2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops

Efficient Computation of the Phylogenetic Likelihood Function on the Intel MIC Architecture

Alexey M. Kozlov*, Christian Goll*, Alexandros Stamatakis*†
*The Exelixis Lab, Scientific Computing Group
Heidelberg Institute for Theoretical Studies
Heidelberg, Germany
Email: {alexey.kozlov, christian.goll, alexandros.stamatakis}@h-its.org
†Karlsruhe Institute of Technology, Institute for Theoretical Informatics,
Postfach 6980, 76128 Karlsruhe.

***Abstract*—Phylogenetic inference is the process of reconstructing the evolutionary history of species based on their traits, nowadays mostly using molecular sequence data. Current state-of-the-art inference methods, like Bayesian and Maximum Likelihood (ML) inference, rely on the Phylogenetic Likelihood Function (PLF) as their computational core. Due to the large number of floating-point operations involved, the PLF evaluation is *the* major bottleneck for large-scale phylogenetic analyses comprising thousands of genes or even whole genomes.**

Here, we describe an optimized implementation of the PLF kernel for the novel Intel Many Integrated Core (MIC) architecture. Using a MIC-based accelerator (Xeon Phi 5110P), we were able to achieve speedups ranging from 1.9x to 2.8x for different PLF kernels, compared to a highly optimized AVX implementation running on dual-socket Xeon E5-2680 system. By integrating the optimized PLF into the phylogenetic inference program RAxML-Light, we reduced the overall execution times by up to factor of two. To assess the scalability on multiple Xeon Phi cards, we also developed a hybrid MPI-OpenMP version of the ExaML code. When ExaML is executed on two coprocessors on the same node, we obtain speedups of up to a factor of 3.7 (vs. a CPU baseline) and 1.8 (vs. a single MIC). As expected, speedups increase with growing dataset size and become stable for alignments that require processing 1-2 million sites per MIC card.

***Keywords*-bioinformatics; phylogenetics; maximum likelihood; Intel MIC; parallel processing; MPI**

I. Introduction

Computational phylogenetics deals with the problem of reconstructing the evolutionary history of living species, or *taxa*. For each of the n species, a set of m traits is determined or obtained, that is used to assemble a $n \times m$ input matrix. The goal of a tree reconstruction (or *inference*) algorithm is to find a tree structure (*phylogenetic tree*), which best fits the data of the input matrix, given some pre-defined optimality criterion (model of evolution).

Nowadays, most phylogenetic analyses use molecular traits, that is, DNA, RNA, or protein sequences. In this case, the trait matrix is a multiple sequence alignment (MSA), where characters in the same column (also *site*, *base pair* or *bp*) share the same evolutionary history. The number of columns m in a MSA is often called *alignment width* and is one of the dimensions of the tree inference problem. The second dimension is the number of taxa n. The character-based phylogenetic inference methods on which we focus here, typically require $O(nm)$ operations to calculate the optimality score of a single, given tree.

Modern phylogenetic inference methods use probabilistic evolutionary models and rely on the Phylogenetic Likelihood Function (PLF, [1]) to score and chose among alternative tree topologies. In other words, they calculate the likelihood that the molecular data in the MSA were generated by the evolutionary process as described by a specific tree. Depending on the tree search or sampling strategy used, tree inference programs are classified into Maximum Likelihood (e.g., GARLI [2], PhyML [3], RAxML [4]) and Bayesian (e.g., MrBayes [5], PhyloBayes [6]) methods.

Due to frequent PLF calculations, such probabilistic tree inference methods require significant computational resources in terms of CPU time *and* memory. These computations become a limiting factor as datasets grow larger in terms of both, the number of taxa and alignment width. The former is driven by the ultimate goal to reconstruct a comprehensive Tree Of Life [7]. The latter is driven by advances in sequencing technologies, since substantially more molecular data can be obtained at a lower cost. In the past, it was a common practice to use short sequences comprising only several thousands of characters (e.g., the 16S rRNA gene, ∼1200 bp). Now, we are facing alignments with 1 up to 50 millions of sites, for instance in the 1KITE (1K Insect Transcriptome Evolution) project[1]in which we are directly involved. Inferring phylogenies on such datasets requires a tremendous amount of computational resources. Therefore, computer clusters or even supercomputers are required to accomplish this task. Yet, even with the most powerful hardware available to date, such analyses still require days or weeks. Thus, further algorithmic and hardware optimizations

[1]http://1kite.org/

978-1-4799-4116-2/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPSW.2014.198

IEEE computer society

are necessary to keep pace with the data deluge.

In the last years, using hybrid and heterogeneous architectures has become standard practice in high-performance computing (HPC), and has been largely dominated by NVIDIA GPUs. More recently, Intel introduced the Xeon Phi accelerator series that are based on the novel Many Integrated Core (MIC) architecture. As of November 2013, two out of the ten most powerful supercomputers in the world[2]already have Intel Xeon Phi coprocessors: *Tianhe-2* (#1, National Super Computer Center in Guangzhou, China) and *Stampede* (#7, TACC/Univ. of Texas, USA). Also, plans have been announced, to extend the *SuperMUC* (#10, Leibniz Rechenzentrum, Germany) we are typically using for large-scale analyses, by a MIC island. Therefore, it is important to adapt scientific codes to the MIC such that they run efficiently on this emerging accelerator.

The remainder of this paper is organized as follows: First, we survey related work on accelerating the PLF on different hardware/accelerator platforms in Section II. Then, we give an overview of the Intel Xeon Phi architecture and compare it to GPGPUs in Section III. In Section IV we identify the most compute-intensive routines involved in PLF computations, and provide details on their adaption and optimization for the MIC in Section V. Thereafter, we evaluate the performance of the kernels we developed and assess their impact on the overall runtime of ML tree inferences in Section VI. Finally, we conclude and discuss directions of future work in Section VII.

II. Related work

Substantial efforts have already been undertaken to port the PLF to various hardware accelerators.

Several approaches for implementing PLF kernels on reconfigurable logic (FPGAs) have been proposed by our lab [8], [9] and by others [10], [11]. The reported speedups vary substantially, ranging between 4x and 65x compared to the respective single-thread CPU versions. Unfortunately, these results are difficult to compare to each other due to different hardware and software that was used as a reference. Furthermore, all speedups were measured using kernel execution time only. Hence, additional overheads caused by PLF invocation latency and data transfer were not considered. Thus, the impact on overall runtime in real applications remains unclear.

Recently, a proof-of-concept GPU implementation of the PLF as implemented in our phylogenetic likelihood library (see http://www.libpll.org/) was presented [12].

An alternative PLF library is offered by the BEAGLE [13] framework. BEAGLE can be executed on multi-core systems using SSE3 vector intrinsics and OpenMP/OpenCL as well as on GPUs. Extensions of BEAGLE to support AVX intrinsics and the Intel Xeon Phi are planned (D. Ayres, pers. comm. December 2013).

We are not aware of any previously published implementations and adaptations of the PLF to the MIC, which is perhaps not surprising, given the novelty of the platform. Nevertheless, some promising results have been reported for other scientific applications. In particular, for fluid dynamics [14], N-body simulations ([15], and numerical weather prediction [16] speedups between 1.8x and 2.5x over a dual-slot (2S) Intel Xeon E5-2680 system have been obtained.

In Bioinformatics, the Xeon Phi has been successfully used to accelerate molecular dynamic simulations [17] and sequence motif extraction [18].

III. Intel MIC architecture

A. Overview

The first coprocessors based on the Intel Many Integrated Core (MIC) architecture became commercially available in late 2012 under the brand name Xeon Phi. Depending on the model, they include 57 to 61 physical cores running at ∼1GHz, which deliver a total of about 1 TFLOPS peak double-precision performance. Each core has a dedicated 512KB L2 cache, and can access the caches of all other cores via the ring interconnect. Additionally, each card is equipped with 6 to 16GB of GDDR5 memory that is shared among all cores. The instruction set was inherited from the x86 architecture and extended by 512-bit wide vector operations (AVX-512). This allows to process 8 DP (double precision) or 16 SP (single precision) floating point values simultaneously, that is, twice as much as with AVX/AVX2 on Sandy/Ivy Bridge CPUs.

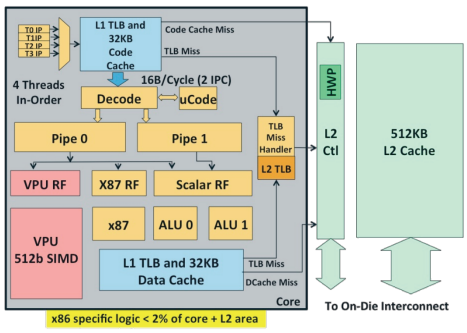

Figure 1: Architecture of the Intel Xeon Phi (codename: Knights Corner)[3]

B. Comparison with GPUs

Ths Xeon Phi is intended as direct competitor to GPUs. In fact, both accelerator families exhibit similar peak performance and power consumption (s. Table I), but differ significantly in their programming model.

[2]http://www.top500.org/lists/2013/11/

[3]Image source: http://semiaccurate.com/2012/08/28/intel-details-knights-corner-architecture-at-long-last/intel_xeon_phi_core/

First, the Xeon Phi has an order of magnitude less logical cores/threads (240 vs. >2000). However, this is compensated by the 512 bit wide vector unit, such that the overall degree of parallelism required to fully exploit hardware capabilities is roughly the same for both platforms.

Second, unlike NVIDIA GPUs, the Intel MIC does not offer direct control over the memory hierarchy. Instead, the programmer is expected to write cache-efficient code (e.g., preserve access locality) and rely on the hardware and/or compiler to decide, how to cache data. The only mechanism for influencing this process is to place explicit prefetch hints in the code.

Third, the Xeon Phi can be programmed using a standard x86 development toolchain (icc, pthreads, OpenMP), whereas NVIDIA relies on dedicated frameworks like CUDA or OpenCL.

Finally, the MIC offers two different program execution modes. The *offload* mode resembles the classical GPGPU approach: the main program runs on the host and 'offloads' compute-intensive code blocks (kernels) to the coprocessor. In contrast to this, in *native* mode, the entire program is executed exclusively on the coprocessor, without any host involvement. This is possible, because each MIC card executes a Linux-based micro-OS, thus, turning it into a self-contained 'host' system. Furthermore, the coprocessors obtain IP addresses and can communicate through the simulated network interface, allowing to use MPI for MIC-MIC and MIC-Host communication. Thereby, existing HPC codes can be easily ported to MIC-only or heterogeneous MIC-CPU clusters. We outline our initial experiences with this approach in Section V-D.

IV. Calculating and optimizing the likelihood of a tree

In RAxML [4] and derived programs, there are two main functions that are involved in likelihood calculations:

- `newview()` calculates a conditional probability array (CLA) v_p for a parent node in the tree, given the CLAs v_1 and v_2 for the two child nodes, and the corresponding transition probability matrices P_1 and P_2, which incorporate the branch lengths leading to these child nodes.
- `evaluate()` computes the overall log likelihood of the tree, given the CLAs of the two adjacent nodes and the branch length connecting them. We say that, we place a *virtual root* into this branch. The branch can be chosen arbitrarily, since under the general time-reversible model of substitution, which is used by most likelihood-based codes, the resulting likelihood is the same for all possible virtual root placements.

Additionally, in the Maximum Likelihood (ML) framework, explicit branch length optimization (with respect to the likelihood) is performed. Typically, a Newton-Raphson optimization method is used for this purpose. The most compute-intensive part, the calculation of PLF derivatives, is implemented in two RAxML routines:

- `derivativeSum()` performs a pre-computation of the element-wise product of CLAs of the two nodes adjacent to the branch under optimization. Since this part of the derivative calculation remains constant across all Newton-Raphson iterations, computations can be saved by storing and re-using these values.
- `derivativeCore()` computes the first and second derivative of the PLF with respect to the branch length being optimized.

In ML tree inference, these four functions (henceforth called *kernels*) account for more than 85% of overall execution time [19], and thus usually constitute the primary target for optimization.

V. Implementation

A. Scope

Here, we focus on the efficient adaptation of the four computational kernels outlined in the preceding Section to the MIC. To evaluate MIC performance for the PLF under realistic conditions, we integrated the kernels with two state-of-the-art tree inference programs developed in our lab: RAxML-Light [20] and ExaML (Exascale Maximum Likelihood [21]). Since this is exploratory work, our MIC-based version only supports a subset of their features at present:

- Only DNA data (4 states).
- Only one model of rate heterogeneity, the Γ model with four discrete rates [22].
- Although multiple data partitions are supported, we neither optimized nor evaluated tree inference on partitioned, multi-gene datasets. However, for a large number of partitions, performance will degrade due to decreasing parallel block size (less alignment sites evolving under the same statistical model of evolution) and growing communication overhead.
- Advanced memory saving techniques, which rely on CLA recomputations [23] and ignoring missing data [24], are not supported yet.

We plan to address these limitations in the future. Our focus here is to assess if the MIC is suited for accelerating PLF calculations, and how optimal performance can be achieved.

B. Porting and optimizing the kernels

In theory, getting an existing C code to run on the Intel MIC might be as simple as recompiling it with the `-mmic` compiler option. In practice, however, a program 'ported' in this way, will most probably exhibit insufficient performance. In fact, in some cases it might even execute slower than the original CPU version (see e.g., [14]). Hence, to fully exploit the hardware capabilities, one still needs to

invest a significant effort to optimize, adapt, and tune the code. In the following, we will describe several optimization techniques, and how we deployed them to improve PLF kernel efficiency on the Xeon Phi.

1) Vectorization: To attain optimal performance on the Phi, the code must be vectorized; there are several ways to achieve this. *Compiler intrinsics* are pseudo-functions which are usually mapped directly to the corresponding processor instructions, thus offering the highest degree of control. On the other hand, the Intel compiler offers automatic loop vectorization. Multiple loop iterations or arithmetic operations can be combined and replaced by a single vectorized instruction. For automatic vectorization to be successful, several conditions must hold, most importantly: the loop in question must be the innermost loop, all vectors must be properly aligned (see below), and there should be no data dependencies between input and output vectors. In difficult cases, the programmer can provide hints to the compiler using `ivdep` (no data dependency) and `vector aligned` pragmas. Compared to intrinsics, this approach produces more readable and less error-prone code (see Figure 2). With respect to the PLF, the compiler did not experience problems in vectorizing 'simple' loops (aligned vectors, uniform access pattern), hence we resorted to intrinsics in non-trivial cases only.

2) Memory alignment: Vectorized instructions can only operate on memory addresses which are aligned to 64-byte boundaries. This restriction has several implications. First, all arrays must start at addresses which are a multiple of 64. This can be achieved by using appropriate memory allocation functions (`__mm_malloc` or `memalign`). Second, one has to ensure that all accesses to the array elements are also aligned, that is, all offset accesses are aligned as well. In our case, this condition always holds. For DNA data evolving under the Γ model of rate heterogeneity the offset is 16 DP numbers or 128 bytes ($4\ states{*}4\ \Gamma\ rates{*}8\ bytes$). However, under the CAT model of rate heterogeneity [25] which only has one rate per site, special care must be taken to keep accesses aligned.

3) Re-organizing loops: As part of conditional likelihood computations in `newview()`, the CLA vector of a child node has to be multiplied with the transition matrix P. The dimension of this matrix is equal to the number of states (DNA characters in our case). For DNA data we therefore multiply a 1×4 vector with a 4×4 matrix. For this operation, the innermost loop executes 4 iterations, which is smaller than the vector unit width on the MIC (8 doubles). Therefore, the loop can not be vectorized efficiently without changes. Note that, under the Γ model with 4 discrete rates, we actually need to perform 4 such vector-matrix multiplications for each alignment site. If we execute these multiplications simultaneously, we obtain 16 iterations in the innermost loop, which is sufficient for vectorization. Since all iterations must access contiguous memory locations, we need to re-arrange the input arrays accordingly. Then, the inner loop can be calculated by two fused-multiply-add (FMA) vector operations.

4) Site blocking: Computing derivatives in `derivativeCore()` can be divided into two phases: for each alignment site, 16 elements of a vector are preprocessed, then several scalar operations are applied to obtain the final result. Obviously, the first phase can be easily vectorized, but the scalar operations pose a problem. To this end, we re-organized the main loop that iterates over single alignment sites to process alignment sites in groups of 8. This allows for executing one single vector operation that replaces the 8 problematic scalar operations.

5) Streaming stores: When writing into a memory cell, the old contents of the corresponding cache line have to be loaded first. If our intention, however, is to overwrite the entire cache line (64B), this reading operation is not required. The MIC introduces a special *streaming store* instruction, which allows to avoid this unnecessary read and associated time penalty. Even though the compiler implements some heuristics to automatically generate streaming store instructions, the programmer can enforce these by placing the `#pragma vector nontemporal` directive in front of the loop. We make use of this feature in `newview()` and `derivativeSum()` when writing results to the parent CLA and the summation buffer, respectively.

6) Manual prefetching: Prefetching allows for hiding memory access latency by predicting which data will be read in the future and preloading it into the cache *beforehand*. Thereby, the actual data access occurs directly from the low-latency cache without further delays. The MIC provides both, hardware and software prefetching mechanisms. The latter can be compiler-generated or inserted manually by the programmer (`#pragma prefetch` or `_mm_prefetch`). The programmer can fine-tune this, by determining the best *prefetch distance*, that is, for how many loop iterations ahead in the future, the prefetch instruction shall be issued. While the optimal value is mainly influenced by memory latency and the amount of computations per iteration, a direct estimation is complicated by other factors (e.g., number of threads per core). Thus, in practice one has to resort to an empirical tuning approach.

In a study led by Intel, different prefetching modes (HW vs. SW auto vs. SW manual) have been compared for a variety of applications. It concluded that automatic prefetching is nearly optimal for all but one application [26]. Nevertheless, we observed notable speedups by manually inserting prefetching instructions in our code. This can be explained by the streaming access patterns of our kernels. They linearly read input vectors from memory (summing buffers in `derivativeCore` and CLAs in other functions), perform relatively few computations, and then write results to the output vectors. In this setting, memory access latencies dominate runtimes and therefore an optimal

```
#pragma ivdep
#pragma vector aligned
for (int l = 0; l < 16; l++)
{
 sum[l] = left[l] * right[l];
}
```

(a)

```
__m512d l1 =
  _mm512_load_pd(&left[0]);
__m512d l2 =
  _mm512_load_pd(&left[8]);

__m512d r1 =
  _mm512_load_pd(&right[0]);
__m512d s1 =
  _mm512_mul_pd(l1, r1);

__m512d r2 =
  _mm512_load_pd(&right[8]);
__m512d s2 =
  _mm512_mul_pd(l2, r2);

_mm512_store_pd(&sum[0], s1);
_mm512_store_pd(&sum[8], s2);
```

(b)

```
vmovapd (%rsp,%r10,1), %k0, %zmm0

vmovapd 0x40(%rsp,%r10,1), %k0, %zmm1




vmulpd (%rcx,%rdi,8), %zmm0, %k0, %zmm2




vmulpd 0x40(%rcx,%rdi,8), %zmm1, %k0, %zmm3

vmovapd %zmm2, %k0, (%rsi,%rdi,8)
vmovapd %zmm3, %k0, 0x40(%rsi,%rdi,8)
```

(c)

Figure 2: Loop vectorized using pragmas *(a)* and compiler intrinsics *(b)*. In both cases, the generated assembly code is the same *(c)*.

prefetching strategy is of crucial importance.

C. RAxML-Light integration

RAxML-Light [20] already implements several PLF kernel variants (SSE3, AVX, different memory saving options) and provides a clear separation between the PLF routines and the rest of the code (tree search algorithm, file parsing, etc.). For this reason, it was relatively straightforward to integrate the MIC kernels into the program. However, selecting the appropriate MIC usage mode was more complicated.

Initially, we opted for the *offloading* approach: given a few, well-defined compute-intensive kernels, it seems natural to offload them to the Xeon Phi, and let the main tree search algorithm run on the host processor, invoking the kernels as needed. Costly host↔MIC data transfers can easily be avoided by allocating CLAs in coprocessor memory and only sending the node indices – a solution, which was used in our earlier GPU implementation [12]. Nevertheless, initial experiments with this offloading-based version showed that, even if no substantial data transfer is involved, the overhead of calling the offloaded kernel is still too high. In fact, it is comparable to and partially exceeds the time required for the actual computation. This seems to be an inherent limitation of the offloading approach: both hardware (initiating the data transfer over PCIe) and software (calling the offload runtime) components induce a certain latency [27], which can not easily be alleviated. Since ML inference algorithm perform thousands of kernel invocations per second, even for small trees, the offload latency becomes *the* major bottleneck.

For this reason, we then assessed the *native* execution model. After minor modifications, we were able to compile the entire RAxML-Light program on the MIC platform and execute it on the coprocessor without any host involvement. In this *native* version, the kernel invocations are simple function calls with negligible latency. Thus, we observed a speedup exceeding a factor of two compared to the initial offloading-based version. Moreover, the code became significantly simpler, because allocating and orchestrating separate CLAs on the coprocessor is not required any more. In fact, the only major differences between the CPU and the *native* MIC implementations are in the kernel codes.

Yet another benefit of this approach is the ability to re-use the existing PThreads-based parallelization of RAxML-Light [28]. The alignment sites are distributed evenly among the *worker* threads, each thread computes the likelihood for its own portion of the data, and finally a reduction is performed to obtain the overall likelihood score. Since the kernels are invoked by each *worker*, there is no need to introduce a thread-level parallelization in the kernel code.

D. ExaML integration

In the classical *fork-join* parallelization approach used in RAxML-Light, master and worker processes have to communicate at least twice per parallel region/kernel. If executed on multiple nodes, this communication occurs over the network, resulting in high latencies and performance loss. To alleviate this problem, a novel parallelization scheme was developed and implemented in our ExaML code: each process runs its own consistent (with all other processes) copy of the tree search algorithm, and they only communicate if information needs to be exchanged (e.g., to compute the overall likelihood after a call to `evaluate()`). This allows for avoiding redundant synchronization/communication between independent steps, for instance, between two subsequent calls to `newview()`. We have shown that, ExaML can be up to 3 times faster than RAxML-Light on a cluster systems [21]. Therefore, we decided to use ExaML to test the efficiency of parallel PLF calculations on multiple MIC cards.

Unlike RAxML-Light, ExaML has no multi-threading capabilities. Instead, MPI processes are used to exploit both intra- and inter-node parallelism, such that one MPI

process per core has to be started. On a typical multi-core system, this approach induces no significant performance penalty. In fact, our tests on a 16-core Xeon E5-2680 system showed, that the MPI-based ExaML code is even faster than PThreads-based RAxML-Light code on large datasets.

Unfortunately, the situation is different on the MIC. To fully utilize all 60 cores, a minimum of 120 threads must be executed in parallel. Hence, we need to execute 120 MPI processes with ExaML. An attempt to run ExaML in this configuration resulted in a substantial slowdown, suggesting the need for an alternative solution to exploit intra-MIC parallelism.

For this reason, we implemented a hybrid MPI-OpenMP approach by parallelizing the main loops over the alignment sites in each kernel using OpenMP. Thereby, we need to start only one MPI process per MIC card and can reduce the MPI communication overhead to an absolute minimum. Moreover, the ratio between the two parallelization levels can be changed by starting more MPI processes and less threads, respectively. Surprisingly, this strategy yielded better results in some tests, which can be explained by an improved trade-off between many inexpensive (OpenMP) and a few expensive (MPI) synchronizations.

VI. Performance evaluation

In the early days of GPGPU computing, it was common to report 100-fold speedups [29]–[31]. Despite being impressive, such results were often difficult to interpret because of the somewhat arbitrary and poor baseline reference. The comparison was usually made against a single-threaded, poorly optimized implementation running on an inexpensive desktop CPU. Recently, there has been a major shift towards using more adequate software *and* hardware for performance comparisons, resulting in more moderate speedups, rarely exceeding a factor of 5 (e.g., [15], [32], [33]). Since differences in execution times are less striking nowadays, practical considerations such as cost and energy efficiency become more important and are often included in the analysis.

In our experiments, we try to follow these guidelines to yield the performance comparison as fair as possible.

A. Experimental setup

1) Hardware: For our tests, we used two Xeon Phi 5110P cards installed on the same host system (Intel Xeon E5-2630, 6 cores, 2.30GHz, 32GB RAM).

We chose a dual-socket (2S) Intel Xeon E5-2680 system as a primary performance baseline. Its price and power budget are higher than those of the Xeon Phi 5110P by 30% and 15%, respectively. This makes the systems roughly comparable with respect to these parameters. Also, the 2S E5-2680 system is one of the most powerful CPU-based systems currently available on the market. For the same reasons, similar configurations have been used in other studies evaluating the Xeon Phi performance [14], [34], allowing to put our results into perspective. We also included a less expensive system based on the Xeon E5-2630 in our comparisons to assess the intra-CPU variability of our results.

The most important characteristics of all test systems are summarized in Table I.

2) Software: On CPUs, we used the AVX versions of RAxML-Light 1.1.1 (further denoted as **RAxML-CPU**) and ExaML 1.0.9 (**ExaML-CPU**), that are both freely available via GitHub[4]. On the coprocessor, we executed the same codes using the MIC-optimized PLF kernels, which we will refer to as **RAxML-MIC** and **ExaML-MIC**.

The code and test datasets used in this analysis are available for download at http://exelixis-lab.org/material/hicomb14mic.tar.gz. An up-to-date version of **ExaML-MIC** is also available via GitHub[5].

The respective versions of installed system software are provided in Table II. We had to use the Intel C compiler (`icc`) for the **ExaML-MIC** code due to lack of full MIC support in the current `gcc` version (4.7.3). For the **ExaML-CPU** we didn't observe any significant performance differences between `icc` and `gcc`. So we ran all experiments with `gcc`, since it is used in the production **ExaML** code. Both **ExaML-CPU** and **ExaML-MIC** were compiled with `-O2` flag, because `-O3` gave no measurable performance improvement, while being less stable.

Table II: Software configuration of test systems

Xeon E5-2630	Linux kernel 2.6.32 gcc 4.7.0 Intel MPI 4.1.2.040
Xeon E5-2680	Linux kernel 3.0.93 gcc 4.7.3 Intel MPI 4.1.1.036
Xeon Phi	Linux kernel 2.6.32 icc 13.1.3 Intel MPI 4.1.2.040

3) Datasets: We used INDELible V1.03 [35] to simulate 8 test alignments with sequence length varying between 10,000 and 4,000,000 DNA characters/sites. Since number of taxa has no influence on relative speedups, it is fixed and equals 15 for all datasets. Also, there is no need to include empirical datasets, because we are exclusively testing parallel performance.

B. Results

For each system configuration, we performed 3 independent runs, performing a full ML tree search and report the median execution times here.

[4]https://github.com/stamatak

[5]https://github.com/amkozlov/ExaML

Table I: Specifications of CPUs and accelerators used for performance evaluation

(Co-)processor	Peak DP GFLOPS	No. of cores	Core clock	Memory	Memory BW	Max TDP	Approx. price
2S Xeon E5-2630	220	12	2.30 GHz	32 GB	85.2 GB/s	190 W	$ 1224
2S Xeon E5-2680	346	16	2.70 GHz	32 GB	102.4 GB/s	260 W	$ 3486
1S Xeon Phi 5110P	1074	60	1.053 GHz	8 GB	320 GB/s	225 W	$ 2649
2S Xeon Phi 5110P	2148	120	1.053 GHz	16 GB	640 GB/s	450 W	$ 5298
NVIDIA K20 (ref.)	1170	2496	0.706 GHz	5 GB	208 GB/s	225 W	$ 2800

1S = single slot, 2S = dual slot; NVIDIA K20 listed for reference only

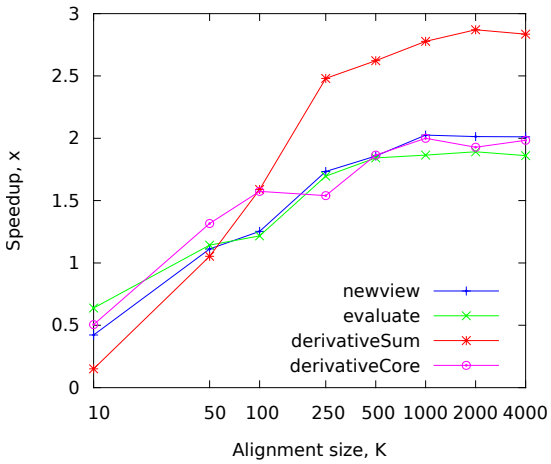

Figure 3: Speedups of the individual PLF kernels (relative to the CPU baseline).

1) Kernel performance: To assess the speedups of individual PLF kernels, we instrumented the **RAxML-CPU** *and* **RAxML-MIC** codes and measured the total time spent in the corresponding functions during one complete program run (tree search). As Figure 3 shows, the highest speedup (2.8x) was achieved for the `derivativeSum` kernel. This is, because `derivativeSum` performs a simple element-wise multiplication of vector entries (see Figure 2), which can be efficiently vectorized. The other kernels exhibit a less favorable mixture of numerical operations. Hence, the speedups for these kernels are at most a factor of two. These findings are in line with the results for our GPU implementation [12].

2) Tree search on a single MIC: Initially, we used the RAxML-Light versions, **RAxML-CPU** and **RAxML-MIC**, to assess the application-level performance on a single MIC card. This is because we assumed that, the PThreads-based parallelization scheme of RAxML-Light would be more efficient than the hybrid scheme employed by ExaML (MPI/OpenMP). However, this only holds for the smaller alignments. Furthermore, speedups for both implementation pairs (**RAxML-CPU** vs. **RAxML-MIC** and **ExaML-CPU** vs. **ExaML-MIC**) are almost identical. For the sake of clarity, we thus only report results for ExaML.

With **ExaML-CPU**, we used one MPI rank for each physical core available (i.e., 12 ranks on the E5-2630 and 16 ranks on the E5-2680). With **ExaML-MIC**, we tested different combinations and found that, 2 MPI ranks and 118 OpenMP threads per rank yield the best performance for almost all datasets. Thus, we always used this configuration in our experiments.

The results are summarized in Table III. On small alignments, **ExaML-CPU** is significantly faster than the MIC version. This is expected, because in **ExaML-MIC** each of the 236 OpenMP threads only processes a very small number of alignment sites that have been assigned to it. Therefore, the computational gain is not sufficient to outweigh memory access latencies and a high synchronization overhead relative to the amount of computation. For the 100K sites alignment, the execution times are almost identical, and on larger datasets **ExaML-MIC** shows superior performance. The difference increases rapidly with growing alignment size, and stabilizes at a factor of two for 1000K-4000K sites. This is nonetheless still significantly smaller than the theoretical hardware performance difference between the platforms ($\sim$3x in both peak GFLOPS and memory bandwidth, see Table I). However, this hypothetical maximum performance has almost never been achieved for real applications, with typical speedups ranging between 1.7x and 2.8x. [34].

Note that, the 4000K dataset already uses *all* available memory on our Xeon Phi card (8GB), suggesting that more on-card RAM would be desirable to attain improved performance on large-scale datasets. At present, a 16GB version of the coprocessor is available (Xeon Phi 7120P), and Intel announced plans to offer a user-extendable system memory in future MIC series. While this might imply giving up the high-throughput GDDR5, the overall effect for memory-intensive applications like ML tree inference will be positive.

3) Scaling to multiple MICs: To further evaluate scalability, we executed **ExaML-MIC** on two MIC cards residing on the same host. As in our previous experiments, we used 2 MPI ranks with 118 OpenMP threads *per card*. Execution times and speedups relative to the baseline CPU system are given in Table III, whereas Figure 4 shows runtime improvements compared to the single MIC card.

In general, we observed substantially better scaling on larger datasets for two reasons. First, in the dual-MIC scenario each card processes only one half of the entire

Table III: ExaML execution times and speedups on CPUs and MIC

System	Dataset size (# alignment patterns)															
	10K		50K		100K		250K		500K		1000K		2000K		4000K	
	Inference time (s) and speedup (x)															
2S Xeon E5-2630	5.6	0.73	32.4	0.74	93.5	0.72	183	0.81	372	0.84	753	0.84	1465	0.84	2965	0.84
2S Xeon E5-2680	**4.1**	**1.00**	**24.0**	**1.00**	66.9	1.00	148	1.00	312	1.00	633	1.00	1237	1.00	2494	1.00
1S Xeon Phi 5110P	12.9	0.32	29.7	0.81	65.6	1.02	101	1.47	176	1.77	328	1.93	619	2.00	1228	2.03
2S Xeon Phi 5110P	18.7	0.22	32.0	0.75	**54.4**	**1.23**	**72**	**2.06**	**122**	**2.56**	**203**	**3.12**	**354**	**3.49**	**667**	**3.74**

dataset, such that the effective alignment size is halved. This makes the comparison 'unfair', given that, performance is generally better on larger datasets (see above). Second, the overhead of intra-MIC communication is almost constant for all datasets, and hence contributes a larger fraction to overall execution time for short alignments.

Still, even on the largest dataset the speedup is 1.84x and thus suboptimal. This is most probably caused by a rather inefficient MPI communication over the PCIe bus. In particular, using the most recent version of the Intel MPI library (4.1.2.040, November 2013), we measured that the latency of an `AllReduce` is approximately 20 μs between two MIC cards compared to less than 5 μs between two cluster nodes connected via a QLogic InfiniBand interconnect. This is a major bottleneck, since ExaML communication pattern is dominated by `AllReduce` calls with very small packet size (usually just one or several doubles, for instance, to sum over partial tree likelihoods after `evaluate()`). On the other hand, latency has already improved substantially, compared to the previous Intel MPI releases ($\sim$35 μs with 4.0.3.008), and we expect it to further improve as the MIC software stack matures. Also, work is underway to introduce better MIC support in alternative MPI implementations. Recently, some optimizations for both, intra-node [36] as well as inter-node [37] communication in MIC clusters have been proposed for MVAPICH2.

4) Energy efficiency: We estimated the energy consumption for each system and program run using the following equation:

$$E\ [Wh] = MaxTDP\ [W] \times RunTime\ [s]\ /\ 3600,$$

where E is an estimate of the energy consumed, $MaxTDP$ is the thermal design power of CPU and the MIC, respectively, and $RunTime$ is the corresponding total execution time of the ML tree search. Then, we normalized the values to the energy consumed by the CPU baseline system to obtain relative energy savings (see Figure 5). The single MIC system becomes more energy-efficient on 100K alignments, and consumes up to 2.3x less energy on the largest datasets. Adding a second MIC card reduces the energy efficiency on all datasets. This is expected, given the suboptimal scalability of ExaML across two cards. Nevertheless, for alignments over 500K sites, the double-MIC configuration is still significantly more efficient than

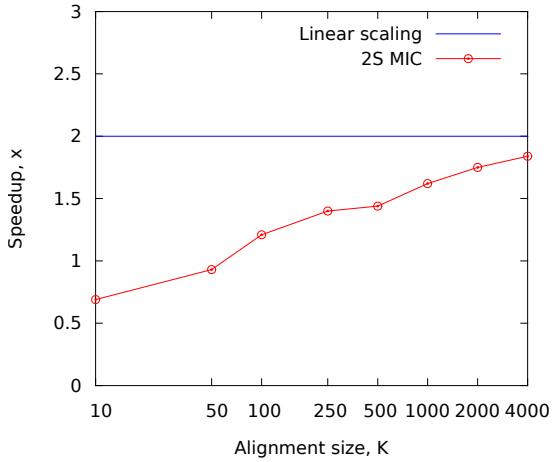

Figure 4: Relative speedup of 2 MICs vs. 1 MIC depending on alignment size.

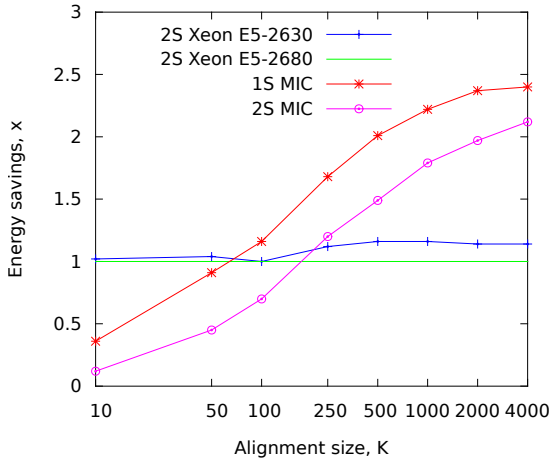

Figure 5: Relative energy savings compared to the CPU baseline.

both CPU systems, and also 3.7 times faster. Finally, we observe that the difference between the two CPU systems we tested is rather negligible (10–13%).

VII. Conclusion and Future Work

In this paper, we presented an optimized MIC implementation of the PLF kernels. We show that, for sufficiently large

datasets (>100,000 alignment sites) both, execution time, and energy efficiency of ML tree search algorithms can be improved compared to current CPU-based systems. Despite the fact that, the speedups are slightly suboptimal, they are in line with previous results for real-world applications on the MIC.

We also identify a relatively small per-card memory and a currently high MPI latency as main limitations of the current Xeon Phi platform with respect to PLF performance. We expect, however, that both shortcomings will be addressed by future hardware and software improvements.

We plan to extend our implementation to cover a broader subset of the ExaML functionality (e.g., support protein data and the CAT model of rate heterogeneity). Also, further performance optimizations and new load balancing strategies for partitioned alignments are required, given the popularity of such datasets and corresponding analyses in empirical biological studies.

Another direction of future work might be to extend this beyond the classical tree inference task and apply MIC kernels to accelerate other likelihood-based phylogenetic methods. In particular, the Evolutionary Placement Algorithm (EPA, [38]) appears to be a promising candidate, since different placement branches *and* query sequences can be evaluated independently, allowing for efficient parallelization with less communication overhead.

REFERENCES

[1] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *J. Mol. Evol.*, vol. 17, pp. 368–376, 1981.

[2] D. Zwickl, "Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion," Ph.D. dissertation, University of Texas at Austin, 2006.

[3] S. Guindon, J. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel, "New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0," *Systematic biology*, vol. 59, no. 3, pp. 307–321, 2010.

[4] A. Stamatakis, "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[5] F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Hhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck, "Mrbayes 3.2: Efficient bayesian phylogenetic inference and model choice across a large model space," *Systematic Biology*, 2012. [Online]. Available: http://sysbio.oxfordjournals.org/content/early/2012/03/02/sysbio.sys029.abstract

[6] N. Lartillot, T. Lepage, and S. Blanquart, "Phylobayes 3: a bayesian software package for phylogenetic reconstruction and molecular dating," *Bioinformatics*, vol. 25, no. 17, pp. 2286–2288, 2009. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/25/17/2286.abstract

[7] R. B. P. J. M. G. A. J. S. O. J. L. R. P. C. J. D. A. S. D. A. d. L. M. J. G. Hai Fang, Matt E. Oates, "A daily-updated tree of (sequenced) life as a reference for genome research," pp. –, 2013. [Online]. Available: http://www.nature.com/srep/2013/130618/srep02015/full/srep02015.html

[8] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis, "Exploring fpgas for accelerating the phylogenetic likelihood function," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–8.

[9] S. Berger, N. Alachiotis, and A. Stamatakis, "An optimized reconfigurable system for computing the phylogenetic likelihood function on dna data," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, 2012, pp. 352–359.

[10] S. Zierke and J. Bakos, "Fpga acceleration of the phylogenetic likelihood function for bayesian mcmc inference methods," *BMC Bioinformatics*, vol. 11, no. 1, p. 184, 2010. [Online]. Available: http://www.biomedcentral.com/1471-2105/11/184

[11] Z. Jin and J. Bakos, "Extending the beagle library to a multi-fpga platform," *BMC Bioinformatics*, vol. 14, no. 1, p. 25, 2013. [Online]. Available: http://www.biomedcentral.com/1471-2105/14/25

[12] F. Izquierdo-Carrasco, N. Alachiotis, S. Berger, T. Flouri, S. Pissis, and A. Stamatakis, "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*, 2013, pp. 530–538.

[13] D. L. Ayres, A. Darling, D. J. Zwickl, P. Beerli, M. T. Holder, P. O. Lewis, J. P. Huelsenbeck, F. Ronquist, D. L. Swofford, M. P. Cummings, A. Rambaut, and M. A. Suchard, "BEAGLE: An Application Programming Interface and High-Performance Computing Library for Statistical Phylogenetics," *Systematic Biology*, vol. 61, no. 1, pp. 170–173, 2012.

[14] C. Rosales, "Porting to the intel xeon phi: Opportunities and challenges," 2013.

[15] A. Vladimirov and V. Karpusenko, "Test-driving intel xeon phi coprocessors with a basic n-body simulation," 2013.

[16] J. Michalakes, "Code restructuring to improve performance in wrf model physics on intel xeon phi," in *Workshop on Programming weather, climate, and earth-system models on heterogeneous multi-core platforms*, 2013.

[17] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. Jarvis, "Exploring simd for molecular dynamics, using intelx00ae; xeonx00ae; processors and intelx00ae; xeon phi coprocessors," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1085–1097, 2013.

[18] S. P. Pissis, C. Goll, A. Stamatakis, and P. Pavlidis, "Accelerating string matching on MIC architecture for motif extraction," in *10th International Conference on Parallel Processing and Applied Mathematics*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.

[19] M. Ott, J. Zola, A. Stamatakis, and S. Aluru, "Large-scale maximum likelihood-based phylogenetic analysis on the ibm bluegene/l," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 4:1–4:11. [Online]. Available: http://doi.acm.org/10.1145/1362622.1362628

[20] A. Stamatakis, A. J. Aberer, C. Goll, S. A. Smith, S. A. Berger, and F. Izquierdo-Carrasco, "RAxML-Light: a tool for computing terabyte phylogenies," *Bioinformatics*, vol. 28, no. 15, pp. 2064–2066, 2012.

[21] A. Stamatakis and A. Aberer, "Novel parallelization schemes for large-scale likelihood-based phylogenetic inference," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013, pp. 1195–1204.

[22] Z. Yang, "Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites," *J. Mol. Evol.*, vol. 39, pp. 306–314, 1994.

[23] F. Izquierdo-Carrasco, J. Gagneur, and A. Stamatakis, "Trading running time for memory in phylogenetic likelihood computations," in *BIOINFORMATICS*, J. Schier, C. M. B. A. Correia, A. L. N. Fred, and H. Gamboa, Eds. SciTePress, 2012, pp. 86–95.

[24] F. Izquierdo-Carrasco, S. Smith, and A. Stamatakis, "Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees," *BMC bioinformatics*, vol. 12, no. 1, p. 470, 2011.

[25] A. Stamatakis, "Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective," in *Proc. of IPDPS2006*, ser. HICOMB Workshop, Proceedings on CD, Rhodos, Greece, April 2006.

[26] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin, and H. Saito, "Compiler-based data prefetching and streaming non-temporal store generation for the intel(r) xeon phi(tm) coprocessor," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 1575–1586.

[27] C. Newburn, R. Deodhar, S. Dmitriev, R. Murty, R. Narayanaswamy, J. Wiegert, F. Chinchilla, and R. McGuire, "Offload compiler runtime for the intel xeon phitm coprocessor," in *Supercomputing*, ser. Lecture Notes in Computer Science, J. Kunkel, T. Ludwig, and H. Meuer, Eds. Springer Berlin Heidelberg, 2013, vol. 7905, pp. 239–254. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38750-0_18

[28] A. Stamatakis and M. Ott, "Exploiting fine-grained parallelism in the phylogenetic likelihood function with mpi, pthreads, and openmp: A performance study," in *Proceedings of the Third IAPR International Conference on Pattern Recognition in Bioinformatics*, ser. PRIB '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 424–435. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88436-1_36

[29] J. Tolke and M. Krafczyk, "Teraflop computing on a desktop pc with gpus for 3d cfd," *Int. J. Comput. Fluid Dyn.*, vol. 22, no. 7, pp. 443–456, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1080/10618560802238275

[30] W. Xu and K. Mueller, "A performance-driven study of regularization methods for gpu-accelerated iterative ct," in *Workshop on High Performance Image Reconstruction (HPIR)*, 2009.

[31] M. A. Suchard and A. Rambaut, "Many-core algorithms for statistical phylogenetics," *Bioinformatics*, vol. 25, no. 11, pp. 1370–1376, 2009. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/25/11/1370.abstract

[32] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1816038.1816021

[33] E. Saule, K. Kaya, and U. V. Catalyurek, "Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi," *ArXiv e-prints*, Feb. 2013.

[34] Intel, "Intel xeon phi product family performance." [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf

[35] W. Fletcher and Z. Yang, "INDELible: a flexible simulator of biological sequence evolution." *Molecular biology and evolution*, vol. 26, no. 8, pp. 1879–1888, 2009.

[36] S. Potluri, A. Venkatesh, D. Bureddy, K. Kandalla, and D. Panda, "Efficient intra-node communication on intel-mic clusters," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013, pp. 128–135.

[37] K. Kandalla, A. Venkatesh, K. Hamidouche, S. Potluri, D. Bureddy, and D. Panda, "Designing optimized mpi broadcast and allreduce for many integrated core (mic) infiniband clusters," in *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*, 2013, pp. 63–70.

[38] S. A. Berger, D. Krompass, and A. Stamatakis, "Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood," *Systematic Biology*, vol. 60, no. 3, pp. 291–302, 2011. [Online]. Available: http://sysbio.oxfordjournals.org/content/60/3/291.abstract