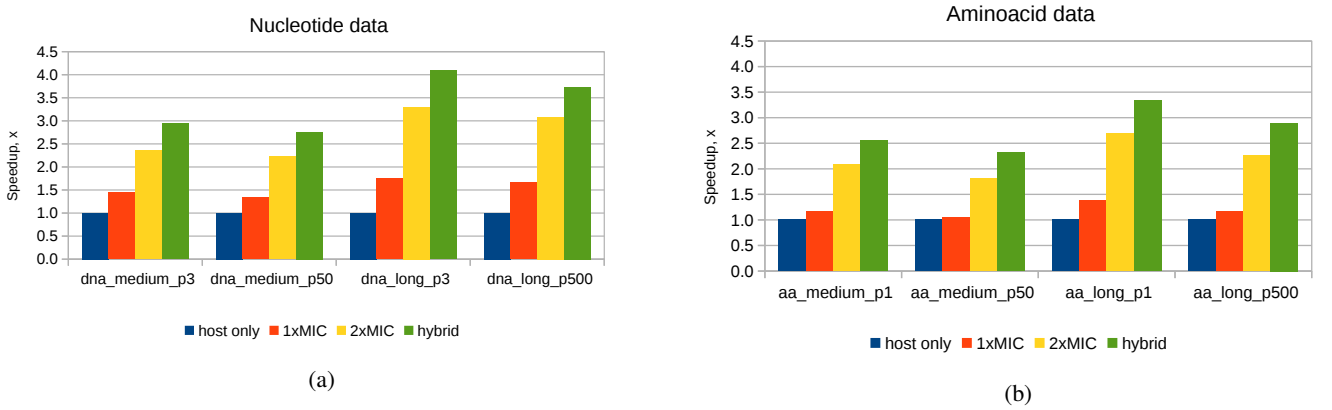
- MPSS 3.4
- MLNX_OFED_LINUX-2.2-1.0.1.1 (OFED-2.2-1.0.0)
- Intel MPI 5.0.1.035
- Intel Composer XE 2015.0.090
- Intel MKL 11.2
- Intel Amplifier XE 2015

4.2.2 Single-node performance To assess the performance on the Intel MIC, we measured `ExaML` runtimes under the following 4 configurations:

- `host`: 16 MPI ranks are placed on the host CPUs only (reference for speedup calculation),

Table 1. ExaML 3.0 execution times and speedups on the host CPUs (Host), on the single (1×MIC) and dual (2×MIC) Xeon Phi coprocessor(s), and in hybrid mode where CPUs and MICs are used simultaneously (Hybrid). Results for several nucleotide (DNA) and protein (AA) alignments with different dimensions and partitioning schemes are shown. The execution times are medians of 3 independent runs.

Dataset properties						Execution time, s				Speedup to host, ×		
Data type	Code	# taxa	# sites	# patterns	# part.	Host	1×MIC	2×MIC	Hybrid	1xMIC	2xMIC	Hybrid
DNA	`dna_short_1p`	2000	6k	5322	1	6776	29010	NA	NA	0.23	NA	NA
	`dna_medium_3p`	200	300k	293780	3	9579	6643	4073	3251	1.44	2.35	2.95
	`dna_medium_50p`	200	300k	293158	50	9505	7092	4264	3444	1.34	2.23	2.76
	`dna_long_3p`	20	4000k	1940554	3	1591	908	482	389	1.75	3.30	4.09
	`dna_long_500p`	20	4000k	3084131	500	2542	1528	825	680	1.66	3.08	3.74
AA	`aa_short_1p`	2000	2k	2044	1	41741	132019	NA	NA	0.32	NA	NA
	`aa_medium_1p`	200	60k	56655	1	18911	16304	9070	7423	1.16	2.08	2.55
	`aa_medium_50p`	200	60k	56966	50	22098	21169	12165	9534	1.04	1.82	2.32
	`aa_long_1p`	20	600k	442153	1	3402	2477	1265	1019	1.37	2.69	3.34
	`aa_long_500p`	20	600k	544580	500	4334	3711	1912	1497	1.17	2.27	2.90



(a) (b)

Fig. 4: ExaML speedups on the Intel MIC for DNA (*a*) and AA (*b*) alignments.

- `1xMIC`: 1 MPI rank on a single MIC card, 118 OpenMP threads (on `short` datasets, a smaller number of threads was used; see discussion below),
- `2xMIC`: 1 MPI rank with 118 OpenMP threads on each of the two MIC cards,
- `hybrid`: 16 MPI ranks on the host CPUs and 30 MPI ranks (× 4 threads) on each of the two MIC cards.

Please note that, we did not use the Hyper-Threading feature of the CPUs, since, according to our tests, running `ExaML` with two MPI ranks per physical core does not lead to any performance improvements.

The experimental results are summarized in **Table** 1, relative speedups are shown in **Fig.** 4.

As expected, the MICs perform better on longer alignments, where synchronization and sequential overhead are better amortized. On the other hand, dividing several thousand kilobases among >100 threads proved to be inefficient. This is in line with our empirical observations for the `ExaML` CPU version, where one process per 100–500 alignment site patterns should be used for good efficiency. Hence, we only used 30 and 40 threads for the `dna_short_1p` and `aa_short_1p` datasets, respectively. Analyzing such extremely short alignments on the MIC only makes sense when a 'multiplexing' strategy is applied. If multiple ML searches are executed (e.g., with different starting trees), one can start several (independent) `ExaML` instances in parallel, so that all MIC cores are being used (e.g., 4 independent instances with 30 threads each). In such a 'multiplexing' configuration, a single MIC card is on par with the performance of the host CPUs. Thus, more

than two-fold speedups are expected if both MIC cards *and* the host CPUs are used.

On the `medium` and `long` datasets, speedups on DNA data are better than on AA data (1.34×–1.66× vs. 1.04×–1.37×). Also, despite our dedicated optimizations for partitioned alignments, the MIC version shows a 5–20% performance decrease compared to the CPU version if the number of partitions is large.

In `hybrid` mode, up to four-fold speedups have been achieved on large DNA alignments. Since in this configuration the CPU and MIC cores work together, load balancing becomes essential. Given that ExaML distributes alignment patterns and partitions evenly among MPI ranks, one can fine-tune the ratio between MPI ranks on CPUs and MICs to improve load balance. For example, in our experiments we used 30 MIC ranks and 16 CPU ranks, which yields a ratio of 1.875. Intuitively, this ratio should be close to the MIC/CPU speedup. Thus, lower values might yield better results on smaller alignments (e.g., $24/16 = 1.5$).

5 SCALABILITY ANALYSIS

In scalability analyses, one usually distinguishes between *strong* and *weak* scaling. Both metrics assess the runtime improvement that can be achieved by using more CPUs for solving the problem in parallel. However, strong scaling is calculated based on a *fixed* problem (dataset) size, whereas for weak scaling the problem size is increased with the number of CPUs (i.e., the working set size is fixed *per CPU*).

More formally, strong and weak scaling efficiency are calculated as follows:

$$S_{strong} = \frac{t_{1,1}}{N * t_{1,N}} * 100\% \quad (1)$$

$$S_{weak} = \frac{t_{N,1}}{t_{N,N}} * 100\% \quad (2)$$

where N is the number of CPUs and $t_{1,1}$, $t_{1,N}$, $t_{N,1}$, $t_{N,N}$ are the runtimes for the respective dataset size (1 or N working units, first index) and the number of CPUs (second index).

5.1 Scalability of CPU version

We assessed the scalability of `ExaML` using three datasets from a recent phylogenomic study (Jarvis *et al.*, 2014). Their characteristics are summarized in **Table** 2. We performed our tests on SuperMUC compute nodes, that are equipped with two 8-core Xeon E5-2680 CPUs and 32GB of RAM each, and interconnected via Infiniband FDR10. We started 16 MPI processes per node, which corresponds to one process per physical core.

Due to the high memory requirements of the test datasets and limited per-node RAM capacity, it was impossible to obtain reference runtimes on just a single node. Therefore, we used the running time on the smallest possible number of nodes (4 nodes for `aa_4M` and `dna_20M`, and 64 nodes for `dna_320M`) as reference runtime for scaling efficiency calculations. Starting from this initial configuration, we increased the number of nodes step-by-step up to the point where no further runtime reduction was observed. We assessed scalability using the Γ model of rate heterogeneity, since it requires 4 times more computations than the PSR model. We also focus on the Γ model here, since it is widely used and broadly accepted.

Table 2. Characteristics of the datasets used for scalability tests.

Designator	Data type	# taxa	# alignment sites	# unique patterns	# partitions
`aa_4M`	AA	48	4,432,759	2,341,493	8000
`dna_20M`	DNA	48	19,258,311	15,424,402	500
`dna_320M`	DNA	51	322,150,876	161,845,822	1

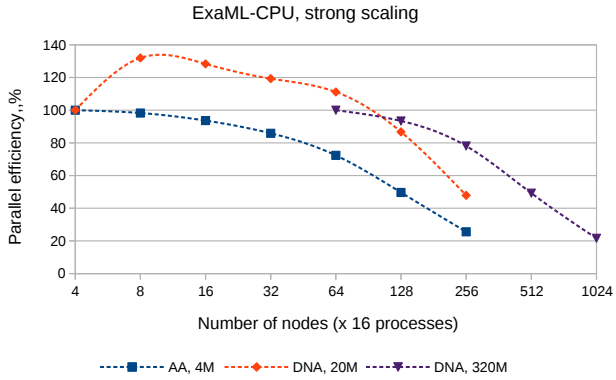


Fig. 5: Strong scaling of `ExaML` on the CPU-only nodes of the SuperMUC.

Strong scaling performance of `ExaML` is shown in **Fig.** 5. On the aminoacid dataset, parallel efficiency gradually decreases, and the maximum speedup of 16.36 can be achieved on 256 nodes (4096 cores). On the smaller DNA dataset (`dna_20M`), we measured a superlinear speedup (132%–111%) for 16 up to 64 nodes (256–1024 cores). This effect is in line with previous observations (Stamatakis *et al.*, 2012; Aberer *et al.*, 2014). It can be explained by better cache efficiency and/or increased memory bandwidth. The maximum speedup on `dna_20M` dataset is 30.6 and it was achieved on 256 nodes (4096 cores). The parallel efficiency is 48% in this configuration. Please note, that analyzing `dna_20M` on 4096 cores means that each core computes likelihood on less than 4,000 alignment patterns. Hence the high communication overhead cannot be sufficiently amortized. Finally, on the largest DNA dataset with more than 300 million alignment sites, `ExaML` scales up to 256 nodes with 78% efficiency, achieving a speedup of 3.12 compared to 64 nodes. Although the runtime can be further reduced by using 512 nodes, reaching a speedup of 3.94, the parallel efficiency drops to 49% in this case.

5.2 Scaling of the MIC version

On the Intel MIC accelerators, strong scaling of `ExaML` is limited by two factors:

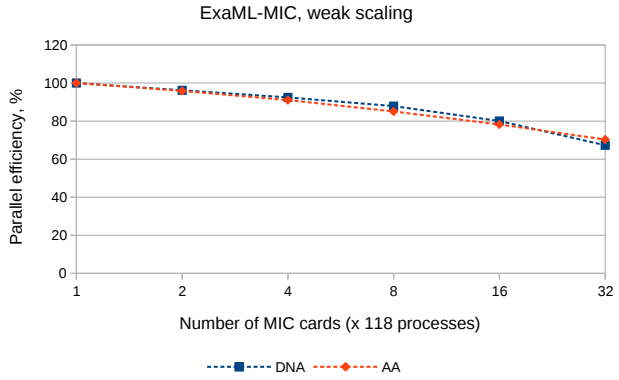


Fig. 6: Weak scaling of `ExaML-MIC`. Each MIC card has been assigned a part of an alignment comprising 50 taxa and 1000k DNA sites or 200k AA sites, divided into 100 partitions.

1. The maximum size of an alignment which can be analyzed on a *single* card is constrained by the amount of on-board memory (8GB for the Xeon Phi 5110).
2. As we distribute an alignment of *fixed* size over multiple co-processors, the number of site patterns per card and per core decreases, and so does efficiency.

In most practical cases, it will therefore be suboptimal to run an analysis on multiple cards, if one card can handle the data in terms of memory requirements. On the other hand, one might *have to* use multiple MICs if the dataset does not fit into the memory of a single card. To account for this scenario, we evaluated the *weak* scaling behavior of `ExaML-MIC` on large datasets.

As **Fig.** 6 shows, our implementation scales reasonably well: it attains a parallel efficiency of about 80% on 16 MIC cards and about 70% on 32 MIC cards. The difference in scalability between DNA and AA alignments amounts to only 1–3% and is therefore negligible.

REFERENCES

Aberer, A. J., Kobert, K., and Stamatakis, A. (2014). Exabayes: Massively parallel bayesian tree inference for the whole-genome era. *Molecular biology and evolution*, **31**(10), 2553–2556.

Fletcher, W. and Yang, Z. (2009). INDELible: a flexible simulator of biological sequence evolution. *Molecular biology and evolution*, **26**(8), 1879–1888.

Izquierdo-Carrasco, F., Alachiotis, N., Berger, S., Flouri, T., Pissis, S., and Stamatakis, A. (2013). A generic vectorization scheme and a gpu kernel for the phylogenetic likelihood library. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 530–538.

Jarvis, E. D., Mirarab, S., Aberer, A. J., Li, B., Houde, P., Li, C., Ho, S. Y., Faircloth, B. C., Nabholz, B., Howard, J. T., *et al.* (2014). Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, **346**(6215), 1320–1331.

Kobert, K., Flouri, T., Aberer, A. J., and Stamatakis, A. (2014). The divisible load balance problem and its application to phylogenetic inference. In *Algorithms in Bioinformatics*, pages 204–216. Springer Berlin Heidelberg.

Kozlov, A. M., Goll, C., and Stamatakis, A. (2014). Efficient Computation of the Phylogenetic Likelihood Function on the Intel MIC Architecture. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2014 IEEE 28th International*, pages 518–527. IEEE.

Lanfear, R., Calcott, B., Ho, S. Y. W., and Guindon, S. (2012). Partitionfinder: Combined selection of partitioning schemes and substitution models for phylogenetic analyses. *Molecular Biology and Evolution*, **29**(6), 1695–1701.

Stamatakis, A. (2006). Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective. In *Proc. of IPDPS2006*, HICOMB Workshop, Proceedings on CD, Rhodos, Greece.

Stamatakis, A., Aberer, A. J., Goll, C., Smith, S. A., Berger, S. A., and Izquierdo-Carrasco, F. (2012). Raxml-light: a tool for computing terabyte phylogenies. *Bioinformatics*, **28**(15), 2064–2066.

Zhang, J. and Stamatakis, A. (2012). The multi-processor scheduling problem in phylogenetics. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 691–698. IEEE.