Optimal Data Distribution for Phylogenetic Inference with Site Repeats via Hypergraph Partitioning

Benoit Morel, Sebastian Schlag, and Alexandros Stamatakis

Abstract

We recently introduced the so-called site repeats technique for accelerating the phylogenetic likelihood function (PLF). The site repeats technique identifies identical patterns among MSA sites and thereby omits redundant likelihood calculations. The consequence of using site repeats is that the computational cost, in terms of floating point operations per site, varies over the multiple sequence alignment (MSA) or partitions thereof. Note that, in standard PLF implementations, computing the per-site likelihood has the same computational cost for each site. The fact that the computational cost varies across sites in PLF implementations that use the site repeats technique complicates parallelization as allocating the PLF computations of two sites to two different processors might lead to loosing some site repeats. In other words, by splitting a pair of sites that shares "many" repeats, the computational cost for calculating the likelihood at each site can increase. We will try to address the problem of optimally assigning MSA sites to cores in this practical by modeling the problem as a hypergraph problem.

1 Preliminaries

A *multiple sequence alignment* (MSA) consists of the sequences of multiple taxa/species. The columns of the MSA are called *sites*. All DNA characters in a single MSA site are assumed to share a common evolutionary history [13]. In most current empirical phylogenetic data analyses, the sites of the MSA are typically subdivided into *partitions* (e.g., genes or grouping the 1st and 2nd codon position into one partition and the 3rd into a separate partition, see Figure 1 for an example).

Given the MSA and a partitioning scheme, we can calculate the likelihood on a given candidate tree. In partitioned analyses, individual partitions of a MSA, have a separate set of likelihood model parameters (GTR rates, α shape parameter of the Γ model of rate heterogeneity, sometimes also a distinct set of branch lengths). In current tools for likelihood-based phylogenetic inference (Maximum Likelihood and Bayesian Inference) PLF calculations typically account for 85-95% of total runtime. Thus, the PLF is *the* target function for optimization and parallelization. Please see Figure 2 for a schematic outline of a naïve parallelization of PLF calculations. In this example we assume that there is only one partition and that the MSA sites are distributed to the cores in a cyclic fashion. Note that, for partitioned MSAs, we need to assign sites to partitions in a more sophisticated way as explained below and particularly in reference [7]).

In the standard case (i.e., without site repeats), we are given a list of k partitions and c CPUs. As the per-site cost for calculating the likelihood is identical for each site, each partition has a computation cost that is essentially (but, see below for the difference) linear to the number of sites/columns it comprises. Analogously

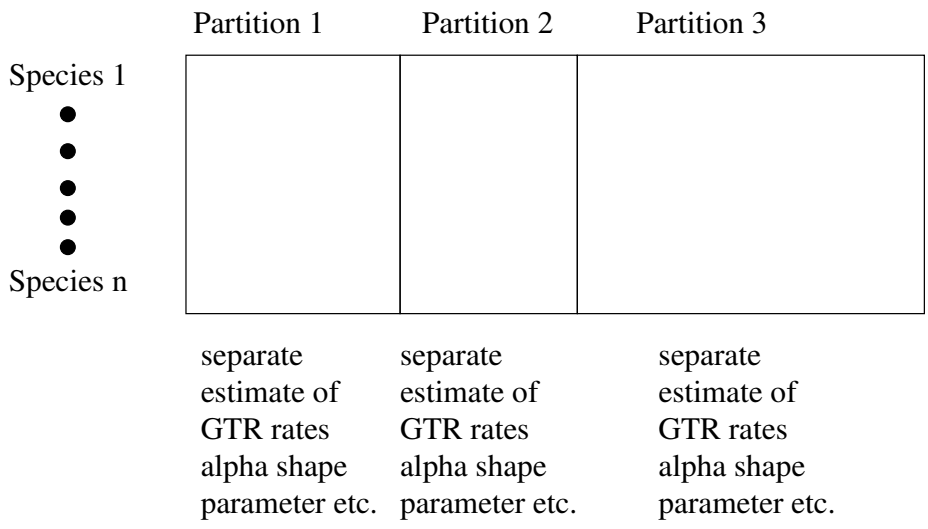

Figure 1: A partitioned MSA.

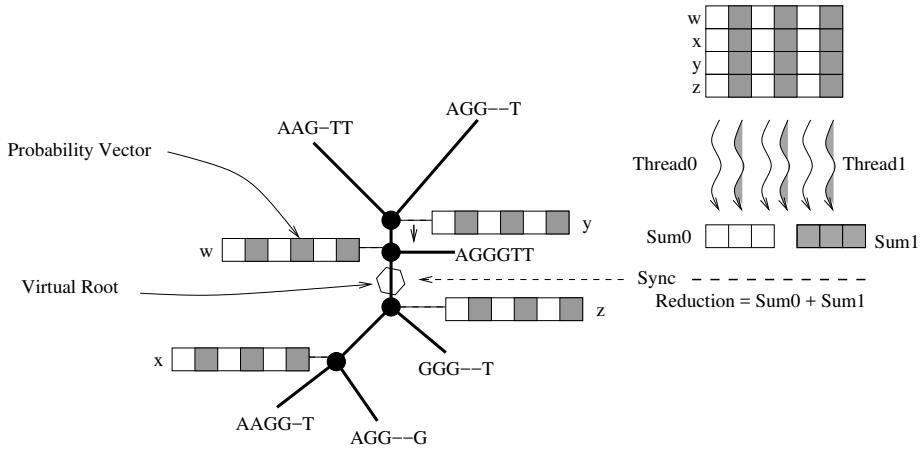

Figure 2: Schematic outline of the parallelization scheme for a simple unpartitioned MSA with a cyclic distribution of sites to threads/cores/MPI processes.

to the classic bin packing problem, we try to optimally assign the partitions (items) to the CPUs (bins).

The *first key difference* is that the number of CPUs c is given, and that we intend to balance the data among all CPUs. In other words, we want to minimize the maximal per-CPU load. The *second key difference* is that partitions *are* divisible, since a partition consists of (originally independent) sites/subelements. Thus, for improving load balance, we can split partitions into disjoint sets of sites that are allocated to distinct CPUs. But splitting a partition comes at a cost. This is the *third key difference*, as the computational cost of each partition has actually two components: The fundamental challenge is that all partitions have an identical

constant base cost α. This base cost α is the exponentiation of the Q matrix, i.e., the calculation of $P(t)$ for a given branch length t. Note that, the transition probability matrices P are different among distinct partitions, as for each partition, we typically estimate a separate rate matrix (model) Q.

Thus, if we split a single partition among two or more CPUs, each subset of that partition incurs this base cost α on the CPU it is assigned to. Thus, if the sites of a partition are assigned to two distinct CPUs, α needs to be computed redundantly, that is, once on each CPU.

Alternatively, we could also parallelize the $P(t)$ (α) calculations and then broadcast the $P(t)$ values to all CPUs. However, based on previous computational experiments, the $P(t)$ calculations are too fine-grained and frequent, to allow for efficient parallelization. Therefore, we have to unfortunately compute $P(t)(\alpha)$ redundantly at several CPUs if the sites of a partition have been split up and allocated to more than one CPU.

The second component of the per-partition PLF calculation is the variable cost ϕ. For standard PLF implementations ϕ is linear in the number of sites per partition. This is because the same amount of arithmetic operations is required to compute individual per-site likelihoods. Since MSA sites are assumed to evolve independently in the likelihood model, calculations on a single site of a partition can therefore be performed independently *of* and concurrently *to* all other sites. Therefore, we can easily distribute the sites of a single partition to several CPUs (see again Figure 2).

Thus, based on the prolegomena, to maximize parallel efficiency, we need to minimize redundant calculations of $P(t)(\alpha)$ by only splitting partitions when necessary, while distributing sites evenly among CPUs" [13].

As mentioned in the abstract *site repeats* (SR) are a technique for accelerating PLF calculations. The technique takes repeating MSA site patterns in subtrees of the phylogeny into account to reduce the amount of per-site floating point computations. Note that, the exact amount of savings depends on the actual tree topology!

Essentially, the SR-based optimization of the PLF reduces the computation cost ϕ by detecting and re-using identical intermediate PLF results among two or more sites. As a consequence, distinct sites of a partition now have *varying* computational cost in terms of arithmetic operations per site. Furthermore, as mentioned before, if we assign two sites that share a large fraction of intermediate results to different CPUs, the accumulated computation cost ϕ for those two sites will actually increase. Note that, communicating intermediate shared site computation results between CPUs does not represent a viable solution, since the computation to communication ratio is unfavorable. Analogously to the argument for recomputing $\alpha(P(t))$ we can not communicate such intermediate results because of the extremely fine-grained nature of PLF computations. This has implications for distributing data of SR-based parallel PLF implementations to distinct processors. While we can still arbitrarily split partitions among CPUs, we will have to sacrifice some site repeats. Thus, as for α, we will now need to also conduct some redundant likelihood computations of the component ϕ for sites that (i) belong to the same partition and (ii) that share repeats if these sites are allocated to distinct CPUs [13].

In general, in this practical, we will be interested to devise algorithms for optimally splitting the sites of a partition such as to achieve optimal load balance

and minimize the amount of lost repeats induced by this splitting. The cool thing is that we can easily compute a lower bound for the maximum number of site repeats (minimum number of lost site repeats) as this is just the number of site repeats for executing the PLF on a single processor (see [13]).

2 Phylogenetics

A partitioned MSA consists of t taxa, s sites $S = \{s_1, s_2, \ldots, s_s\}$, and a partition scheme of p disjoint *partitions* (e.g., into genes) such that $p_i \in 2^S$ (see Figure 1). The *general* load balance problem is to distribute theses partitions among c CPUs, such that the maximum computational cost among the CPUs is minimized. Given a set P_i of partitions assigned to CPU i, the computational cost of CPU i is defined as

$$\mathcal{C} := |P_i|\alpha + \sum_{p \in P_j} \phi(p)$$

, where ϕ is the SR-based PLF cost for a specific subset of sites from a partition.

Definition 1 (Site Repeats (SRs) [8])**.** *Let T_u be the subtree of T that presents the evolutionary relationships among the taxa at the leaves $L(T_u)$. We denote the sequence of the i-th taxon $x^i = x^i_1 x^i_2 \ldots x^i_n$. Two sites j and k are called* ***repeats*** *of one another* iff *$x^i_j = x^i_k$ for all taxa i, $1 \leq i \leq |L(T_u)|$, in T_u.*

Observation 1 ([8])**.** *Let u be a node whose children are nodes v and w. Then, two sites j and k are repeats in T_u iff j and k are also repeats in T_v and T_w.*

Definition 2 (Repeats class)**.** *Let T_u be a subtree. Let R be a set of sites that are repeats of one other in T_u. Then R forms a repeats class if there is no other site that is a repeat of any elements of R in T_u.*

Definition 3 (Repeats Class Count (RCC))**.** *Let T_u be the subtree of T under the node u. Then the RCC of u is the number of repeats classes in T_u. The RCC of T is the sum of the RCCs of its internal nodes.*

For a given tree T and a given partition p, $\phi(p)$ is proportional to the RCC of T. A partition with a high proportion of repeats will have a low RCC because almost all pairs of sites in that partition will share repeats with each other (i.e., there are but a few classes). Thus, PLF computations on such partitions can be substantially accelerated via site repeats. As a consequence, the RCC value constitutes *the* target quantity for minimization and optimizing load balance (data distribution).

2.1 Background Reading

Reading the background papers is important for getting started, so please read them first, before starting with the practical!

For gaining a detailed understanding of site repeats you must read the following paper [8].

For more details about optimal data distribution for standard PLF implementations *without* SRs please read [7]. This paper shows that the data distribution problem for standard PLF implementations is already NP-hard. It also introduces

```
2 3
partition_0 6
0 1 2 3 4 4
0 0 0 1 2 3
0 1 2 3 4 5
partition_1 4
0 1 2 3
0 1 1 2
0 1 2 3
```

```
2
CPU1 1
partition_0 5 0 1 2 3 4
CPU2 2
partition_0 1 5
partition_1 4 0 1 2 3
```

Figure 3: Left: a repeat file with two partitions and a tree with three internal nodes. Right: a distribution of these sites among two CPUs. Each CPU was assigned 5 sites.

an approximation algorithm for data distribution that has a very tight bound (i.e., will only do one more redundant α calculation than the optimal solution in the worst case).

For a first paper on heuristic approaches to data distribution for SR-based PLF implementations read [13]. For a follow-up paper with a simple, yet efficient ad hoc heuristic please read [10]. Evidently, your goal shall be to try and out-compete this ad-hoc heuristic. Note that, we do not know yet if the SR-based data distribution problem is actually NP-hard.

Please let us know if you can't access one of the above articles and we will share them electronically with you.

2.2 SR software interface

In the following, we quickly outline the software interface of a tool that computes the overall RCCs (repeats classes count) for a given tree, alignment, partitioning scheme, number of processors c, and assignment of sites to cores.

The tool and some example files are available via our RepeatsCounter[1] repository.

2.2.1 Repeats file format

A repeats file (see example in Fig. 3) shall be the input of your program. It is generated from a partitioned MSA and a tree.

The repeats file starts with the number of partitions, a space, and the number of internal nodes of the tree (for n taxa the number of inner nodes in an unrooted binary tree is $n-3$). This is followed by a blocks describing the partitions (partition blocks).

A partition block starts with the partition name, a space, and the number of sites in this partition. Then, it contains one line per internal node of the tree. Each element in the line corresponds to a site, and is the repeats class identifier of this site. Elements are separated by spaces. All sites that have the same repeats class identifier belong to the same repeats class (see section 2). The identifiers can range from 1 to the RCC of the partition and node corresponding to this line.

[1] https://github.com/BenoitMorel/RepeatsCounter

2.2.2 Data distribution file format

The data distribution file describes which CPU receives which site(s) from which partition.

It contains one data block per CPU. A block starts with the CPU name, a space, and the number of partitions that CPU works on. Then, it contains one line per partition. Each line comprises the partition name, and a sequence of site identifiers, separated with spaces. A site identifier corresponds to the index of the the site in its original input data partition, starting from 0.

2.2.3 RepeatsCounter tool

RepeatsCounter is a tool that analyzes a data distribution file, given a repeats file. In other words, it evaluates the quality of your distribution of sites among CPUs. RepeatsCounter will output several useful statistics, including the CPU with the worst (highest) *RCC*. This value is the one that you should minimize. Please read the README to understand how to build and use RepeatsCounter.

2.2.4 Benchmark datasets

We provide several datasets to evaluate your algorithms in the aforementioned github repository (`datasets.tar.gz`).

2.3 Initial problem statement

Our goal will be to initially formulate the data distribution problem with site repeats as a graph optimization problem (see Section 3). We should initially consider the case of one partition on c CPUs and subsequently consider the case of several partitions. We will first spend some time (first milestone meeting) to assess how easy or difficult it is to show that the problem is NP-hard, if formulated as a graph problem. Our intuition is that, if re-formulated as a graph problem (see page 34 of the Bachelor thesis at https://sco.h-its.org/exelixis/pubs/bachelorConstantin.pdf for a visualization of a site repeats graph between MSA sites), it might be straight-forward to prove this, but you might prove us wrong.

In a second step, the goal will be to use some of the graph partitioning tools (see below) developed by Sebastian to see (i) if they help solve the problem and (ii) if they can out-compete the ad hoc heuristic presented in [10]. Of course, you can also come up with your own algorithmic ideas for solving the problem. In the end, we want to have an open-source code that can compute an optimal data distribution for a given tree, partitioning scheme, and number of cores c.

3 Graphs and Hypergraph Partitioning

Hypergraphs are a generalization of graphs, where each (hyper)edge (also called *net*) can connect more than two vertices. Given an undirected hypergraph $H = (V, E)$, the *k-way hypergraph partitioning problem* is to partition the vertex set into k disjoint subsets, called *blocks*, of bounded size (at most $1 + \varepsilon$ times the

average block size) such that an objective function involving the cut hyperedges is minimized.

The two most prominent objective functions are the *cut-net* and the *connectivity* (or $\lambda - 1$) metrics. Cut-net is a straightforward generalization of the edge-cut objective in graph partitioning (i.e., minimizing the sum of the weights of those nets that connect more than one block). The connectivity metric additionally takes into account the actual *number* λ of blocks connected by a net. By summing the $(\lambda - 1)$-values of all nets, one accurately models the total communication volume of parallel sparse matrix-vector multiplication [3] and once more gets a metric that reverts to edge-cut for plain graphs.

Hypergraph partitioning (HGP) has a wide range of applications. Two prominent areas are VLSI design and scientific computing (e.g., accelerating sparse matrix-vector multiplications) [11]. While the former is an example of a field where small optimizations can lead to significant savings, the latter exemplifies problems where hypergraph-based modeling is more flexible than graph-based approaches [3].

It is well known that HGP is NP-hard [9], which is why practical applications mostly use heuristic *multilevel* algorithms. These algorithms successively *contract* the hypergraph to obtain a hierarchy of smaller, structurally similar hypergraphs. After applying an *initial partitioning* algorithm to the smallest hypergraph, contraction is undone and, at each level, a *local search* method is used to improve the partitioning induced by the coarser level. The intuition behind this approach is that a good partition at one level of the hierarchy will also be a good partition on the next finer level. Hence, depending on the definition of the neighborhood, local search algorithms are able explore local solution spaces very effectively in this setting.

3.1 Notation and Definitions

An *undirected hypergraph* $H = (V, E, c, \omega)$ is defined as a set of n vertices V and a set of m hyperedges/nets E with vertex weights $c : V \to \mathbb{R}_{>0}$ and net weights $\omega : E \to \mathbb{R}_{>0}$, where each net is a subset of the vertex set V (i.e., $e \subseteq V$). The vertices of a net are called *pins*. We extend c and ω to sets, i.e., $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. A vertex v is *incident* to a net e if $v \in e$. $\mathrm{I}(v)$ denotes the set of all incident nets of v. The *degree* of a vertex v is $d(v) := |\mathrm{I}(v)|$. The *size* $|e|$ of a net e is the number of its pins.

A *k-way partition* of a hypergraph H is a partition of its vertex set into k *blocks* $\Pi = \{V_1, \ldots, V_k\}$ such that $\bigcup_{i=1}^{k} V_i = V$, $V_i \neq \emptyset$ for $1 \leq i \leq k$, and $V_i \cap V_j = \emptyset$ for $i \neq j$. We call a k-way partition Π *ε-balanced* if each block $V_i \in \Pi$ satisfies the *balance constraint*: $c(V_i) \leq L_{\max} := (1 + \varepsilon)\lceil \frac{c(V)}{k} \rceil$ for some parameter ε. For each net e, $\Lambda(e) := \{V_i \mid V_i \cap e \neq \emptyset\}$ denotes the *connectivity set* of e. The *connectivity* of a net e is $\lambda(e) := |\Lambda(e)|$. A net is called *cut net* if $\lambda(e) > 1$.

The *k-way hypergraph partitioning problem* is to find an ε-balanced k-way partition Π of a hypergraph H that minimizes an objective function over the cut nets for some ε. Several objective functions exist in the literature [2, 9]. The most commonly used cost functions are the *cut-net* metric $\mathrm{cut}(\Pi) := \sum_{e \in E'} \omega(e)$ and the *connectivity* metric $(\lambda - 1)(\Pi) := \sum_{e \in E'} (\lambda(e) - 1)\ \omega(e)$, where E' is the set of all cut nets [4]. Optimizing both objective functions is known to be NP-hard [9].

3.2 Reading List and Partitioning Tools

A comprehensive overview of hypergraph partitioning and its applications is given in [11]. To understand how hypergraph partitioning has been used to optimize the communication volume of parallel sparse matrix-vector multiplication please read [3].

KaHyPar[2], developed By Sebastian Schlag, is a multilevel hypergraph partitioning framework for optimizing the cut-net and the $(\lambda - 1)$-metric. Traditional multilevel HGP algorithms contract matchings or clusterings and therefore work with a coarsening hierarchy of $\mathcal{O}(\log n)$ levels. In contrast to this, KaHyPar instantiates the multilevel paradigm in the extreme n-level version, removing only a *single* vertex between two levels. Furthermore, it incorporates global information about the community structure of the hypergraph into the coarsening process. After coarsening, a portfolio of simple algorithms is used to create an initial partition of the coarsest hypergraph. During uncoarsening, strong localized local search heuristics based on the FM (Fiduccia-Mattheyses) algorithm as well as flow-based local improvement algorithms are used to refine the solution. Detailed experimental results show that KaHyPar computes solutions of very high quality, outperforming the state-of-the art multilevel hypergraph partitioning tools hMetis and PaToH. Although KaHyPar can be used as a black box tool in this practical, we refer to [6, 12, 5, 1] for detailed descriptions of its algorithms.

4 Site Repeats and Hypergraphs

As already mentioned, whenever it is not possible to assign a partition of a MSA to a single CPU for attaining optimal load balance it has to be *split* into k parts which are then assigned to k CPUs. The sites are split according to a given split ratio (such that the load on each CPU remains balanced).

The current splitting algorithm [7] (we also call it data distribution algorithm) just splits the partition according to the split ratio. Since it does *not* consider SRs, sites that share a "lot" of repeats *can* be assigned to different CPUs, inducing redundant computations because these repeats are, and can not be shared across CPUs. Hence, the overall SRC decreases (and hence floating point operations increase) compared to a sequential computation of the likelihood for that partition.

Our initial goal will be to improve partition splitting such that each part of the partition that is split up across two or more CPUs contains as many site repeats as possible, while still balancing the overall computational load (i.e., the per-CPU accumulated SRC should ideally be identical among all CPUs). To make things simpler we can get started with only one partition (essentially an unpartitioned MSA) and subsequently extend this to multiple partitions. What we want to do is to maximize the SRC for the sites of each partition that needs to be split. We thereby minimize the variable per-partition cost ϕ (the cost of computing the likelihood of all sites in a partition).

Assume a partition p of a MSA consisting of s sites S_p. Partition p contains x SRs and has to be assigned to k CPUs (see Figure 4).

[2] https://github.com/SebastianSchlag/kahypar

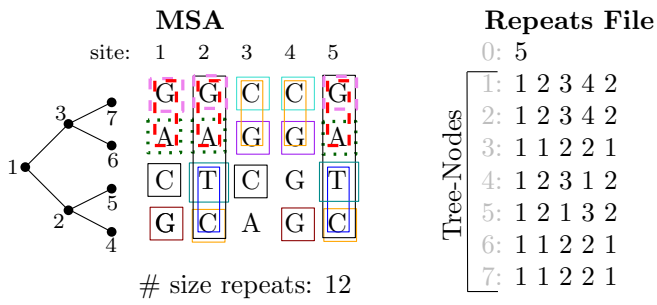

Figure 4: MSA with site repeats and the corresponding repeats file as specified in Section 2.2

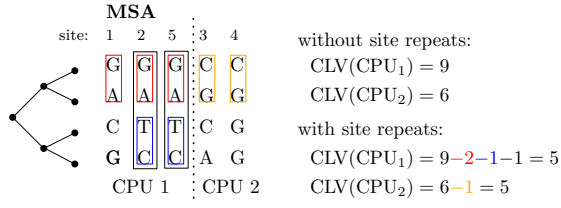

Figure 5: MSA with reordered sites and the corresponding number of CLV (conditional likelihood vector) computations with and without site repeats. For a given MSA with s sites and a given tree T the total number of CLV calculations (without site repeats) is $I(T) \cdot s$, where $I(T)$ is the number of internal nodes in T.

Some key questions are: How can this problem be formulated as a hypergraph partitioning problem? How can we deploy KaHyPar to solve it? Note that, it might be necessary to slightly modify KaHyPar to handle the task at hand. These adaptations will most probably be implemented by Sebastian Schlag.

H

In Figure 5 we depict the operations count (number of CLV entries/cells that need to be calculated) on a given partition and given tree with and without site repeats. This partition has been split up among two CPUs (CPU 1 and CPU 2). Note that, the sites of the partition have been re-ordered in comparison to Figure 4. This is allowed as the phylogenetic likelihood model assumes that all sites evolve independently, so the order doesn't matter. This is true in theory, but in practice we might observe slight deviations in the likelihood score as per-site log likelihoods are accumulated in a different order and this can, and has been observed to, yield slightly deviating overall likelihood scores.

5 Task specification

ilestone 1 (May 24) Formulate the problem for splitting the sites of a single partition among k CPUs as hypergraph problem. Try to prove that this problem is NP-hard.

lestone 2 (June 21) Extend the formulation to several partitions and use the tools written by Benoit (SRC counts) and Sebastian (hypergraph partitioning) to experiment with the data provided by Benoit. Compare your performance in terms of maximum per-CPU accumulated SRC to that of Benoit's ad hoc heuristic

algorithm. Assess the impact of the input tree topology (using Benoit's data) on your performance.

ilestone 3 (July 19) Extend the formulation further (also using the algorithm in [7] to also minimize the number of partitions assigned to each CPU. Remember that our goal is to split as few partitions as possible between CPUs. Provide an open-source code that executes your algorithm and computes a data distribution. **in which format shall we do this?**

of teaching period) Decide together if we will try to write a paper (preferred option, this has worked well in the past, and you will learn more) about this or if you just hand in reports.

The overall goal is to develop an open-source code (using Benoit's and Sebastian's tools) that performs better than Benoit's ad hoc heuristic. We will form 2-3 teams depending on the number of participants. The team that comes up with best algorithm (it also needs to be better than Benoit's algorithm), will be invited for dinner by Alexis.

References

[1] Akhremtsev, Y., Heuer, T., Sanders, P., Schlag, S.: Engineering a direct k-way hypergraph partitioning algorithm. In: 19th Workshop on Algorithm Engineering and Experiments, (ALENEX), pp. 28–42 (2017)

[2] Alpert, C.J., Kahng, A.B.: Recent Directions in Netlist Partitioning: a Survey. Integration, the VLSI Journal **19**(1–2), 1 – 81 (1995)

[3] Catalyürek, Ü.V., Aykanat, C.: Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. IEEE Transactions on Parallel and Distributed Systems **10**(7), 673–693 (1999)

[4] Donath, W.: Logic partitioning. Physical Design Automation of VLSI Systems pp. 65–86 (1988)

[5] Heuer, T., Schlag, S.: Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In: 16th International Symposium on Experimental Algorithms, (SEA), p. 21:1–21:19 (2017)

[6] Heuer, T. and Sanders, P. and Schlag, S.: Network Flow-Based Refinement for Multilevel Hypergraph Partitioning (2018)

[7] Kobert, K., Flouri, T., Aberer, A., Stamatakis, A.: The divisible load balance problem and its application to phylogenetic inference. In: D. Brown, B. Morgenstern (eds.) Algorithms in Bioinformatics, pp. 204–216. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[8] Kobert, K., Stamatakis, A., Flouri, T.: Efficient detection of repeating sites to accelerate phylogenetic likelihood calculations. Systematic Biology **66**(2), 205 (2017). DOI 10.1093/sysbio/syw075

[9] Lengauer, T.: Combinatorial Algorithms for Integrated Circuit Layout. John Wiley & Sons, Inc. (1990)

[10] Morel, B., Flouri, T., Stamatakis, A.: A novel heuristic for data distribution in massively parallel phylogenetic inference using site repeats. In: High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2017 IEEE 19th International Conference on, pp. 81–88. IEEE (2017)

[11] Papa, D.A., Markov, I.L.: Hypergraph Partitioning and Clustering. In: T.F. Gonzalez (ed.) Handbook of Approximation Algorithms and Metaheuristics. Chapman and Hall/CRC (2007)

[12] Schlag, S., Henne, V., Heuer, T., Meyerhenke, H., Sanders, P., Schulz, C.: k-way Hypergraph Partitioning via n-Level Recursive Bisection. In: 18th Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 53–67 (2016)

[13] Scholl, C., Kobert, K., Flouri, T., Stamatakis, A.: The divisible load balance problem with shared cost and its application to phylogenetic inference. bioRxiv (2016). DOI 10.1101/035840