CHAPTER 15

# Phylogenetic Parameter Estimation on COWs

EKKEHARD PETZOLD, DANIEL MERKLE, MARTIN MIDDENDORF,
ARNDT VON HAESELER, and HEIKO A. SCHMIDT

**Abstract:** Phylogenetic analysis is a routine task in biological research. The growing amounts of biological sequence data makes the computational part a bottleneck of the analysis of these data. Parallel computing is applied to reduce the computational burden. In this chapter we discuss different factors that influences the performance of parallel implementations. Using the example of parameter estimation in the TREE-PUZZLE program, we analyze the performance and speedup of different scheduling algorithms on two different kinds of workstation clusters, which are the most abundant parallel platform in biological research. To that end different parts of the TREE-PUZZLE program with diverse parallel complexity are examined and the impact of their characteristics are discussed. In addition, an extended parallelization for the parameter estimation program is introduced.

## 15.1 INTRODUCTION

Phylogenetic analysis aims to reconstruct the relationship between species, organisms, and so on. With the advent of molecular biology, DNA and protein sequences are routinely used to reconstruct the relationships based on gene trees. Thus, the reconstruction of evolutionary or phylogenetic trees is a fundamental task in everyday biological and biomedical researches.

Genome sequencing projects and efficient sequencing techniques lead to the doubling of public sequence databases every 9 months. Therefore, when analyzing this tremendous data set, the computational part of tree reconstruction causes a bottleneck.

Several facts contribute to this problem. On one hand the large number of possible binary unrooted tree topologies $B(n) = (2n-5)!/2^{n-3}(n-3)!$ [1] with $n$ labeled leaves (typically $n$ sequences) prevents exhaustive searches of the tree space to find a tree that fits the data best with respect to some objective function. In addition, a number of phylogeny reconstruction problems are known to be NP-hard [2–4]. Therefore, it is

*Parallel Computing for Bioinformatics and Computational Biology,* Edited by Albert Y. Zomaya
Copyright © 2005 John Wiley & Sons, Inc.

unlikely that they can be solved in polynomial time. Thus, phylogenetic reconstruction relies on the use of heuristics for even moderately sized data [5–7].

Currently, the computationally most intensive tree reconstruction methods are based on the likelihood principle [8], which tends to give more accurate results when compared with other heuristics. However, those likelihood methods get more and more with common with increasing processor speeds.

Unfortunately, the gain of speed is foiled by the increasing amount of data. Accordingly, also parallel computing has been applied since the 1990s to limit the runtime of phylogenetic analyses [9, 10]. The parallelization of tree reconstruction was intuitive because it requires the computation of large numbers of independent tasks. This is typical for many analyses in molecular biology, for example, comparing a number of sequences against all sequences in a biological sequence database.

Before we start with the description of the phylogenetic parallelization, we introduce some basic notation from computer science and molecular evolution.

### 15.1.1 Parallel Platforms

Three types of parallel computing platforms are popular

1. Massively parallel processors (MPPs) consisting of a large number of processing elements (PEs) communicating via a fast interconnection facility
2. Symmetric multiprocessors (SMPs) where a number of processors share a common memory
3. Clusters of workstations (COWs) typically consisting off-the-shelf workstations coupled with a local area network (LAN) like Ethernet.

In biological sciences, COWs are the most abundant parallel platforms [10]. Typically, such COWs either originate as a network of desktop workstations or have been bought as a dedicated cluster, which typically has been or will be later extended with additional up-to-date machines. Both developments produce heterogeneous workstation clusters — a setting to be considered in the development of parallel software suitable for genetic sequence analysis.

### 15.1.2 Parallel Performance

To effectively use parallel platforms, it is crucial to keep all processors equally busy while keeping the communication overhead low. According to *Amdahl's law* [11] the parallel performance is limited by the sequential fraction $S$ of the program, because only the runtime of the parallelized fraction $P$, also called the *coverage*, can be reduced by parallel computing with $p$ processors. Thus, it is favorable to parallelize large portions of a program. The parallel performance of a parallel program is commonly measured by its *speedup*

$$\text{speedup}\ (p) = \frac{\text{runtime with one processor}}{\text{runtime with } p \text{ processors}} \tag{15.1}$$

where the runtime with one processor is typically measured with the corresponding sequential program missing the overhead of a parallel version. The success of a parallel implementation often depends on the granularity, that is, the amount of independent problems that are executed by the processors until communication is necessary, as well as the way the workload is distributed among the available processors.

In the following, the parameter estimation part of the TREE-PUZZLE program [12] serves as an example to demonstrate the impact of task granularity and the scheduling algorithm on the parallel performance. TREE-PUZZLE implements a maximum likelihood (ML) based method, quartet puzzling [13], to reconstruct evolutionary trees from (biological) sequences. To infer ML based trees, it is necessary to estimate the model parameters by ML method. In sequential programs, this is a time-consuming task and several heuristics are employed [7, 14, 15]. The parallelization of the sequential ML based optimization technique is nontrivial, when developing parallel phylogenetic analysis software.

In the following, we will examine the parallel version of TREE-PUZZLE [12] that uses the message passing interface (MPI) [16, 17] for parallel execution on a heterogeneous and a homogeneous COW.

## 15.2 PHYLOGENETIC TREE RECONSTRUCTION USING QUARTET PUZZLING AND ITS PARALLELIZATION

### 15.2.1 Modeling Molecular Evolution

Here, we give a short introduction to the probabilistic framework that is used in phylogenetic analysis. In the following, we confine ourself to DNA sequence evolution. However, we note that everything transforms easily to model protein sequence evolution. As input serves a DNA sequence alignment (cf. Table 15.1) of $n$ sequences $S_1, \ldots, S_n$ with length $L$ (measured in base pairs). All characters within one column trace back to one common ancestor nucleotide. Q1

The ML reconstruction of a tree with $n$ sequences requires a model of evolution describing how substitutions from one character to another occur at a given position in a sequence alignment $A$ (cf. Table 15.1). Typically, this is modeled by time-homogeneous reversible-time Markov process [18, 19]. This stochastic process

**TABLE 15.1 Alignment Example for Eight DNA Sequences $S_1, \ldots, S_8$** Q2

| | | | | |
|---|---|---|---|---|
| $S_1$ | …TGT**C**T**CA**A**G**G | A**C**TAAGCCAT | GCA**A**GT**GT**AA | GTAT**A**A**GTAT**… |
| $S_2$ | …TGT**C**T**AT**A**G**G | A**C**TAAGCCAT | GCA**A**GT**GC**AA | GTAT**G**A**GTGA**… |
| $S_3$ | …TGT**C**T**CA**A**A**G | A**T**TAAGCCAT | GCA**T**GT**CT**AA | GTAT**A**A**ACAT**… |
| $S_4$ | …TGT**C**T**CA**A**G**G | A**C**TAAGCCAT | GCA**A**GT**GT**AA | GTAT**G**A**GTGA**… |
| $S_5$ | …TGT**T**T**AA**A**G**G | A**C**TAAGCCAT | GCA**A**GT**GC**AA | GTAT**G**A**GTGA**… |
| $S_6$ | …TGT**C**T**CA**A**A**G | A**C**TAAGCCAT | GCA**T**GT**CT**AA | GTAT**A**A**ACAT**… |
| $S_7$ | …TGT**C**T**CA**A**A**G | A**T**TAAGCCAT | GCA**T**GT**CT**AA | GTAT**A**A**ACGT**… |
| $S_8$ | …TGT**C**T**CA**A**A**G | A**C**TAAGCCAT | GCA**A**GT**CT**AA | GTAT**A**A**GTGT**… |

describes the evolutionary history for a single nucleotide position. To specify the evolutionary process, one typically defines an instantaneous rate $R_{ij}$ of substituting nucleotide $i$ by $j$. All possible substitutions are given by an instantaneous rate matrix $\mathbf{Q}$. Additionally, it is usually assumed that the composition of the nucleotides $\pi = (\pi_A, \pi_C, \pi_G, \pi_T)$ remains stationary over time. Because we assume time-reversibility, the elements of $\mathbf{Q}$ can be written as

$$Q_{ij} = \begin{cases} R_{ij}\pi_j & \text{if } i \neq j \\ -\sum_k R_{ik}\pi_k & \text{if } i = j \end{cases} \tag{15.2}$$

where the $4 \times 4$ matrix $\mathbf{R} = (R_{ij})$ is symmetrical and the elements on the main diagonal are 0. $\mathbf{R}$ comprises the nucleotide-specific substitution rates. Equation (15.2) describes the general time-reversible model (GTR) [18, 20], which has six independent rate parameters and three independent frequency parameters (the nucleotide composition). The GTR model can be reduced to a number of simpler models such as the HKY model [21] or Jukes–Cantor model [22], which have less parameters. Given $\mathbf{Q}$, a probability matrix $\mathbf{P}(t) = (P_{ij}(t))$ can be computed [23]. The off-diagonal elements of $\mathbf{P}(t)$ determine the probabilities to observe a substitution from nucleotide $i$ to $j$ in time $t$. The probability of one sequence $S_1$ to evolve to sequence $S_2$ during a given period of time $t$ is the product of the probabilities $P_{s_1(l),s_2(l)}(t)$ to observe the nucleotide pair $[s_1(l), s_2(l)]$ at each alignment column $l$. For two sequences, one can estimate $\mathbf{P}(t)$ by counting the observed nucleotide pairs in the alignment. This estimate of $\mathbf{P}(t)$ can then be used to compute the ML estimate $\hat{t}$, which for simplicity we will call evolutionary distance $D$ between two sequences (see Ref. [24] for details).

This approach can be extended [8] to compute the probability of the data to be produced by a given tree $T$ and an evolutionary model $M$. $M$ includes the necessary parameters such as the character frequencies $\pi$ and the independent rate parameters of $\mathbf{R}$. The probability $\Pr(A_l|T, M)$ of an alignment column $A_l$ is the product of all pairwise probabilities along each branch, summed over all possible (typically unknown) character states at the internal nodes of the tree $T$ [7, 8]. To efficiently compute this probability of the alignment given, the tree Felsenstein [8] suggested a recursive algorithm that avoids multiple redundant computations along the tree. The probability of the entire alignment $A$ of length $L$ is the product of all column probabilities

$$\Pr(A|T, M) = \prod_{l=1}^{L} \Pr(A_l|T, M) \tag{15.3}$$

The probability $\Pr(A|T, M)$ is also called the likelihood $\mathcal{L}(T|A, M)$ of the tree $T$ with branch lengths, interpreted as evolutionary distances between the corresponding nodes of the branch (edge), given the data $A$. To avoid multiplications of small numbers, one computes the log-likelihood

$$\log\, \mathcal{L}(T|A, M) = \log\, \Pr(A|T, M) = \sum_{l=1}^{L} \log\, \Pr(A_l|T, M) \tag{15.4}$$

of the tree.

Biological sequences usually do not evolve with the same rate at all positions of an alignment. Thus, it is often necessary to introduce a column specific rate parameter $r_l$, which is typically assumed to be distributed according to a discrete $\Gamma$-distribution with expectation 1 and variance $1/\alpha$ [25, 26]. If the so-called shape parameter $\alpha$ of the $\Gamma$-distribution is high, there is almost no rate heterogeneity observed among the columns, whereas a low $\alpha$ implies that a high amount of columns evolve at low rates while a small number of sites are evolving fast.

Assuming $\Gamma$-distributed rate heterogeneity, the likelihood of a tree $T$ is computed by

$$\mathcal{L}(T|A, M, \alpha) = \prod_{l=1}^{L} \left( \sum_{h=1}^{c} \Pr(r_h) \Pr(A_l|T, M, r_h) \right) \tag{15.5}$$

or taking the log-likelihood

$$\log\ \mathcal{L}(T|A, M, \alpha) = \sum_{l=1}^{L} \log \left( \sum_{h=1}^{c} \Pr(r_h) \Pr(A_l|T, M, r_h) \right) \tag{15.6}$$

where $c$ is the number of discrete rate categories and each rate $r_h$ is equally likely $[\Pr(r_h) = 1/c]$.

The likelihood is used as a measure for the goodness-of-fit of a tree. Moreover, it can be used to compare different tree topologies and parameter estimates. The parameters of the evolutionary model $M$, namely, the nucleotide distribution $\pi$, the rate matrix $\mathbf{R}$, and the shape parameter $\alpha$, have to be inferred from the given alignment $A$.

### 15.2.2 Quartet Puzzling Algorithm

TREE-PUZZLE uses the ML principle to reconstruct evolutionary trees from biological sequence alignments (Table 15.1), applying the quartet puzzling algorithm [13, 27]. The reconstruction is partitioned in four main steps of different computational complexity

1. *Initialization and Estimation of Evolutionary Parameters (PEst).* The input data are given in the form of an alignment of biological sequences (DNA or protein sequences, cf. Table 15.1). For the underlying model of sequence evolution, the most appropriate model parameters (e.g., rate matrix, rate heterogeneity shape parameter $\alpha$) are estimated from the input data. The estimation procedure will be detailed subsequently.
2. *Maximum Likelihood Step (MLStep).* On the basis of evolutionary model and the estimated parameters, we compute for all $\binom{n}{4}$ quartets (subsets of size 4 from the $n$ input sequences) the ML (15.6 (cf. [8]) of each of the three possible tree topologies (Fig. 15.1) to determine the collection of highly supported quartet topologies [13, 27]. The ML values are computed applying the ML principle as outlined in Section 15.2.1.

3. *Puzzling Step (PStep)*. From the quartet topologies with high ML values, so-called *intermediate trees* containing all given sequences are constructed in a stepwise procedure by inserting sequence by sequence [13]. To that end, we use the neighborhood relations induced from the highly supported quartet topologies. We order all sequences in a random order, say {A, B, C, D, E, F, . . .}. We start with the supported topology of the first quartet {A, B, C, D}. For the starting quartet in Figure 15.2*a*, the sequences A and B are neighbors as are C and D. This neighborhood relation is denoted by AB|CD, depicting the bipartition of the sequences induced by the splitting inner edge (see also Fig. 15.1). To insert the next sequence E, we examine the neighborhood relations in all quartets {u, v, w, E} with u, v, w being part of the already reconstructed subtree. In Figure 15.2*a*, and *b*, the considered quartet topology is AE|BC. To maintain the neighborhood relations, E should not be inserted on the path between sequences B and C. Accordingly, every branch on this path gets a penalty of one. This is repeated for all quartets {u, v, w, E}, counting how many quartets contradict the insertion of E into each branch (Fig. 15.2*b–e*). Sequence E is inserted at the branch with lowest penalty (Fig. 15.2*e* and *f*). Then, the next sequences are inserted accordingly (cf. Fig. 15.2*g–i*) until all *n* sequences are contained in the tree. As ties are broken randomly, this step is repeated many (thousand) times, producing many intermediate trees.
4. *Consensus Step (CStep)*. From all intermediate trees, a majority-rule consensus tree [28] is constructed. This consensus tree contains all branches (bipartitions), which were observed in more than 50% of all intermediate trees.

The four steps leave ample space for parallelization on various levels. The parallel implementations of the MLStep and PStep, which together use well over 90% of the total serial computing time, have shown to be almost optimal [29]. Owing to the large number of independent tasks and the use of an efficient dynamic scheduling algorithm, the speedup of the parallelized steps was almost linear up to 64 processors [12, 29]. Parts of the CStep are also executed in parallel, as the preprocessing of the intermediate trees for consensus construction moved to the end of the PStep and is done concurrently by the worker processes.

In the following we explain the PEst part of the serial TREE-PUZZLE program in more detail. In the subsequent paragraphs, we will describe our current parallel implementation of this part. PEst consists of the following steps:

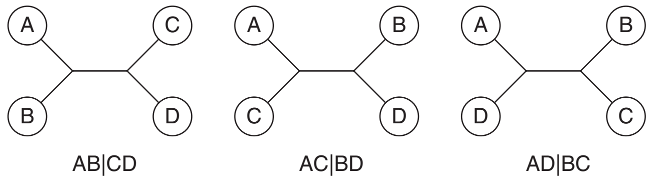

**Figure 15.1** The three possible topologies for a sequence quartet (A, B, C, D) and their corresponding neighborhood relations are shown.

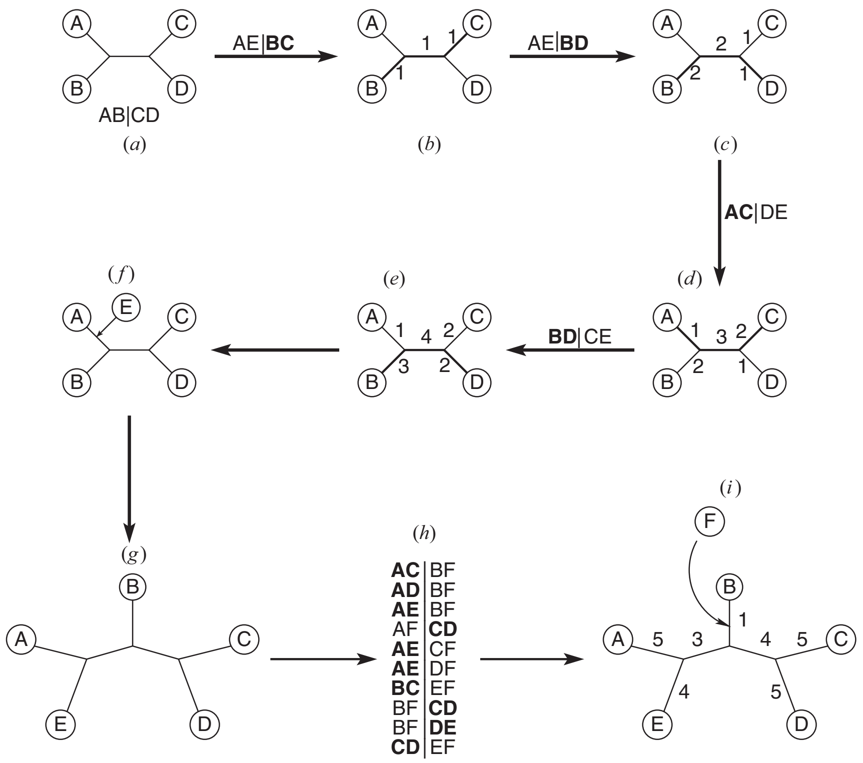

**Figure 15.2** Puzzling step constructing a tree of six sequences: From a starting quartet AB|CD (*a*), sequence F is inserted (*f*) according to the lowest penalty accumulated from the neighborhood relations (*b–e*) in the relevant quartets (AE|BC, AE|BD, and AC|DE, and BD|CE) resulting in a five-tree (*g*). To the five-tree sequence, *F* is again added (*i*) to the branch with lowest penalty computed from the relevant quartets (*h*). Neighborhood relations and associated penalty paths are denoted in bold. Penalties are given at the branches.

(i) Let $\theta$ denote the parameters of the evolutionary model $M$. Initialize $\theta \equiv \theta_0$.
(ii) Compute all pairwise ML distances between sequences (this produces the ML distance matrix **D**, cf. Section 15.2.1) using $\theta_0$.
(iii) Construct a neighbor-joining tree $T$ from **D** [30].
(iv) Compute branch lengths of the neighbor-joining tree $T$ by a least-square fit method to distance matrix **D** [31].
(v) Re-estimate $\theta_0$ as the ML estimate over all admissible $\theta$ using tree $T$ and least-square branch lengths.
(vi) Repeat steps (ii)–(v) until the likelihood of $T$ does not be improved any more.

Figure 15.3 displays the flowchart of the most recent version of parallel TREE-PUZZLE, where the three steps PEst, PStep, and MLStep are parallelized. A master process computes the sequential parts of the program and distributes the parallel parts to several worker processes, which send the results back to the master. The main parts of steps PStep and MLStep are done by the worker processes.

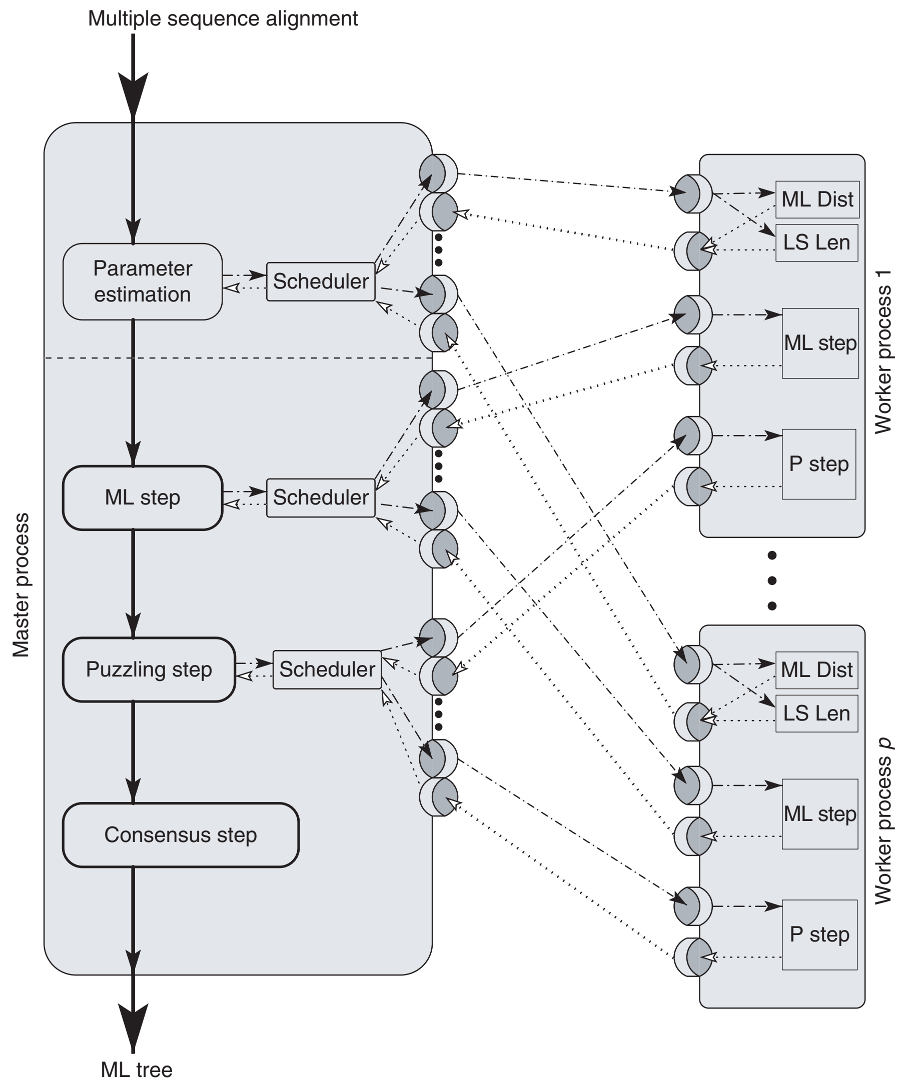

**Figure 15.3** Parallelized work-flow of the current TREE-PUZZLE program. The master process performs the sequential code and distributes the parallel parts to the workers; parallel parts are steps PStep, MLStep, and substeps (ii) and (iv) of PEst.

## 15.3 HARDWARE, DATA, AND SCHEDULING ALGORITHMS

### 15.3.1 Parallel Hardware

Two types of parallel workstation clusters are used to evaluate the performance of the parallelized parameter estimation of TREE-PUZZLE. One cluster is a homogeneous COW (homoCOW) consisting of eight SMP nodes each with four Intel Pentium III

Xeon 550 MHz processors, 512 kb on-chip cache, and 2 Gb RAM. The nodes are coupled with a fast Myrinet network.

The other cluster is a heterogeneous COW (heteroCOW) comprising 12 different SUN Ultra workstations with between one and three CPUs, a total of 15 processors. The machines have CPU clock rates ranging from 143 to 440 MHz and with memory capacities between 128 and 1024 Mb per processor (see Table 15.2 for more detail). A total number of 15 processors is available. The communication between the nodes is handled by 100 Mbit Fast Ethernet.

### 15.3.2 Data Sets

To study the effects of problem size, granularity, and scheduling on the parallel performance, data sets with different alignment lengths (192, 384, 768, and 1536 bp) and numbers of sequences (16, 32, 64, and 128 sequences) were generated using the Seq-Gen program [32].

The implementation was also applied on two biological data sets: data derived from the set of small-subunit ribosomal RNA sequences from Rhodophyta in the European rRNA Database [33] and another data set of Cadherin protein sequences (HBG003908) was collected from the Hovergen database [34].

### 15.3.3 Scheduling Algorithms

To distribute the tasks, a scheduling strategy is required, which facilitates an even load balancing while simultaneously keeping the communication overhead low. The latter is especially important for COWs where a large number of concurrent communication operations cannot be executed very fast.

Scheduling algorithms distribute a set of $N$ tasks to the $p$ processors executing the parallel processes. The two naïve scheduling algorithms are distributing the $N$ independent tasks to the $p$ processes, either in equally sized packages of $B = N/p$ tasks or in an one-by-one scheme ($B = 1$). Whenever a processor finished a batch of task, it receives a new batch until all tasks are done. The former algorithm, *static chunking* (SC) [35], although having a low communication overhead, usually performs

**TABLE 15.2 Specifications of the Heterogeneous COW**

| No. of Machines | CPU Speed (MHz) | No. of CPUs | System Clock (MHz) | Memory (Mb) | Cache (Mb) | Type |
|---|---|---|---|---|---|---|
| 2 | 143 | 1 | 71 | 160 | 0.5 | Sun Ultra 1 SBus |
| 2 | 167 | 1 | 84 | 288–448 | 0.5 | Sun Ultra 1 SBus |
| 1 | 200 | 2 | 100 | 2048 | 2 × 1.0 | Sun Ultra 2 UPA/SBus |
| 3 | 270 | 1 | 90 | 128 | 0.2 | Sun Ultra 5/10 |
| 2 | 360 | 1 | 90 | 320 | 0.2 | Sun Ultra 5/10 |
| 1 | 400 | 3 | 100 | 4096 | 4 × 4.0 | Sun Enterprise 450 |
| 1 | 440 | 1 | 111 | 384 | 2.0 | Sun Ultra 5/10 |

badly if the speed of the processors or the computing times for the tasks differ. The latter algorithm, *self-scheduling* (SS) [35] can guarantee an equal workload on the processors but has in turn a very high communication overhead.

In the first parallel version of TREE-PUZZLE, the parts MLStep and PStep were parallelized [12]. For this implementation, we tested different scheduling algorithms [29, 35], which do not require estimates of a task's runtime. Among the considered schedulers, *guided self-scheduling* (GSS) [36] turned out to show the best performance. GSS assigns task packages of $B_{\mathrm{GSS}} = N/p$ tasks, starting with a $p$th of the full task set and decreasing $B_{\mathrm{GSS}}$ exponentially with every batch of tasks to a minimum of (single) tasks. After a batch of $B_{\mathrm{GSS}}$ tasks has been assigned to a process, the number $N$ of tasks to be scheduled is decreased by $B_{\mathrm{GSS}}$. To gain an even better performance, the GSS algorithm has been generalized by decreasing the starting size of the first package and introducing an *atomic batch size A*, which determines the minimum size of a batch as well as the step size. This *smooth guided self-scheduling* (SGSS) [29] assigns batches of size

$$B_{\mathrm{SGSS}} = \begin{cases} N & \text{if } N \leq A \\ \left\lceil \dfrac{N}{2p} \right\rceil + A - \left( \left\lceil \dfrac{N}{2p} \right\rceil \bmod A \right) & \text{otherwise} \end{cases} \tag{15.7}$$

The first line schedules all remaining tasks if less than the atomic batch size $A$, whereas the second line ensures that each batch size $B$ is divisible by $A$ (for details, see [29]). Again, after scheduling a batch of tasks, $N$ is decreased by $B_{\mathrm{SGSS}}$.

## 15.4 PARALLELIZING PEST

Although parallelizing the parts MLStep and PStep of the TREE-PUZZLE program was straightforward, the PEst part is more difficult to parallelize and needs different scheduling strategies.

### 15.4.1 Parallelizing PEst-(ii): ML Distance Estimation

The pairwise distance estimation [step (ii)] comprises $n^2 - n/2$ independent estimations for $n$ input sequences. Owing to the ML optimization of the distances (cf. Section 15.2.1), the runtime of each estimation is long relative to the time required for a single communication event.

The large number of tasks combined with their long runtime makes this step prominent for an elaborate scheduling approach. We implemented a master–worker scheme where one master process assigns and coordinates the tasks to the $p - 1$ worker processes. As scheduling strategies, GSS and SGSS have been applied. The results are given in the next section.

### 15.4.2 Parallel Performance of PEst-(ii) on a Heterogeneous COW

In this section, we analyze the performance of the SGSS scheduling algorithm to keep the CPUs on a heterogeneous COW equally busy [29] and compare it to the efficiency of GSS [36]. To evaluate the efficiency of GSS and SGSS, we measured the computation times for the pairwise distance computation step (ii) of PEst for the Cadherin data set on the heterogeneous COW.

To simulate the best and worst case scenario with respect to the runtime, we applied two different processor permutations for the assignment of the tasks. In the best case, the first (largest) batches are assigned to the processors in the order of decreasing speed (fastest to slowest CPU). Thus, the fastest processor gets the largest chunk. The second permutation is just the opposite, assigning the chunks from the slowest first to the fastest last, that is, ascending speeds. The latter assignment sequence is supposed to perform worse than the first because the slowest processor gets the largest chunk of tasks. This can be regarded as the worst case scenario that can happen to these scheduling algorithms.

The Gantt charts of a typical pairwise distance estimation [step (ii)] with SGSS and GSS are shown in Figures 15.4 and 15.5, respectively, using identical time scales. Both scheduling algorithms perform well for the best case (Figs. 15.4*b* and 15.5*b*). This means that the load is almost evenly distributed among the worker processes without much idle time at the end of the parallel region. In the worst case, however (Figs. 15.4*a* and 15.5*a*), the GSS scheduler performs badly on the heterogeneous COW. Although

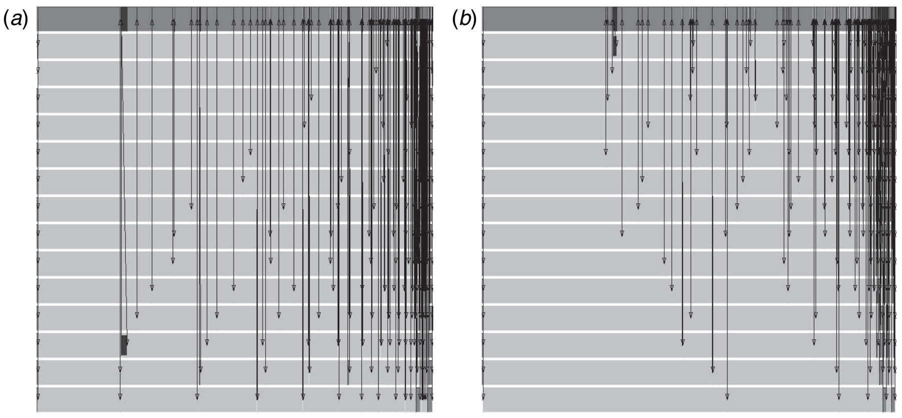

**Figure 15.4** Gantt charts of a single execution of the pairwise distance computation step for Cadherin protein data using SGSS on a heterogeneous COW. Assignment order under (*a*) worst case scenario: The slowest processor receives the largest available batch; (*b*) best case: starting with the fastest CPU. Each row corresponds to one processor sorted by speed according to the earlier assignments. The first row represents the master process distributing tasks to the processors from top to bottom sending new task batches whenever a processor has finished its last batch. Light gray areas depict work and dark gray areas depict idle time and communication or, in case of the master process (first line), coordinating and scheduling the tasks. Vertical arrows represent communication events.

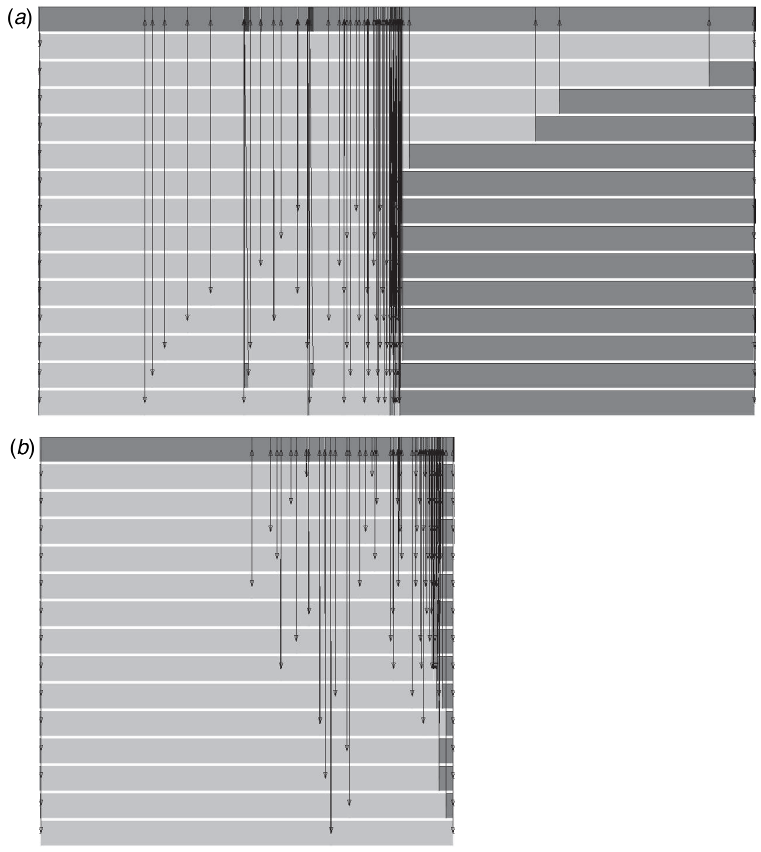

**Figure 15.5** Gantt charts of a single execution of the pairwise distance computation step for Cadherin protein data using GSS on a heterogeneous COW. Assignment order under (*a*) worst case scenario: The slowest processor receives the largest available batch; (*b*) best case: starting with the fastest CPU. Each row corresponds to one processor sorted by speed according to the above assignments. The first row represents the master process distributing tasks to the processors from top to bottom sending new task batches whenever a processor has finished its last batch. Light gray areas depict work and dark gray areas depict idle time and communication or, in case of the master process (first line), coordinating and scheduling the tasks. Vertical arrows represent communication events.

there is no significant difference among the runtimes of the GSS in the best case and SGSS under both scenarios (about 9 s), GSS shows a significant increase of about 60% of its runtime to about 15 s for the worst case scenario (compared with the other cases) and keeping more than 50% of the workers waiting for the tasks to be finished. Although the impact might not be as harsh on average, one can clearly see that there will be a severe increase of running time whenever one of the slower machines gets one of the larger task packages. This effect has been eliminated efficiently by the SGSS algorithm.

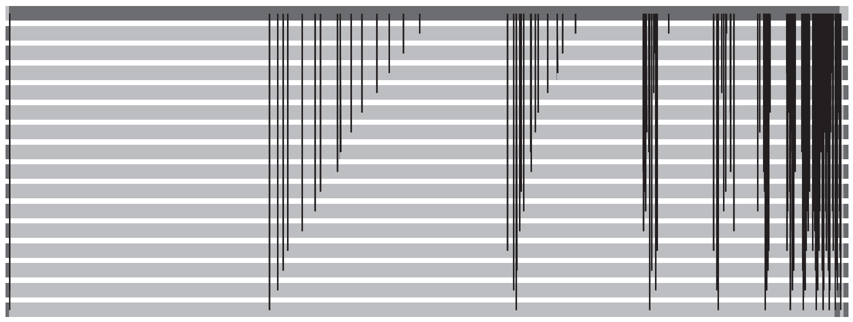

**Figure 15.6** Gantt chart of pairwise distance computation for Cadherin protein data on a homogeneous COW with SGSS: Each row corresponds to one processor, the first as a master process handling the distribution of tasks to the other (worker) processes. Light gray areas depict work and dark gray areas depict idle time and communication or, in case of the master process (first line), coordinating and scheduling the tasks. Vertical lines represent communication events.

For comparison, we also show a Gantt chart from a typical run with Cadherin data on the homogeneous cluster (Fig. 15.6). Note that for this cluster, both assignment sequences are the same because all processors have the same speed. It can be seen that the first chunks have slightly different runtime because of the different size of the chunks. Although the performance results were shown, only for the Cadherin data set, similar results are gained for other protein, DNA, and simulated data sets.

### 15.4.3 PEst-(iii) and (iv): Neighbor-Joining and Least-Square Branch Lengths

The construction of the neighbor-joining tree [30] in step (iii) is very fast and, hence, no parallelization is needed (cf. [29]). The optimization of the branch lengths [step (iv)] in tree $T$ using the least-square method, in contrast, incorporates $N = (2n - 3)(n - 1)$ calculations for $n$ sequences. However, each of the calculations is very short when compared with communication. Hence, the communication overhead was kept minimal by assigning an even share of $N/p$ calculations to each processor (static chunking, cf. Section 15.3.3).

## 15.5 EXTENDING PARALLEL COVERAGE IN PEST

As we have pointed out earlier, it is favorable according to Amdahl's law to increase the parallel coverage within a program if possible.

To identify potential time-consuming blocks in PEst of TREE-PUZZLE, we evaluate its runtime relative to the overall runtime of the program. The aim is to determine

**TABLE 15.3 Runtime of PEst in a TREE-PUZZLE Analysis on homoCOW**

| Data Set | | Sequential Run | | | Parallel (32 Processors) | | |
|---|---|---|---|---|---|---|---|
| Seqs[a] | Length (bp) | PEst (s) | Total (s) | Ratio | PEst[b] (s) | Total (s) | Ratio |
| 16 | 192 | 26.1 | 44.6 | 0.59 | 0.4 | 0.5 | 0.86 |
| 16 | 384 | 14.6 | 51.7 | 0.28 | 0.0 | 0.0 | 0.69 |
| 16 | 768 | 28.3 | 115.9 | 0.25 | 1.6 | 2.4 | 0.68 |
| 32 | 192 | 33.8 | 777.7 | 0.04 | 10.9 | 22.8 | 0.48 |
| 32 | 384 | 57.1 | 1,083.3 | 0.05 | 17.1 | 31.6 | 0.54 |
| 32 | 768 | 102.7 | 1,994.6 | 0.05 | 33.3 | 58.5 | 0.57 |
| 64 | 192 | 180.2 | 28,199.0 | <0.01 | 103.4 | 862.0 | 0.12 |
| 64 | 384 | 276.8 | 32,522.0 | <0.01 | 211.4 | 1,006.5 | 0.21 |
| 64 | 768 | 465.3 | 48,206.0 | <0.01 | 326.8 | 1,485.6 | 0.22 |

[a]Number of sequences.
[b]With parallelized steps (ii) and (iv).

how much increase in performance can be gained by an improved parallelization. The time measurements were performed on the homogeneous cluster for simulated DNA data with different numbers of the sequences and different sequence lengths (see Table 15.3 for details) using TREE-PUZZLE with the SGSS scheduler as described earlier. As the main parts of the quartet puzzling analysis have been shown to be efficiently parallelized [29], we were particularly interested in the runtime of the parameter estimation step. Table 15.3 shows the relative runtimes that are consumed for PEst in sequential as well as parallel runs with 32 processors.

The table shows that the relative amount of time spent for PEst strongly depends on the number of sequences in the test set. For a small number of sequences, 16 sequences, the relative amount of time is more than 25%, whereas for 64 sequences, it is less than 1%. The absolute amount of time consumed in a sequential run has been substantially reduced by the parallelization described in the previous section. This holds especially for small data sets. The relative amount of time needed for PEst in a parallel run on 32 processors has dramatically increased to more than 67% for 16 sequences and still between 12 and 22% for 64 sequences. This clearly indicates that an improved parallelization of PEst is advantageous for further improvement of the overall parallel performance of the program.

### 15.5.1 Parallelizing PEst: Tree Likelihood Computation

The runtimes of all steps of PEst were more closely examined to identify the time-consuming parts. The Gantt chart in Figure 15.7 shows that step (v), which evaluates the likelihood of the tree based on $\theta_0$, should be targeted to extend the parallelization of PEst additionally to the already parallelized steps (ii) and (iv).

Although the overall running time of step (v) covers a long duration likelihood, computations are repeated for 10 to over 100 sequential estimates of $\theta$ to

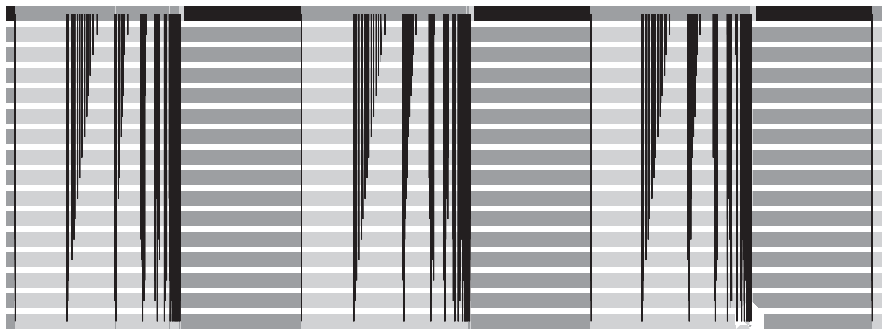

**Figure 15.7** Gantt chart covering 7.16 s of optimization cycles, that is, PEst steps (ii)–(v), with the nonparallelized step (v). The dark parts identify the duration consumed by executing PEst step (v). In the master (first line), the dark gray areas depict step (ii) and black areas depict step (v). Steps (iii) and (iv) are located in the light gray area between steps (ii) and (v). Vertical arrows represent communication events.

find an ML estimate. Hence, the runtime of each iteration is very short relative to communication.

The following parallelization strategy has been chosen to parallelize step (v). The tree likelihood for the parameter $\theta$ is the product over all partial likelihoods over the sites of the sequence alignment. The likelihoods for each alignment position are computed recursively in a tree traversal. As the number of columns in the alignment is typically much higher than the number of $2n - 3$ edges in the tree, the computation of the likelihoods for equal subsets of the alignment columns to the $p$ processors was adopted as a strategy to improve the performance. It is possible to split the alignment such that each processor gets its corresponding section of all sequences in the alignment (Fig. 15.8). After a processor has computed the likelihoods for all sites in its section of the alignment, it computes the likelihood product of that section and returns the result to the master processor. The master computes the overall likelihood product of the entire alignment given the tree and the parameters from the collected section likelihoods.

Owing to the time constraints highlighted earlier, again a hard-wired SC approach was used to avoid communication overhead.

### 15.5.2 Results for PEst-(v) on a Homogeneous COW

To exemplify the performance of the parallel step (v), we analyze its speedup for simulated data. As representative speedup measurements are hardly possible on heterogeneous platforms, we used the homogeneous COW described in Section 15.3.1. Figure 15.9*a* presents the speedup of step (v) for 64 sequences of varying lengths. The parallelization shows promising results. Although a speedup of 9.7 is observed for short sequences (length 192 bp), long sequences show even super-linear speedup (e.g., a speedup of 17.9 for 16 processors with sequences of length 1536 bp).

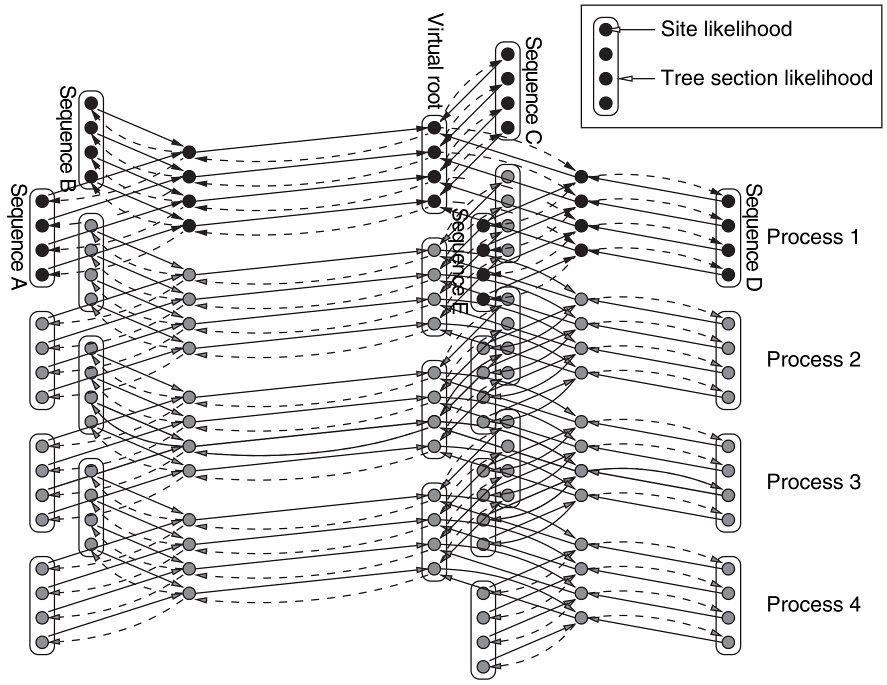

**Figure 15.8** Visualization of the parallel likelihood computation [step (v)] for five sequences (A–D) with alignment length $N = 16$ columns using $p = 4$ processes (1–4). The sequences are depicted vertically, columns horizontally. The dots denote (partial) likelihoods that have to be computed recursively for each position along the structure of tree $T$ starting at the root. The 16 positions are assigned in sections of $B = N/p = 4$ columns to the processors. The section likelihoods at the root are then collected to compute the overall likelihood.

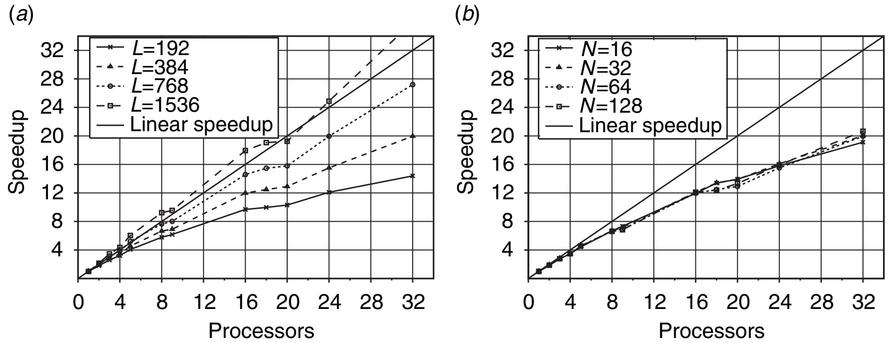

**Figure 15.9** Speedup for step (v) of PEst for simulated data with (*a*) $N = 64$ sequences of different length and (*b*) sequences of length $L = 384$ bp and different numbers of sequences on the homogeneous COW.

This super-linear speedup is due to cache effects. The tree structure with the short sequence sections can be cached completely during the parallelized likelihood computation, whereas the tree with the complete data set does not fit into the local cache of the homogeneous COW causing a caching overhead in the sequential run.

For sequences of length 384 bp, Figure 15.9*b* indicates that the speedup is almost independent of the number of sequences in the data sets. The speedup values are satisfactory for all numbers of sequences examined. For instance, for 32 processors, the speedup for 16 up to 128 sequences is between 19.1 and 20.2.

### 15.5.3 Performance of the Full-Parallel PEst on HomoCOW

Now, we examine the the performance of the parallel version of PEst as presented here. The Gantt chart in Figure 15.10 demonstrates the runtime improvement with the parallelized step (v). Within the time scale of 7.16 s about five estimation cycles could be executed compared with three cycles as shown in Figure 15.7.

The influence of the parallelization on the speedup of the whole PEst step is shown in Figure 15.11. The figure relates the speedup of PEst with (Fig. 15.11*a* and *b*) and without (Fig. 15.11*c* and *d*) the parallelized step (v) for simulated data with $N = 64$ sequences with varying lengths (Fig. 15.11*a* and *c*) and varying numbers sequences of length $L = 384$ bp (Fig. 15.11*b* and *d*).

The speedup obtained without the parallelized step (v) shows early saturation (Fig. 15.11*a* and *b*). An effect that is more pronounced for longer sequences (Fig. 15.11*a*). For instance, the speedup is 6.7 with 16 processors for sequences of length 192 bp, but only 2.9 for longer sequences of length 1536 bp. This is due to the nonparallelized step (v), its time consumption grows almost linearly with the length of sequences.

The speedup substantially improves with step (v) parallelized (Fig. 15.11*c*). The saturation is reached later (i.e., when more processors are used). In addition the level of saturation is higher, thus the speedup is better. Furthermore, the speedup gained for

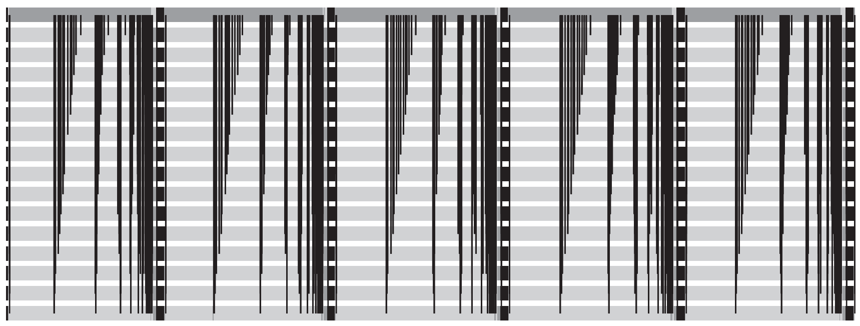

**Figure 15.10** Gantt chart covering 7.16 s of optimization cycles, that is, PEst steps (ii)–(v), after parallelizing step (v) as introduced in Section 15.5. The dark parts identify the duration consumed by executing PEst step (v) on the different processors. Vertical arrows represent communication events.

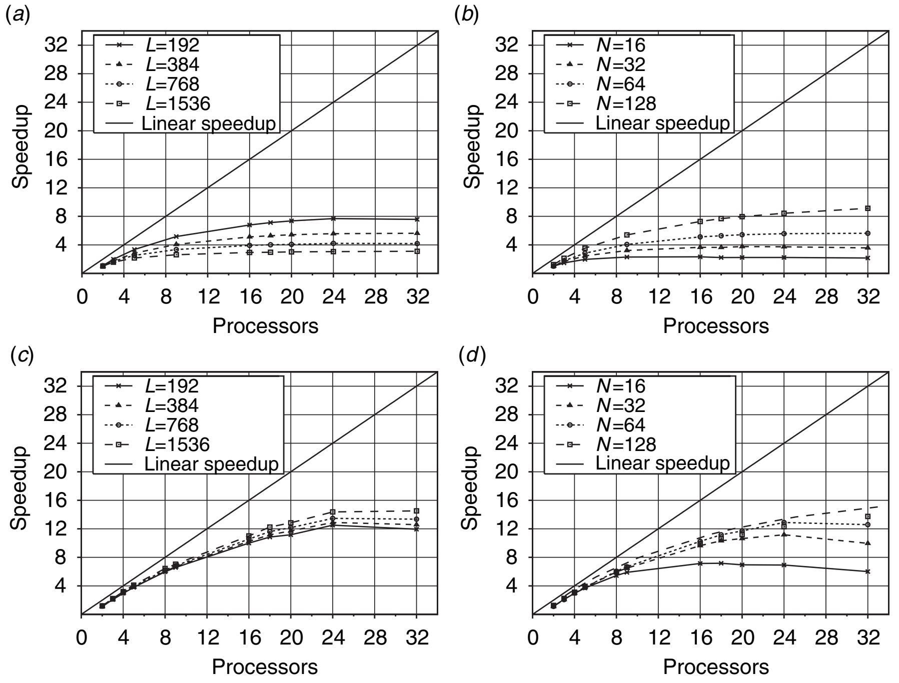

**Figure 15.11** Speedup for PEst without (top) and with (bottom) parallelization of step (v) on simulated data with $N = 64$ sequences of different lengths (left) and sequences of length $L = 384$ bp for different numbers of sequences (right) on the homogeneous cluster.

different sequence lengths is almost identical. Best speedup is reached for the longest sequence this time. Although there is a good speedup up to 16 processors (speedup of $>10$ for all lengths), it reached saturation (between 11.9 and 13.7) with 24 processors.

For a fixed length and sets of different numbers of sequences, the results show (Fig. 15.11*d*) that the speedup for sets with less sequences is worse (earlier saturation) than for larger instances. For large sets with 128 sequences, saturation has not been reached with 32 processors. On the whole, the parallelization of step (v) has significantly increased the speedup for all data set sizes. Using 16 processors, speedup values for data sets with 16 and 128 sequences have increased from 2.3 (and 7.3) to 7.1 (and 9.9). For 32 processors, the speedup has reached saturation for data sets with less than 128 sequences.

*Biological example.* The speedup values for PEst with the improved parallel PEst version of TREE-PUZZLE for the rRNA and protein data are shown in Figure 15.12.

First, we discuss the rRNA example. The speedup saturates early (12–24 processors) for most data sets, only for more than 64 sequences saturation has not been reached with 24 processors. For 16 processors, the speedup is at least 8.6. The speedup becomes worse with 16 and more processors for a small number of sequences and is only 3.9 for 32 processors.

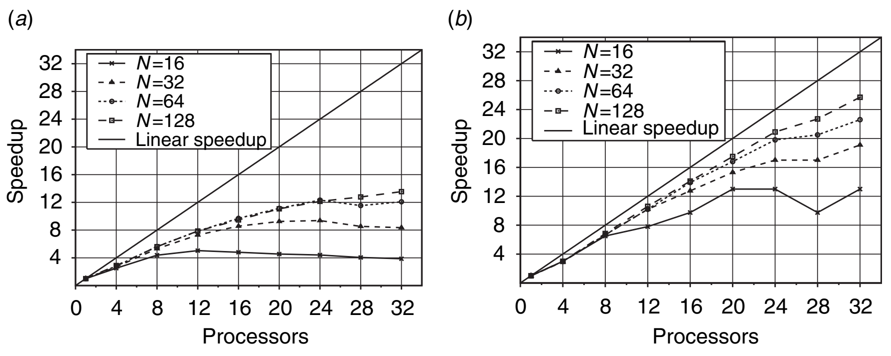

**Figure 15.12** Speedup of the parameter estimation for (*a*) Rhodophyta rRNA data and (*b*) Cadherin protein data measured on the homogeneous COW.

For the protein data, the obtained speedup is better than for the DNA data. For sets with 32 and more sequences, it is at least 12.8 for 16 processors. Even for the small number of 16 sequences, it is 9.8 for 16 processors. The effect of increasing speedup for larger problem sizes on identical numbers of processors is called Amdahl effect [37, 38]. It is caused by a lower complexity of the communication overhead compared with the time complexity of the problem size.

## 15.6 DISCUSSION

The parallel performance of an implementation is limited mainly by the coverage, that is the parallel proportion of the problem, and *granularity*, that is, the amount of work performed until communication is necessary.

The two targets of parallelizing the parameter estimation prior to phylogenetic analysis as delineated earlier exhibit nicely the effects of coverage as well as granularity. The improved parallelization introduced in Section 15.5 aims at the increase of the coverage, after runtime measurements have identified step (v) as consuming most of the runtime (nonparallelized on one processor). After parallelization of step (ii), the relative duration of step (v) became dramatically apparent (cf. Fig. 15.7), now being the limiting time factor to the parameter estimation.

Increasing the coverage alone does not guarantee a good parallel performance. Within the different parallel regions their granularity plays an important role, determined by the size of the independent tasks as well as the scheduling technique assigning batches of tasks (grains) to the processors. Finding efficient ways to parallelize the parameter estimation was much more difficult than in the MLStep and the PStep of the subsequent tree reconstruction analysis, where we have to execute a large number of independent tasks with long durations relative to the communication overhead [39].

Two characteristics of the targeted sequential part determine the achievable granularity of the parallelization, namely, the number of independent tasks and the runtime relative to the communication overhead.

The pairwise distance estimation [step (ii)] comprises $n^2 - n/2$ independent tasks for $n$ input sequences. Owing to the maximum likelihood optimization of the distances, the runtime of each estimation is long relative to communication. In contrast, the likelihood computation [step (v)] based on the parameters along the tree consists of the calculation of a large number (alignment length) of independent (column) likelihoods. The calculation time per task, however, is very short.

These facts allow the application of more elaborate scheduling algorithms like SGSS for step (ii), because the communication overhead is small when compared with the computation time. In step (v), the application of such a dynamic load balancing tool is not possible. Experiments, prior to parallelization, had shown that the communication overhead would have substantially increased the parallel runtime. This is the case even more, if Ethernet is used to communicate among the processes within the COW.

Owing to the relative short parallel regions within the sequential parameter estimation strategy and its inherent high interdependence, the performance of its parallel implementation is not expected to be optimal. Nevertheless, we have to consider that parameter estimation is a minor part of the tree reconstruction procedure, which has been shown to scale well even for large numbers of processors [12, 29]. This makes the results even more valuable, as it reduces substantially the role of the parameter estimation as a bottleneck to the whole analysis.

Considering the positive results in speedup improvement and runtime reduction presented in Figures 15.12 and 15.10, which have been achieved by the additional parallelization of the likelihood computation, makes the effort worthwhile. Moreover, these improvements are very promising, as they have been obtained on COWs, where the parallel performance is hampered by the high communication overhead determined by the high latencies on their relatively slow interconnections. Application on SMP and MPP platforms produces even better results.

Future improvements of the parallel parameter estimation could be achieved approaching the computation using exact likelihood estimation. Unfortunately, the necessary optimization of the tree branch lengths by means of the ML principle is done in a sequential manner. However, this would surely increase the quality of the parallel parameter estimation.

## ACKNOWLEDGMENTS

The authors wish to thank the iBIOS group at DKFZ, Heidelberg, for access to the heterogeneous COW. The Zampano Cluster of the ZAM/NIC at FZ Jülich, Germany, has been used to study the homogeneous COW. E.P. was supported by the *Gaststudentenprogramm 2002* of NIC/ZAM. Support by the *Deutsche Forschungsgemeinschaft* is gratefully acknowledged. The TREE-PUZZLE program is available free of charge at `http://www.tree-puzzle.de`.

## REFERENCES

1. J. Felsenstein, The number of evolutionary trees, *Syst. Zool.*, 27, 27–33 (1978).
2. H. Bodlaender, M. Fellows, and T. Warnow, Two Strikes Against Perfect Phylogeny, in *Proceedings of the 19th International Colloquium on Automata, Language, and Programming (ICALP 1992)*, Vol. 623 of *Lecture Notes in Computer Science*, Springer, New York, 1992, pp. 273–283.
3. W. H. E. Day and D. Sankoff, Computational complexity of inferring phylogenies by compatibility, *Syst. Zool.*, 35, 224–229 (1986).
4. L. R. Foulds and R. L. Graham, The Steiner problem in phylogeny is NP-complete, *Adv. Appl. Math.*, 3, 43–49 (1982).
5. A. D. Baxevanis, D. B. Davison, R. D. M. Page, G. Stormo, and L. Stein, Eds., *Current Protocols in Bioinformatics*, Wiley and Sons, New York, USA, 2002.
6. J. Felsenstein, *Infering Phylogenies*, Sinauer Associates, Sunderland, Massachusetts, 2004.
7. D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis, Phylogeny Reconstruction, in D. M. Hillis, C. Moritz, and B. K. Mable, Eds., *Molecular Systematics*, Sinauer Associates, Sunderland, Massachusetts, 2nd ed., 1996, pp. 407–514.
8. J. Felsenstein, Evolutionary trees from DNA sequences: A maximum likelihood approach, *J. Mol. Evol.*, 17, 368–376 (1981).
9. G. J. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek, fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood, *Comput. Appl. Biosci.*, 10, 41–48 (1994).
10. O. Trelles, On the parallelisation of bioinformatics applications, *Brief. Bioinform.*, 2, 181–194 (2001).
11. G. M. Amdahl, Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, in *American Federation of Information Processing Societies Conference Proceedings: Spring Joint Computing Conference (AFIPS 1967)*, Vol. 30, Afips Press, Reston, Va, 1967, pp. 483–485.
12. H. A. Schmidt, K. Strimmer, M. Vingron, and A. von Haeseler, TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing, *Bioinformatics*, 18, 502–504 (2002).
13. K. Strimmer and A. von Haeseler, Quartet puzzling: A quartet maximum–likelihood method for reconstructing tree topologies, *Mol. Biol. Evol.*, 13, 964–969 (1996).
14. J. Sullivan, K. E. Holsinger, and C. Simon, The effect of topology on estimates of among-site rate variation. *J. Mol. Evol.*, 42, 308–312 (1996).
15. Z. Yang, Maximum-likelihood models for combined analyses of multiple sequence data, *J. Mol. Evol.*, 42, 587–596 (1996).
16. W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI: The Complete Reference. The MPI Extensions*, 2nd ed., Vol. 2, The MIT Press, Cambridge, Massachusetts, 1998.
17. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference — The MPI Core*, Vol. 1, 2nd ed., The MIT Press, Cambridge, Massachusetts, 1998.
18. S. Tavaré, Some probabilistic and statistical problems on the analysis of DNA sequences, *Lec. Math. Life Sci.*, 17, 57–86 (1986).

19. A. Zharkikh, Estimation of evolutionary distances between nucleotide sequences, *J. Mol. Evol.*, 39, 315–329 (1994).
20. Z. Yang, Estimating the pattern of nucleotide substitution, *J. Mol. Evol.*, 39, 105–111 (1994).
21. M. Hasegawa, H. Kishino, and T.-A. Yano, Dating of the human–ape splitting by a molecular clock of mitochondrial DNA, *J. Mol. Evol.*, 22, 160–174 (1985).
22. T. H. Jukes and C. R. Cantor, Evolution of protein molecules, in *Mammalian Protein Metabolism*, H. N. Munro, Ed., Vol. 3, Academic Press, New York, 1969, pp. 21–123.
23. X. Gu and W.-H. Li, A general additive distance with time-reversibility and rate variation among nucleotide sites, *Proc. Natl. Acad. Sci. USA*, 93, 4671–4676 (1996).
24. E. Baake and A. von Haeseler, Distance measures in terms of substitution processes. *Theor. Popul. Biol.*, 55, 166–175 (1999).
25. T. Uzzel and K. W. Corbin, Fitting discrete probability distributions to evolutionary event, *Science*, 172, 1089–1096 (1971).
26. J. Wakeley, Substitution rate variation among sites in hypervariable region 1 of human mitochondrial DNA, *J. Mol. Evol.*, 37, 613–623 (1993).
27. K. Strimmer, N. Goldman, and A. von Haeseler Bayesian probabilities and quartet puzzling, *Mol. Biol. Evol.*, 14, 210–213 (1997).
28. T. Margush and F. R. McMorris, Consensus n-trees, *Bull. Math. Biol.*, 43, 239–244 (1981).
29. H. A. Schmidt, E. Petzold, M. Vingron, and A. von Haeseler, Molecular phylogenetics: Parallelized parameter estimation and quartet puzzling, *J. Parallel Distrib. Comput.*, 63, 719–727 (2003).
30. N. Saitou and M. Nei, The neighbor–joining method: A new method for reconstructing phylogenetic trees, *Mol. Biol. Evol.*, 4, 406–425 (1987).
31. J. Adachi and M. Hasegawa, *MOLPHY Version 2.3 — Programs for Molecular Phylogenetics Based on Maximum Likelihood*, Vol. 28 of *Computer Science Monographs*, Institute of Statistical Mathematics, Minato-ku, Tokyo, 1996.
32. A. Rambaut and N. C. Grassly, Seq-Gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees, *Comput. Appl. Biosci.*, 13, 235–238 (1997).
33. J. Wuyts, G. Perrière, and Y. van de Peer, The European ribosomal RNA database, *Nucl. Acids Res.*, 32, D101–D103 (2004).
34. L. Duret, D. Mouchiroud, and M. Gouy, HOVERGEN, a database of homologous vertebrate genes, *Nucl. Acids Res.*, 22, 2360–2365 (1994).
35. T. Hagerup, Allocating independent tasks to parallel processors: An experimental study, *J. Parall. Distrib. Comput.*, 47, 185–197 (1997).
36. C. D. Polychronopoulos and D. J. Kuck, Guided self-scheduling: A practical scheduling scheme for parallel supercomputers, *IEEE Trans. Comput.*, 36, 1425–1439 (1987).
37. S. E. Goodman and S. T. Hedetniemi, *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York, 1977.
38. M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, New York, 2004.
39. H. A. Schmidt, *Phylogenetic Trees from Large Datasets*, Ph.D. thesis, Universität Düsseldorf, 2003.