2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

The Multi-Processor Scheduling Problem in Phylogenetics

Jiajie Zhang
Graduate School for Computing in Medicine and Life Sciences, University of Lübeck
Lübeck, Germany
bestzhangjiajie@gmail.com

Alexandros Stamatakis
The Exelixis Lab, Scientific Computing Group
Heidelberg Institute for Theoretical Studies
D-69118 Heidelberg, Germany
Alexandros.Stamatakis@h-its.org

***Abstract*—Advances in wet-lab sequencing techniques allow for sequencing between 100 genomes up to 1000 full transcriptomes of species whose evolutionary relationships shall be disentangled by means of phylogenetic analyses. Likelihood-based evolutionary models allow for partitioning such broad phylogenomic datasets, for instance into gene regions, for which likelihood model parameters (except for the tree itself) can be estimated independently. Present day phylogenomic datasets are typically split up into 1000-10,000 distinct partitions. While the likelihood on such datasets needs to be computed in parallel because of the high memory requirements, it has not yet been assessed how to optimally distribute partitions and/or alignment sites to processors, in particular when the number of cores is significantly smaller than the number of partitions. We find that, by distributing partitions (of varying lengths) monolithically to processors, the induced load distribution problem essentially corresponds to the well-known multiprocessor scheduling problem. By implementing the simple Longest Processing Time (LPT) heuristics in the PThreads and MPI version of RAxML-Light, we were able to accelerate run times by up to one order of magnitude. Other heuristics for multiprocessor scheduling such as improved MultiFit, improved Zero-One, or the Three Phase approach did not yield notable performance improvements.**

***Keywords*-scheduling; RAxML-Light; phylogenetics;**

I. Introduction

The on-going accumulation of molecular sequence data that is driven by novel wet-lab techniques poses new challenges regarding the design of programs for phylogenetic inference that rely on computing the Phylogenetic Likelihood Function (PLF [1]) for reconstructing evolutionary trees. In all popular Maximum Likelihood (ML) and Bayesian phylogenetic inference programs, the PLF dominates both, the overall execution time as well as the memory requirements by typically 85% - 95% [2]. The PLF is relatively straightforward to parallelize by exploiting the fine-grain loop level parallelism in the PLF using, for instance, OpenMP, PThreads, CUDA, OpenCL, and MPI [3] [4].

To accommodate the increasing dataset sizes we have recently developed a dedicated light-weight, production-level, and checkpointable PThreads and MPI version of the widely-used RAxML code for ML-based phylogenetic inference that is called RAxML-Light (available at https://github.com/stamatak/RAxML-Light-1.0.5). We are currently involved in a large sequencing and data analysis project that aims to reconstruct the phylogeny of 1000 insect transcriptomes (http://www.1kite.org). The preliminary scalability tests (using the MPI version of RAxML-Light) conducted for this project and the unprecedented data masses have given rise to the present work. Thus, program development and scalability is in a situation where it tries to catch up with the data.

While just a few years ago, phylogenomic datasets consisted of tens of genes, they now comprise hundreds or even thousands of genes. Thus, initially there were typically less partitions p than cores n available and measures needed to be taken to distribute the input alignment sites (regardless of the underlying model) in a cyclic round-robin fashion to obtain "good" load balance. This allowed for computing the likelihood of a tree with 10 partitions on a 48-core machine for instance. While the distribution strategy does not affect performance on unpartitioned datasets, it can, as we show, substantially influence performance on partitioned datasets. One of the main reasons for this is that, apart from computing the per-site log-likelihood scores, we also need to compute the transition probability matrices for each partition for each highly fine-grained parallel region of the code. Thus, when $p \gg n$, the ratio of the number of sites computed per calculation of the transition probability matrix (which is carried out locally on each core) becomes highly unfavorable. In other words, the local computations of the P matrix will dominate execution times. Hence, another data distribution strategy is required for phylogenomic datasets that minimizes the number of P matrix calculations per core and at the same time yields an approximately even distribution of alignment sites to all cores.

The remainder of this paper is organized as follows: In Section II we briefly discuss related work on algorithms for the multi-processor scheduling problem and on load balance in PLF computations. In Section III we describe the cyclic and monolithic data distribution approaches for computing the phylogenetic likelihood function in parallel. In the subsequent Section IV, we briefly describe the multi-processor scheduling algorithms we have tested and adapted. Thereafter, (Section V) we discuss the experimental setup and the results. We conclude in Section VI.

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.86

II. Related Work

A. Algorithms for Multi-Processor Scheduling

The partition distribution problem we face is essentially equivalent to the multiprocessor scheduling problem that falls into the category $P||C_{max}$ using the three-field classification scheme introduced by Graham *et al.* [5]. We want to assign p independent jobs to n identical cores, $p > n \geq 2$. Let C_i be the completion time of core M_i, then the goal is to minimize the maximum completion time (makespan) $C_{max} = max\{C_i\}$. The category of problems $P||C_{max}$ is NP-hard in the strong sense [6]. There exist exact as well as heuristic algorithms for $P||C_{max}$. Evidently, exact algorithms using branch-and-bound [7] or cutting plane [8] approaches are only applicable to problem instances with a small number of jobs and cores (roughly under 50 jobs and 15 cores [8]). To this end, we do not deploy exact algorithms because the number p of partitions (jobs) will typically be larger than 100. For instance, the human genome, is estimated to comprise roughly 30,000 genes, albeit there is a large variation in this gene number estimate.

Heuristic approaches to the multi-processor scheduling problem can roughly be classified into *constructive* heuristics and *improvement/refinement* heuristics. The LPT (longest processing time) algorithm [9] probably represents the best-known and most widely-used constructive heuristic algorithm. LPT is also implemented in the GIT version of RAxML-Light. Initially, LPT sorts all jobs (partitions) in descending order by their processing time (number of site patterns in the respective partitions). Thereafter, starting with the longest job (largest partition), all jobs (partitions) are successively assigned to the least loaded processor. In other words, a job is assigned to the processor that will complete its tasks earlier than all other processors. In phylogenetics, we keep track of the number of site patterns assigned to each processor and assign the next partition to the processor with the least accumulated number of site patterns. LPT has a worst case performance of $\frac{4}{3} - \frac{1}{3n}$, where n is the number of cores. This means that, in the worst case, the schedule computed by LPT will take $1.\bar{3}$ times longer to completion than the optimal solution. Simulations have shown that, LPT exhibits good average performance, in particular when p (the number of jobs/partitions) is large [10].

Numerous alternative constructive approaches have been proposed such as MultiFit [11] and, more recently, PSC [12].

Improvement/refinement heuristics that have been proposed include the 0/1 interchange [10] method, the 3-PHASE [13] heuristics as well as more complex approaches such as the cyclic exchange neighborhood [14] method and genetic algorithms [15].

B. Load Balance in the PLF

To the best of our knowledge, the present paper is the first to discuss this specific load distribution problem. This is because, RAxML-Light is, as far as we know, the only production-level MPI parallelization of the PLF that can handle datasets with RAM requirements of up to 1TB as well as full-genome alignments with up to 1000 species. Preliminary test with partitioned analyses on real data from full-genome and full-transcriptome sequencing projects have only now revealed this issue.

In previous work, we had focused on load balance issues for computing the PLF on partitioned datasets at a smaller scale [16]. In particular, we assessed the performance of simultaneously evaluating model and/or branch length parameter changes across all partitions and all cores. We showed that, proposing and evaluating changes simultaneously for all partitions can substantially improve parallel efficiency for partitioned analyses. This improvement was achieved by assigning larger chunks of work to each processor per broadcast/synchronization point in the code. At the same time, this also allowed for significantly reducing the number of synchronization points and/or barriers in the code. For details please refer to [16]. Note that, these experiments still relied on a cyclic distribution of per-partition sites to cores. Thus, the results of this previous work still hold and have in the meantime been integrated into RAxML-Light. What we report on here, is implemented on top of this previous work and deals with load balance at a more coarse-grained level.

Recently, Ayres *et al.* introduced a library implementation for computing the PLF [4] that can offload likelihood computations to multi-core processors using OpenMP or to GPUs via CUDA. The x86 implementation has also been optimized via SSE3 vector intrinsics. However, the BEAGLE library does not provide mechanisms yet for conducting partitioned analyses and does also not provide a distributed memory MPI implementation.

III. Cyclic versus Monolithic Data Distribution

As mentioned in the introduction, the computation of the per-partition probability transition matrix $P(t) = e^{Qt}$ represents the main cause of inefficiencies associated to computing the PLF on partitioned datasets with p partitions when a cyclic distribution of per-partition site patterns to cores is deployed.

To better explain this, consider the "classic" formula of the Felsenstein pruning algorithm [1] for recursively computing the conditional likelihood vector entries at a node k, given the two child nodes i and j. Given the probability vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$ of the child nodes, the respective branch lengths leading to the children b_i and b_j, and the transition probability matrices $P(b_i), P(b_j)$, the probability of observing an A at position c of the ancestral (parent) vector $\vec{L}_A^{(k)}(c)$ is computed as follows:

$$\vec{L}_A^{(k)}(c) = \Big(\sum_{S=A}^{T} P_{AS}(b_i) \vec{L}_S^{(i)}(c) \Big) \Big(\sum_{S=A}^{T} P_{AS}(b_j) \vec{L}_S^{(j)}(c) \Big) \quad (1)$$

This operation as well as structurally analogous arithmetic operations for computing the log likelihood at the root of the tree and for optimizing the branch lengths in RAxML (and other ML-based inference programs) largely dominate execution times (approximately 90%). A fine-grain parallelization of Equation 1 is straight-forward because the likelihood vector entries $\vec{L}^{(k)}(c), \vec{L}^{(k)}(c+1)$ for site patterns c and $c+1$ can be computed independently and thus simultaneously. Note that, $P(b_i)$ and $P(b_j)$ remain constant over all site patterns $c = 1...m$, where m is the number of site patterns in the partition that evolves according to the instantaneous nucleotide substitution model Q for time dt. To obtain $P(b_i)$ for instance, we need to exponentiate Q by computing $P(b_i) = e^{Qb_i}$ via a standard eigenvector/eigenvalue decomposition. As long as we do not change the values in Q, the eigenvectors and eigenvalues of Q will not change. However, as we traverse a tree for evaluating it, we still need to compute $P(b) = e^{Qb}$ for every branch *and* every partition of the dataset. Also note that, the matrix exponentiation must be conducted *prior* to computing the entries in $\vec{L}^{(k)}(c)$ for all c. The cost of computing $P(b)$ is relatively small compared to the operations in Equation 1 when the number of site patterns m is large. However, when m is small, that is, there are only a few site patterns that evolve under a model Q, matrix exponentiation can dominate run times.

Let us consider a simple example as outlined in Figure 1 with two cores c_0, c_1, and an alignment with two partitions p_0, p_1 and two site patterns per partition. If we use a cyclic distribution of per-partition site patterns to cores, c_0 will conduct likelihood operations on one site of partition p_0 and one site of partition p_1. The same holds true for c_1. Because the dataset is partitioned, p_0 will evolve under a model of nucleotide substitution Q_0 and p_1 under a model Q_1. Thus, if we use a cyclic per-partition site pattern distribution, each core will have to carry out a total of 4 matrix exponentiations for Q_0 and Q_1 to compute Equation 1. If we distribute the partitions monolithically to c_0 and c_1, then c_0 will only need to exponentiate Q_0 and c_1 only Q_1.

Note that, especially for the distributed memory parallelization of Equation 1, it is not desirable to have each MPI process compute some P matrices and subsequently gather them (as needed) via a MPI collective communication operation. By using a monolithic partition-to-process assignment, we can avoid deploying an additional collective communication (or a barrier in the PThreads code) altogether, between the matrix exponentiations and the likelihood computations in Equation 1.

Thus, if the number of partitions p is much larger than the number of cores n, our objective is to assign an approximately equal number of site patterns to each core *and* to distribute partitions monolithically to cores for amortizing the cost for Q matrix exponentiations. Ideally, we also want each core to conduct an equal number of matrix exponentiations, that is, we want to balance the number of sites *and* partitions per core.

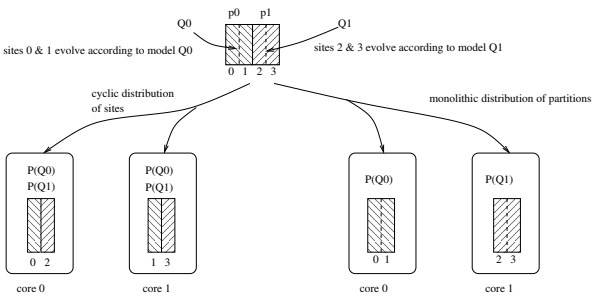

Figure 1. Example for cyclic and monolithic distribution of sites and partitions to cores for a simple alignment with four sites and two partitions.

IV. Partition Scheduling Algorithms

As already mentioned the current production-level implementation of RAxML-Light that is available via https://github.com/stamatak/RAxML-Light-1.0.5 implements the simple and fast LPT heuristics. The monolithic per-partition data distribution to cores/MPI processes can be invoked via the `-Q` command line flag. If `-Q` is not specified, the parallel PThreads and MPI versions of RAxML-Light will use the standard cyclic per-partition distribution of site patterns to cores.

In addition to LPT, we also implemented three slightly modified standard heuristics and a dedicated algorithm that also tries to balance the number of partitions per core (and not only the number of sites).

Henceforth, we consistently use phylogenetic terminology for describing the algorithms, that is, partitions instead of jobs and number of site patterns instead of job length.

A. Standard Heuristics

Improved 0/1 interchange (iZO): The 0/1 interchange algorithm by Fin and Horwowitz [10] comprises two steps: The *first step* initialize the n cores by randomly placing p partitions on them. In the *second step*, one partition will be moved from the most loaded core to the least loaded core if the makespan can be decreased. These interchanges are applied iteratively until makespan can not be further improved. The worst-case performance of this approach is $2 - \frac{2}{n+1}$. The improved 0/1 interchange heuristics [17] simply use LPT for the *first step* instead of random assignments. Hence, the makespan of this improved 0/1 interchange method can never exceed that of LPT. It can be demonstrated that, the improved 0/1 interchange has the same worst-case ratio as LPT.

Modified 3-PHASE (mTP): The 3-PHASE algorithm [13] —as the name suggests— consists of three phases: (i) the initialization phase, (ii) the job interchange

phase, and (iii) the job exchange phase. The job interchange phase is similar to the 0/1 interchanges algorithm. However, it uses the trivial lower makespan bound $C = \sum_{i=1}^{n} C_i/n$ for guiding job reassignments, which can potentially generate better results. Finally, the job exchange phase attempts to exchange jobs between pairs of cores to further reduce the makespan. To yield the two algorithms (3-PHASE versus iZO) more comparable, we adapted the initialization phase of 3-PHASE to also use LPT. Note that, these modified 3-PHASE heuristics can not perform worse than the improved 0/1 interchange.

Improved MultiFit (iMF): The MultiFit method [11] is inspired by a bin-packing method. It uses a bisection search for the minimal bin capacity C_{min} such that all p partitions can be fit into n bins (cores). Then, for each detected capacity, the First Fit Decreasing (FFD) [11] method is used to assign the partitions to the bins. As LPT, the FFD method requires all partitions to be sorted in descending order according to the number of site patterns. The main difference is that, FFD will then (after sorting) subsequently assign the partitions to the core with the lowest index that can complete the job within the capacity C_{min}, instead of assigning the partition to the core with the least number of site patterns (LPT). This difference in the assignment strategy can generate a large variation in the number of partitions that are assigned to each core. The bisection search will search C_{min} between the upper bound $max(P_{max}, 2C)$ and the lower bound $max(P_{max}, C)$, where P_{max} is the largest partition, and $C = \sum_{i=1}^{n} C_i/n$. MultiFit has a worst-case performance of $\frac{11}{9} + 2^{-k}$ where k is the number of executed bisection searches. Improved MultiFit (iMF) [18] deploys tighter bounds to improve the efficiency of the algorithm. iMF first runs LPT to determine the makespan M. If $M \geq 1.5C$ then it stops, otherwise it will execute standard MultiFit with an initial upper bound of M and a lower bound of $max(\frac{M}{\frac{4}{3}-\frac{1}{3n}}, P_{max}, C)$.

B. Dedicated Heuristics

The algorithms described so far are designed to minimize the makespan, as given by the number of accumulated site patterns per core. Since, on top of this, we desire each core to also be assigned an equal number of monolithic partitions (see Section III), we developed additional heuristics of our own. We introduce the vIC (interchange of jobs to minimize job number variance) method that strives to decrease the variance in the number of partitions per core, and at the same time strives not to increase the makespan.

The vIC method consists of three steps:

Step 1: Execute one of the standard multiprocessor algorithms to obtain an initial partition-to-core assignment and calculate the makespan M.

Step 2: Find the core M_{max} that has the largest number of partitions with the number of accumulated site patterns S_{max}. Also find the core M_{min} which has the smallest number of partitions and the corresponding number of accumulated site patterns S_{min}. Note that, M_{min} and M_{max} do not need to have been assigned the largest and smallest number of site patterns. Scan all partitions P_i with S_i sites that have been assigned to M_{max}. If $S_{min} + S_i \leq M$, then assign P_i to M_{min}. Repeat step 2 until no further improvement can be made.

Step 3: Find the core M_{max} which has the largest number of partitions and S_{max} accumulated site patterns. For all the other cores M_j has S_j sites, scan all partitions P_i with S_i sites on M_{max}, if $S_j + S_i \leq M$, then reassign P_i to M_j and go to step 2; else the algorithm terminate.

V. Experimental Setup & Results

A. Experimental Setup

We generated simulated alignments to assess performance of the new data distribution scheme and the heuristics described in Section IV. To emulate a realistic partition length distribution, we extracted the gene lengths from the human protein reference sequence database (ftp://ftp.ncbi.nih.gov/refseq/H_sapiens/mRNA_Prot/). The distribution of human protein sequences lengths (corresponding to partition lengths in ours experiments) is provided in Figure 2.

We applied INDELible [19] to four distinct tree topologies with 10 taxa each to simulate two protein and two DNA datasets. The properties of the simulated datasets are summarized in Table I. The test datasets together with the source code that implements various scheduling algorithms in RAxML-Light can be downloaded at www.exelixis-lab.org/Scheduling_online_material.zip. Note that, for our performance assessment it does not matter if we use simulated or real data, as long as the partition length distribution is realistic.

Table I
Test data set sizes

Test data set	100 partitions	200 partitions	500 partitions	800 partitions	1000 partitions
Protein 1	58032aa	106984aa	273578aa	445940aa	570970aa
Protein 2	50621aa	110827aa	290336aa	437574aa	521612aa
DNA 1	146875nt	336755nt	882494nt	1342040nt	1744252nt
DNA 2	153802nt	333170nt	876077nt	1288862nt	1646677nt

Tests were run on an Infiniband-connected cluster at the Heidelberg Institute for Theoretical Studies that is equipped with 50 48-core AMD Magny-Cours nodes and 128GB of RAM each. We executed runs with the PThreads and MPI-based version of RAxML-Light on 24, 48, and 96 cores.

B. Experimental Results

The evaluation of the alternative scheduling algorithms was based on two values: makespan (maximum accumulated number of sites assigned to a core) and the variance of the number of partitions among cores. Both values should ideally be minimized. We measured RAxML-Light execution times for the standard implementation with a cyclic data

Table II

MAKESPAN (M), VARIANCE (VAR), VARIANCE OF VIC (VAR-VIC), EXECUTION TIMES UNDER GAMMA (T-G) AND EXECUTION TIMES UNDER CAT (T-C) IN SECONDS FOR 4 SCHEDULING ALGORITHMS AND THE STANDARD CYCLIC DATA DISTRIBUTION ON 24, 48, AND 96 CORES. THE VAR COLUMN SHOWS THE JOB NUMBER VARIANCE OF THE SCHEDULING ALGORITHMS THAT DO NOT USE VIC IMPROVEMENT. THE NEXT COLUMN (VAR-VIC) SHOWS THE VARIANCE IMPROVEMENT (IF ANY) OBTAINED BY APPLYING VIC. THE NUMBER OF PARTITIONS VARIES FROM 100 TO 1000. THE RUNS ON 24 AND 48 CORES WERE EXECUTED USING THE PTHREADS VERSION, RUNS ON 96 CORES RUNS WERE TESTED USING THE MPI VERSION ON DATA SETS WITH 1000 PARTITIONS ONLY.

Protein 1		LPT / iZO / mTP					iMF					Standard implementation	
n	p	M	Var	Var-vIC	t-G	t-C	M	Var	Var-vIC	t-G	t-C	t-G	t-C
24	100	**2268**	1.472	0.888	134	41	2268	13.97	0.888	144	43	179	303
	200	2931	0.638	0.555	218	76	2907	25.80		218	100	349	616
	500	7441	0.388		549	190	7430	197.2		554	270	881	1497
	800	12046	0.638	0.305	893	309	12015	586.7	570.6	903	435	1434	2393
	1000	15125	11.13		1113	402	15107	1010		1127	565	1779	3035
48	100	**2268**	0.493	0.118	128	39	2268	0.493	0.118	128	38	129	292
	200	1802	0.763	0.388	121	40	1802	10.34	1.347	136	42	264	593
	500	3746	0.659	0.618	288	103	3718	47.99	46.28	291	144	654	1443
	800	6019	0.430		466	167	6011	140.5	138.5	473	239	1063	2307
	1000	10118	9.263	8.597	668	219	10118	457.9	47.38	750	250	1313	2920
96	100	**2268**	0.039		-	-	2268	0.039		-	-	-	-
	200	1802	0.493	0.097	-	-	1802	0.493	0.097	-	-	-	-
	500	2624	0.789	0.519	-	-	2624	26.20	1.456	-	-	-	-
	800	3026	1.034		-	-	3009	33.68	32.72	-	-	-	-
	1000	10118	1.847	1.180	581	179	10118	1.847	1.180	585	179	1129	2847
Protein 2		LPT / iZO / mTP					iMF					Standard implementation	
24	100	2138	1.222	0.722	156	49	2138	9.638	2.388	161	51	223	354
	200	4143	0.472	0.388	341	111	4091	28.97	27.80	341	136	478	709
	500	**10647**	**0.472**		867	304	**10603**	283.3	**276.9**	878	389	1216	1783
	800	16120	0.388		1328	441	16089	647.7	628.9	1338	575	1895	2791
	1000	19274	0.222		1581	551	19260	940.8	934.6	1598	730	2306	3512
48	100	2138	0.534	0.118	138	43	2138	0.534	0.118	139	43	158	338
	200	2483	1.138	0.638	184	57	2483	10.63	1.722	209	63	334	673
	500	5326	0.951		456	155	5307	69.70	64.61	457	208	845	1690
	800	8062	0.388		692	235	8048	154.8		715	324	1327	2668
	1000	9666	0.305	0.263	813	**292**	9631	239.2	235.8	830	417	1627	**3319**
96	100	2138	0.039		-	-	2138	0.039		-	-	-	-
	200	2483	0.597	0.097	-	-	2483	0.597	0.097	-	-	-	-
	500	3830	1.581	0.894	-	-	3830	31.97	2.081	-	-	-	-
	800	4067	0.951		-	-	4026	37.53	34.15	-	-	-	-
	1000	4855	0.701	0.618	411	180	4819	58.18	52.43	423	230	1289	3319
DNA 1		LPT / iZO / mTP					iMF					Standard implementation	
24	100	3287	0.555		87	75	3229	6.055	4.722	91	76	97	99
	200	7123	1.138		385	373	7099	24.72		387	375	409	457
	500	18851	0.305	0.138	2264	2161	18756	175.9	163.1	2285	2151	2335	2369
	800	28812	0.222		5587	5638	28735	501.7	493.6	5595	5670	5703	5978
	1000	36950	2.638	2.555	9429	9289	36856	839.8	800.4	9455	9307	9559	9682
48	100	2816	0.576	0.118	78	74	2816	0.576	0.118	78	74	84	98
	200	6050	1.013	0.430	350	367	6050	1.013	0.430	350	367	367	448
	500	9462	0.659		2176	2125	9382	42.28	40.95	2178	2131	2227	2345
	800	14397	0.388	0.347	5438	5581	14381	121.2		5441	5577	5547	5942
	1000	18529	4.388	3.763	9225	9199	18432	207.6	199.1	9242	9209	9364	9641
96	100	2816	0.039		-	-	2816	0.039		-	-	-	-
	200	6050	0.472	0.097	-	-	6050	0.472	0.097	-	-	-	-
	500	5261	1.164	0.727	-	-	5261	13.12	3.644	-	-	-	-
	800	7261	0.868	0.722	-	-	7192	29.18	28.18	-	-	-	-
	1000	16429	1.305	1.180	8878	8921	16429	1.305	1.180	8878	8921	9051	9357
DNA 2		LPT / iZO / mTP					iMF					Standard implementation	
24	100	3584	0.305		96	94	3470	3.972	2.638	96	95	105	120
	200	7295	0.805	0.722	381	354	7219	25.47	23.80	381	353	400	418
	500	19498	0.305		2269	2186	19433	172.5		2269	2157	2313	2341
	800	29095	0.472	0.388	5418	5897	29004	426.5	422.2	5417	5904	5496	6160
	1000	36771	0.222		8863	8754	36729	782.2		8887	8779	8969	9077
48	100	2452	0.368	0.159	85	92	2452	3.076	0.159	90	94	92	119
	200	5512	0.763	0.388	347	348	5512	0.763	0.388	347	348	365	411
	500	9806	0.409		2177	2150	9727	40.99	38.32	2185	2152	2236	2321
	800	14549	0.388	0.305	5277	5848	14505	107.2		5283	5863	5382	6141
	1000	18411	0.138		8682	8696	18368	190.8		8688	8708	8795	9047
96	100	2452	0.039		-	-	2452	0.039		-	-	-	-
	200	5512	0.513	0.097	-	-	5512	0.513	0.097	-	-	-	-
	500	5170	0.560	0.394	-	-	5170	10.43	4.269	-	-	-	-
	800	7358	0.638	0.555	-	-	7262	24.65	22.86	-	-	-	-
	1000	9283	0.472	0.430	8398	8463	9189	45.86	44.57	8412	8484	8545	8962

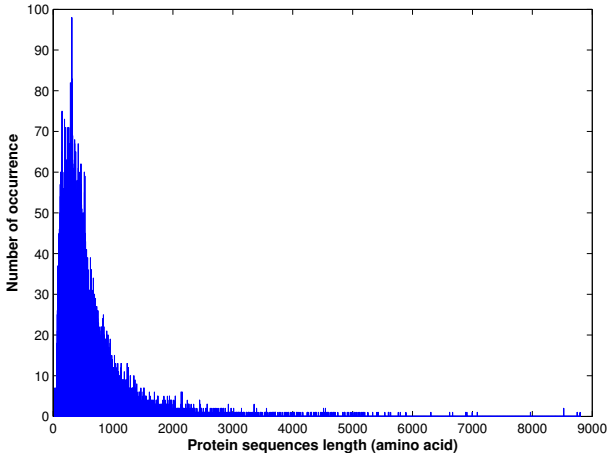

Figure 2. Human protein sequences length distribution. Five proteins that are longer than 9000 amino acids are not shown in the histogram.

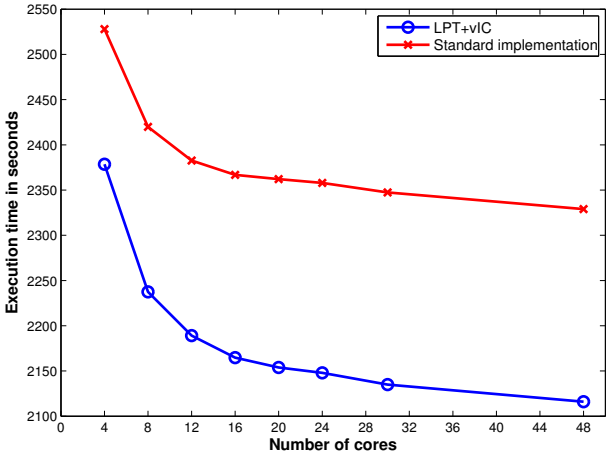

Figure 3. Execution times for 500 DNA data partitions under the CAT model (DNA 1).

distribution strategy and the scheduling heuristics including vIC improvement. The results are depicted in Table II.

Across all test configurations, LPT, iZO, and mTP returned identical results. This suggests that partition interchange and/or exchange as implemented in iZO and mTP does outperform LPT in our experiments. Generally, iMF yielded a smaller makespan, but at the cost of a substantially larger per-core partition number variance. When we applied vIC, the variance could be further reduced without increasing the makespan in most of the cases. Nonetheless, the iMF partition number variance is still much larger than for other algorithms, even *after* applying vIC to correct for this. As indicated by our results, the processing times for iMF-based data distribution is still longer than for alternative approaches, because of the high partition number variance and despite the fact that the makespan is smaller.

With respect to absolute execution times, LPT (iZO and mTP) returned the best results. For one protein dataset (1000 partitions on 48 cores) under the CAT model of rate heterogeneity [20], we observed an 11-fold speedup with respect to the standard implementation. For protein data under the Γ model of rate heterogeneity [21] the average speedup is around two-fold. On the DNA datasets, we observe an average speedup of 1.386 under CAT model (see Figure 3) and 1.046 under Γ speed up.

The main reason for the differences between DNA and protein data is that matrix exponentiation of the 20×20 Q matrix is substantially more expensive than for the 4×4 nucleotide substitution matrix. The difference between the CAT and Γ models of rate heterogeneity is due to the fact, that more Q matrix exponentiations are required per partition to obtain the transition probability matrices P for each per-site rate category r_i (for details please see [20]). Overall, the expected performance improvements depend on the ratio of CPU cycles required for exponentiating the Q matrix and for computing Equation 1.

The iMF-based partition distribution is always slower than LPT (iZO and mTP) because of the large variances. See, for instance, protein dataset 2 with 500 partitions on 24 cores where iMF returned a makespan of $10,603$ and LPT(iZO and mTP) a makespan of $10,647$ site patterns. However, the partition number variance of iMF is 276.9 compared to a variance of only 0.472 for LPT(iZO and mTP). Therefore, the iMF-based partition distribution is 28% slower than the LPT(iZO and mTP)-based approach under the CAT model of rate heterogeneity. With respect to parallel scalability, the standard implementation (cyclic distribution of sites) scales poorly above 4 cores, while the LPT(iZO and mTP)-based parallelization scales well, in particular on large protein data sets under the CAT model (see Figure 4).

The simple LPT heuristics, return the best results in our test data. This is not only because it can distribute the processing time (number of site patterns) evenly among cores, but also because it assigns an approximately equal number of partitions to each core. Thus, each core calculates the likelihood vector entries on approximately the same number of site patterns *and* computes roughly the same number of P matrices. iMultifit did not perform well, despite the fact that it can distribute the sites more evenly among cores than LPT. This is because iMultifit assigns few large partitions to cores with the lower index number, and many, small partitions to cores with higher index numbers. Therefore, cores with higher index numbers need to calculate many more P matrices such that the load balance actually becomes worse. Hence, more elaborate heuristics than LPT, did not substantially improve the makespan. This may be due to the distribution of human gene lengths we sampled from to obtain partition lengths and the limited number of data

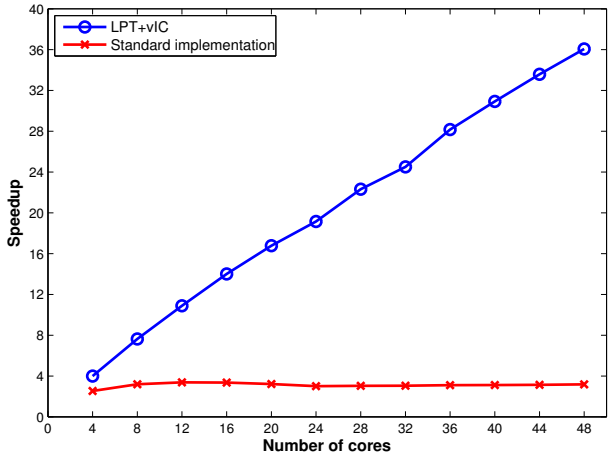

Figure 4. Speedup of LPT+vIC over the standard implementation (cyclic distribution of sites) for 1000 protein data partitions under the CAT model (Protein dataset 2).

sets we tested. One may expect more elaborate heuristics to improve the makespan when there is a large number of very small jobs. However, one important result from our study is that, the simple LPT heuristics that are easy to implement already yield substantial performance improvements.

In general, multiprocessor scheduling algorithms for partitions should be used and implemented for cases where $p \gg n$. In cases where the largest partitions is greater than the average completion time $C = \sum_{i=1}^{n} C_i/n$, the makespan will be dominated by this partition and the code will hence not scale well. One such example is the protein dataset 1 with 100 partitions.

VI. Conclusion & Future Work

To the best of our knowledge, this paper is the first to describe the analogy of the multi-processor scheduling problem to load balance issues in large parallel partitioned phylogenetic analyses. Essentially, the phylogenetic scheduling/data distribution problem represents a bi-criterion problem, since the number of sites *and* partitions assigned to each core needs to be balanced.

We show that, the simple and fast-to-compute LPT heuristics work well for solving the load balance problem. LPT has already been integrated into the production-level version of RAxML-Light. Moreover, we show that, parallel execution times can be improved by up to one order of magnitude for partitioned protein datasets under the CAT model of rate heterogeneity when partitions are distributed monolithically to cores using LPT. The techniques we have developed and assessed here, can be generally applied to all likelihood function implementations (Bayesian and Maximum Likelihood inference) that exploit the intrinsic parallelism of the PLF at a fine-grain level.

With respect to future work, we intend to integrate a mechanism in RAxML that will automatically and adaptively determine whether cyclic or monolithic scheduling shall be deployed. Moreover, we will work on fine-tuning and further optimizing load balance by allowing for a mix of cyclic as well as monolithic data distribution of sites/partitions to cores. The problems that we will encounter in this context are similar to the multi-processor scheduling problem *with* preemption.

Acknowledgment

Part of this work was supported by the Graduate School for Computing in Medicine and Life Sciences funded by Germanys Excellence Initiative (DFG GSC 235/1) and by a visiting PhD student scholarship of the Heidelberg Institute for Theoretical Studies.

References

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

[2] 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*. ACM, 2007, p. 4.

[3] A. Stamatakis and M. Ott, "Exploiting fine-grained parallelism in the phylogenetic likelihood function with mpi, pthreads, and openmp: A performance study," *Pattern Recognition in Bioinformatics*, pp. 424–435, 2008.

[4] 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*, 2011.

[5] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete Mathematics*, vol. 5, no. 2, pp. 287–326, 1979.

[6] D. Johnson and M. Garey, "Computers and intractability: A guide to the theory of np-completeness," *Freeman&Co, San Francisco*, 1979.

[7] M. DellAmico and S. Martello, "Optimal scheduling of tasks on identical parallel processors," *ORSA Journal on Computing*, vol. 7, no. 2, pp. 191–200, 1995.

[8] E. Mokotoff, "An exact algorithm for the identical parallel machine scheduling problem," *European Journal of Operational Research*, vol. 152, no. 3, pp. 758–769, 2004.

[9] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[10] G. Finn and E. Horowitz, "A linear time approximation algorithm for multiprocessor scheduling," *BIT Numerical Mathematics*, vol. 19, no. 3, pp. 312–320, 1979.

[11] E. Coffman Jr., M. Garey, and D. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, pp. 1–17, 1978.

[12] M. Gualtieri, P. Pietramala, and F. Rossi, "Heuristic algorithms for scheduling jobs on identical parallel machines via measures of spread," *IAENG International Journal of Applied Mathematics*, vol. 39, 2009.

[13] G. Paletta and F. Vocaturo, "A composite algorithm for multiprocessor scheduling," *Journal of Heuristics*, vol. 17, no. 3, pp. 281–301, 2011.

[14] L. Tang and J. Luo, "A new ils algorithm for parallel machine scheduling problems," *Journal of Intelligent Manufacturing*, vol. 17, no. 5, pp. 609–619, 2006.

[15] E. Hou, R. Hong, and N. Ansari, "Efficient multiprocessor scheduling based on genetic algorithms," in *Industrial Electronics Society, 1990. IECON'90., 16th Annual Conference of IEEE*. IEEE, 1990, pp. 1239–1243.

[16] A. Stamatakis and M. Ott, "Load balance in the phylogenetic likelihood kernel," in *Parallel Processing, 2009. ICPP'09. International Conference on*. IEEE, 2009, pp. 348–355.

[17] M. Langston, "Improved 0/1-interchange scheduling," *BIT Numerical Mathematics*, vol. 22, no. 3, pp. 282–290, 1982.

[18] C. Lee and J. David Massey, "Multiprocessor scheduling: combining lpt and multifit," *Discrete applied mathematics*, vol. 20, no. 3, pp. 233–242, 1988.

[19] 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.

[20] A. Stamatakis, "Phylogenetic models of rate heterogeneity: a high performance computing perspective," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. Ieee, 2006, pp. 8–pp.

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