



Ruprecht-Karls Universität Heidelberg
Institut für Informatik
Arbeitsgruppe Parallele und Verteilte Systeme

Bachelor-Arbeit

Implementierung einer Parallelen Ausgabe für das
Stammbaumprogramm MrBayes

Name: Amir Rais Zeervi
Matrikelnummer: 2360844
Betreuer: Dipl.-Inform. Hipolito Vasquez Lucas
Professor: Prof. Dr. Thomas Ludwig
Abgabe Datum: 31.10.2005

Ich erkläre hiermit, dass ich die vorliegende Bachelor-Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Heidelberg, den 31. Oktober 2005

.....

Danksagung

Ich möchte mich bei meinem Betreuer Herrn Dipl.-Inform. Hipolito Vasquez Lucas für seine Hilfe und Verbesserungsvorschläge bedanken.

Abstract

Many scientific applications need parallel I/O. MrBayes 3.1 is a program for the Bayesian estimation of phylogeny. Like many other applications, it is implemented with Message Passing Interface (MPI) and uses the concept of sequential I/O to write data into several output files. In the current version of MrBayes 3.1 the processes send their data to a master process which writes the received data into the files. The main issue of this work is to change the non-parallel output of MrBayes 3.1 with MPI-IO interface to parallel I/O, so that the sampled parameters of an analysis could be written to a single file. In order to increase performance different file mappings are possible. The most suitable mapping for the output data is proposed in this work. It is shown how to partition a single file with different file views and how derived data types can be used to write complex data structures.

Inhaltsverzeichnis

Inhaltsverzeichnis	5
1 Einleitung	7
1.1 Motivation	7
1.2 Einführung in MrBayes 3.1	7
1.3 Durchführen einer Analyse	8
1.3.1 Einlesen von Daten	8
1.3.2 Festlegen eines Evolutionsmodells	9
1.3.3 Durchführen einer Analyse	9
1.3.4 Die Standardausgabe	10
1.4 Aufgabenstellung der Arbeit	13
2 Analyse	14
2.1 Existierende Funktionen	14
2.1.1 DoMcmc	14
2.1.2 RunChain	16
2.1.3 GetChainIds	16
2.1.4 PreparePrintFiles	17
2.1.5 PrintStatesToFiles	18
2.1.6 PrintStates	18
2.1.7 AddToPrintString	18
2.1.8 GetTreeFromIndex	19
2.1.9 PrintTree	19
2.1.10 MrBayesPrint	19
3 Implementierung	20
3.1 Partitionierung der parallelen Datei	20
3.2 Einzelheiten zur Implementierung	21
3.2.1 Zuordnen von Analysen	21
3.2.2 Ermitteln von Hauptspeichergöße eines Cluster-Knotens	21
3.2.3 Kleinsten Hauptspeicher den Prozessen mitteilen	21
3.2.4 Ermitteln von Hauptspeicherbedarf einer Analyse	22
3.2.5 Typ eines Zyklus	24
3.2.6 Reservieren von Puffern	24
3.2.7 Kommunikation zwischen Prozessen	24
3.2.8 Ermittlung von Durchläufen	25

<i>INHALTSVERZEICHNIS</i>	6
3.2.9 Erstellen der parallelen Datei	26
3.2.10 Ausgabe in die parallele Datei	26
4 Systemumgebung und Tools	28
5 Zusammenfassung	29
6 Ausblick	30
Literaturverzeichnis	31

Kapitel 1

Einleitung

1.1 Motivation

Parallele Eingabe/Ausgabe wird immer wichtiger bei Anwendungen, die viele Daten erzeugen. Viele wissenschaftliche Anwendungen werden heutzutage mit dem Message Passing Interface (MPI) [02] parallelisiert, so dass sie auf einem Clustersystem ausgeführt werden können. Dabei senden bzw. empfangen Prozesse Nachrichten untereinander über das Netzwerk, um eine Aufgabe koordiniert parallel durchzuführen. Diese Möglichkeit der Parallelisierung hat dazu geführt, dass komplexe Aufgaben schneller bewältigt werden konnten. Öfters wurden sequentielle Programme mit MPI-1 so parallelisiert, dass zwar die Berechnung parallel geschah, das Schreiben der Ausgabedaten in die Dateien jedoch immer nur von einem Prozess übernommen wurde. Das war bisher das Standard-Konzept, was bei MPI-1 Programmen eingesetzt wurde. Obwohl diese Programme parallel liefen, blieb ein Teil des Programms weiterhin sequentiell, nämlich die Eingabe/Ausgabe.

In dieser Arbeit sollen die Funktionen von MPI-2 [GTL99],[Sei] auf das Phylogenie-Programm MrBayes 3.1 angewendet werden, so dass die bisherige Ausgabe von MrBayes 3.1 durch eine Gruppe von MPI-Prozessen in eine einzige Datei geschrieben werden kann.

1.2 Einführung in MrBayes 3.1

In diesem Abschnitt wird eine Einführung in das Programm MrBayes 3.1 gegeben. MrBayes 3.1 ist eines der häufig verwendeten Computerprogramme zur phylogenetischen Analyse. Es ist ein Open-source-Programm und im Internet kostenlos verfügbar [01]. Das Programm verwendet das Markov-Chain-Monte-Carlo Verfahren, um die Phylogenie-Bäume zu berechnen [Pri].

MrBayes 3.1 kann sowohl auf ein Einzelplatzsystem als auch auf ein Unix bzw. Linux-Cluster ausgeführt werden. Die Parallelversion ist mit Message Passing Interface (MPI) implementiert und kann auf ein Cluster ausgeführt werden. Die Parallelversion wird auf dem Cluster mit dem Befehl `mpiexec` wie folgt verwendet:

```
mpiexec -np #Prozesse ./mb
```

MrBayes 3.1 hat keine grafische Oberfläche (GUI), sondern wird über eine eigene Befehlszeile gesteuert. Das Programm bietet eine Vielzahl von Befehlen, jedoch soll hier nur auf einige dieser Befehle eingegangen werden. Details über die Befehle können im Handbuch [FRvdM05] nachgelesen werden. Der wichtigste Befehl in MrBayes 3.1 ist `help`, damit lassen sich alle in MrBayes 3.1 verfügbaren Befehle anzeigen. Um beispielsweise Hilfe über den Befehl `mcmc` anzuzeigen, gibt man an der Befehlszeile `help mcmc` ein. MrBayes 3.1 enthält mehrere Beispieldateien, die man für eigene Analyse als Vorlage verwenden kann. Für diese Arbeit wird die Datei `primates.nex` verwendet.

1.3 Durchführen einer Analyse

In diesem Abschnitt wird erläutert, wie in MrBayes 3.1 eine einfache Analyse durchgeführt wird. Dazu bedarf es drei Schritten:

1. Einlesen von Daten
2. Festlegen eines Evolutionsmodells
3. Durchführen einer Analyse

1.3.1 Einlesen von Daten

Um eine Analyse durchzuführen müssen in MrBayes 3.1 Daten eingelesen werden. Dies geschieht mit dem Befehl `exe` bzw. `execute`. Die Datei `primates.nex` wird beispielsweise über die Befehlszeile wie folgt eingelesen:

```
exe primates.nex
```

oder

```
execute primates.nex
```

MrBayes 3.1 verwendet für die input files das Nexus-Format [03]. Eine Nexus-Datei ist eine einfache Textdatei und enthält Sequenzen der Aminosäure, mitochondrische DNA-Sequenzen bzw. RNA-Sequenzen. Wie eine solche Datei aussieht, wird anhand der Datei `primates.nex` (siehe Abbildung 1.1) erklärt.

Die erste Zeile einer Nexus-Datei enthält das Schlüsselwort `#NEXUS`. Damit wird gekennzeichnet, dass es sich hier um eine Nexus-Datei handelt. Der Datenblock beginnt mit `begin data`; und endet mit `end`; Nach `begin data`; folgt in der nächsten Zeile die Dimension der Datenmatrix, hier wird durch `ntax` die Anzahl der Organismen angegeben, die in dieser Datenmatrix vorkommen bzw. für die eine Analyse durchgeführt werden soll. Anschließend wird mit `nchar` die Länge einer Sequenz angegeben, gefolgt von dem Befehl `format`, welcher definiert, um was für eine Art von Datenmatrix es sich hierbei handelt. Je nach Daten kann dem Parameter `datatype` DNA, RNA oder Protein zugeordnet werden. Der Befehl `format` enthält ausserdem die Parameter `gap` und `interleave`. Die “Gaps” sind sog. “Leerstellen”, um alle

```

#NEXUS
begin data;
dimensions ntax=12 nchar=898;
format datatype=dna interleave=no gap=-;
matrix
Tarsius_syrichta      AAGTTTCATTGGAGCCACCACTCTTATAATTGCCCATGGCCTCACCTCCTCCC
Lemur_catta           AAGCTTCATAGGAGCAACCATTCCTAATAATCGCACATGGCCTTACATCATCCA
Homo_sapiens          AAGCTTCACCGGCGCAGTCATTCTCATAATCGCCCACGGGCTTACATCCTCAT
Pan                   AAGCTTCACCGGCGCAATTATCCTCATAATCGCCCACGGACTTACATCCTCAT
Gorilla               AAGCTTCACCGGCGCAGTTGTTCTTATAATTGCCACGGACTTACATCATCAT
Pongo                 AAGCTTCACCGGCGCAACCACCTCATGATTGCCCATGGACTCACATCCTCCC
Hylobates             AAGCTTTACAGGTGCAACCGTCTCATAATCGCCCACGGACTAACCTCTTCCC
Macaca_fuscata        AAGCTTTTCCGGCGCAACCATCCTTATGATCGCTCACGGACTCACCTCTTCCA
M_mulatta             AAGCTTTTCTGGCGCAACCATCCTCATGATTGCTCACGGACTCACCTCTTCCA
M_fascicularis        AAGCTTCTCCGGCGCAACCACCTTATAATCGCCCACGGGCTCACCTCTTCCA
M_sylvanus            AAGCTTCTCCGGTGCAACTATCCTTAGTTGCCCATGGACTCACCTCTTCCA
Saimiri_sciureus      AAGCTTCACCGGCGCAATGATCCTAATAATCGCTCACGGGTTTACTTCGTCTA
;
end;

```

Abbildung 1.1: primates.nex

Gen-Sequenzen auf die gleiche Länge zu bringen. Die nächste Zeile enthält das Schlüsselwort `matrix`, danach folgen die einzelne Sequenzen mit ihren Namen.

1.3.2 Festlegen eines Evolutionsmodells

Vor jeder Analyse wird in MrBayes 3.1 ein Evolutionsmodell definiert. So wie andere Befehle, wird auch das Evolutionsmodell über die Befehlszeile eingegeben. Dafür gibt es zwei Befehle, `lset` und `prset`.

1.3.3 Durchführen einer Analyse

Eine Analyse wird mit dem Befehl `mcmc` gestartet und an der Befehlszeile wie folgt eingegeben:

```
mcmc <Parameter> = <Wert> ... <Parameter> = <Wert>
```

Beispiel:

```
mcmc ngen=10000 samplefreq=10 nruns=3 nchains=4
```

Mit `help mcmc` lassen sich alle Parameter des Befehls `mcmc` auflisten. Ausser `mcmc` gibt es in MrBayes 3.1 noch den Befehl `mcmcp`, dieser wird verwendet, um die Parameter einer Analyse zu ändern, ohne die Analyse zu starten.

MrBayes 3.1 ist standardmäßig so eingestellt, dass es zwei vollständig unabhängige Analysen durchführt. Dadurch wird sichergestellt, dass die berechneten A-posteriori-Wahrscheinlichkeiten für die Bäume zuverlässig sind. Die Anzahl der Analysen kann mit dem Parameter `nruns` geändert werden.

Dem Parameter `ngen` wird die Anzahl der Schritte (Generationen) zugewiesen, mit der die Analyse laufen soll. Der Parameter `printfreq` legt fest, nach wievielen Schritten die Werte auf dem Bildschirm angezeigt werden sollen. Wird dem Parameter der Wert 100 zugewiesen, so wird jeder 100. Schritt der Stammbaumanalyse auf dem Bildschirm angezeigt. Mit `samplefreq=100` wird jeder 100. Phylogenie-Baum gesammelt und in die `.t` Dateien geschrieben. Der Parameter `nchains` legt fest, wieviele Suchläufe (Chains) parallel ausgeführt werden sollen.

Häufig verwendete Befehle in MrBayes 3.1 sind:

```
help    = Zeigt alle MrBayes-Befehle
mcmc    = Startet eine Analyse
mcmcp   = Ändert die Parameter einer Analyse
```

Neben der Möglichkeit MrBayes 3.1 über die Befehlszeile zu steuern, gibt es auch die Möglichkeit alle Befehle in eine einzige Batch-Datei (Dateiname.nex) einzugeben. Eine solche Datei sieht beispielsweise wie folgt aus:

```
begin mrbayes;
set autoclose=yes nowarn=yes;
execute primates.nex;
lset nst=6 rates=gamma;
mcmc nruns=1 ngen=10000 samplefreq=10;
end;
```

Der Block `begin mrbayes;` enthält alle Befehle, die normalerweise über die Befehlszeile eingegeben werden. Jede Zeile in der Batch-Datei wird durch ein Semikolon beendet.

Die obige Batch-Datei liest zunächst die Datei `primates.nex` ein, definiert das Evolutionsmodell und führt anschließend die Analyse durch. Um beispielsweise die Batch-Datei `batch.nex` beim Öffnen von MrBayes 3.1 sofort auszuführen, wird der Befehl `mpiexec` wie folgt eingegeben:

```
mpiexec -np 4 ./mb ./batch.nex
```

1.3.4 Die Standardausgabe

MrBayes 3.1 erstellt für jede unabhängige Analyse eine `.p`-, `.t`- und `.cal`-Datei. Außerdem wird eine Datei mit der Endung `.mcmc` erstellt. In der `.p`-Datei werden die A-posteriori-Wahrscheinlichkeiten sowie andere Ergebnisse einer Analyse gespeichert. Die Anzahl der Spalten dieser Datei kann sich je nach Modell ändern, die einzelnen Spalten sind durch Tabulatoren getrennt. Für `ngen=10` und `samplefreq=2` ist die `.p`-Datei in Abbildung 1.2 dargestellt.

[ID: 3799909121]							
Gen	LnL	TL	pi(A)	pi(C)	pi(G)	pi(T)	
1	-8192.395		2.100	0.250000		0.250000	0.250000 0.250000
2	-7985.111		2.100	0.250000		0.250000	0.250000 0.250000
4	-7860.976		2.134	0.250000		0.250000	0.250000 0.250000
6	-7592.893		2.083	0.250000		0.250000	0.250000 0.250000
8	-7564.997		2.017	0.250000		0.250000	0.250000 0.250000
10	-7542.346		1.973	0.250000		0.250000	0.250000 0.250000

Abbildung 1.2: .p-Datei

Die erste Zeile der .p-Datei enthält eine Kennung ID. Diese Kennung wird für jede Analyse nach dem Zufallsprinzip generiert und wird verwendet, um eine Analyse von einer anderen zu unterscheiden. Wenn eine Analyse mit mehreren Durchläufen *nruns* gestartet wird, so wird dieselbe ID in allen Dateien gespeichert. Die nächste Zeile enthält die Überschriften für die einzelnen Spalten, danach werden zeilenweise die gesammelten Ergebnisse in die Dateien geschrieben. In der ersten Spalte *Gen* werden die Schritte (Generationen) gespeichert, die gesammelt werden. Die zweite Spalte *LnL* enthält die Wahrscheinlichkeiten für die “cold chains”, welche die höchsten A-posteriori-Wahrscheinlichkeiten darstellen, die während einer Analyse in der Wahrscheinlichkeitsdichte liegen. In die Spalte *TL* werden die Längen der Bäume gespeichert. Diese sind die Summen aller Äste eines Baums. Danach folgen die Parameter des Modells, diese können je nach Modell unterschiedlich aussehen.

Die .t-Datei enthält alle Phylogenie-Bäume, die während einer Analyse gesammelt werden. Die .t-Dateien sind Nexus-Dateien und beginnen mit *#NEXUS*. Eine solche Datei ist in Abbildung 1.3 dargestellt.

```
#NEXUS
[ID: 3799909121]
begin trees;
  translate
    1 Tarsius_syrichta,
    2 Lemur_catta,
    3 Homo_sapiens,
    4 Pan,
    5 Gorilla,
    6 Pongo,
    7 Hylobates,
    8 Macaca_fuscata,
    9 M_mulatta,
    10 M_fascicularis,
    11 M_sylvanus,
    12 Saimiri_sciureus;
  tree rep.1 = ((2:0.100000, ((7:0.100000, (8:0.100000, 10:0.100000):
0.100000):0.100000, 5:0.100000):0.100000, (4:0.100000,
11:0.100000):0.100000):0.100000):0.100000, (12:0.100000,
((6:0.100000, 9:0.100000):0.100000, 3:0.100000):0.100000)
:0.100000, 1:0.100000);
  tree rep.2 = ...
  tree rep.4 = ...
  tree rep.6 = ...
  tree rep.8 = ...
  tree rep.10 = ...
end;
```

Abbildung 1.3: .t-Datei

Die zweite Zeile der `.t`-Datei enthält die Kennung `ID` der Analyse. Das ist dieselbe Kennung, die auch in die `.p`-Dateien geschrieben wurde. Alle Bäume, die während einer Analyse gesammelt werden, stehen zwischen `begin trees;` und `end;`. In der Abbildung 1.3 ist der erste Baum (tree rep. 1) komplett dargestellt. Die anderen Bäume sind genauso aufgebaut.

Die Anzahl der `.t` und `.p`-Dateien wird durch den Parameter `nruns` bestimmt. Wird beispielsweise dem Parameter `nruns` der Wert 3 zugewiesen, so werden drei `.t`- und drei `.p`-Dateien erstellt, jeweils eine für jede Analyse.

In der bisherigen Version von MrBayes 3.1, ohne die MPI-2 Implementierung, werden alle Dateien nur von einem Prozess (Prozess 0) sequentiell geschrieben. Jeder Prozess, der eine “cold chain” besitzt, sendet sein Ergebnis an Prozess 0. Dieser schreibt dann zuerst seinen eigenen Wert in die Ausgabedateien und dann die von anderen Prozessen.

MrBayes 3.1 prüft in der Funktion `PrintStatesToFiles(int curGen)` für jeden berechneten Phylogenie-Baum, ob es ein “calibrated tree” ist. Wenn ein Phylogenie-Baum diese Bedingung erfüllt, so wird dieser von der Funktion `PrintCalTree(curGen, tree)` in die `.cal`-Datei gespeichert.

Wie zuvor bereits beschrieben, wird auch eine Datei mit der Endung `.mcmc` erstellt. Während einer Analyse werden in dieser Datei Standard-Abweichungen zwischen Partitionen von einem Baum gespeichert. Dafür wird die Funktion `AddTreeToPartitionCounters(Tree *tree, int treeId, int runId)` aufgerufen. Ausserdem enthält diese Datei Statistiken über den Austausch (Swap), der zwischen verschiedenen Suchläufen (Chains) stattgefunden hat. Die Werte in dieser Datei werden mit der Funktion `PrintMCMCDiagnosticsToFile(int curGen)` in die Datei geschrieben. Eine `.mcmc`-Datei ist in der Abbildung 1.4 dargestellt.

[LEGEND:						
Gen			--	Generation		
<name>(1){2}			--	Acceptance rate of move <name> changing parameter 1 in run 2		
Swap(1-2){3}			--	Acceptance rate of swaps between chains 1 and 2 in run 3		
StdDev(s)			--	Average standard deviation of split frequencies		
]						
Gen	Dirichlet(1){1}	LOCAL(2){1}	eTBR(2){1}	Swap(1-2){1}	Swap(1-3){1}	
10	N/A	0.000000	0.500000	1.000000	0.000000	0.5000

Abbildung 1.4: `.mcmc`-Datei

Die `.mcmc`-Datei wird erstellt, wenn in der Analyse dem Parameter `Mcmcdiagn` der Wert `YES` zugewiesen ist.

1.4 Aufgabenstellung der Arbeit

Die Aufgabenstellung dieser Arbeit besteht darin, die MPI-Version von MrBayes 3.1 mit MPI-2 so anzupassen, dass die Ausgabedaten für die .t- und .p-Dateien in eine einzige Datei gespeichert werden. Diese gemeinsame Datei soll so partitioniert werden, dass sie durch möglichst viele MPI-Prozesse parallel geschrieben werden kann.

Kapitel 2

Analyse

2.1 Existierende Funktionen

In diesem Kapitel wird beschrieben, welche Funktionen in MrBayes für die Ausgabe relevant sind. Anhand des Schaubilds (siehe Abbildung 2.1) wird der Ablauf für die Ausgabe ausführlich beschrieben. Alle Funktionen, die im Schaubild vorkommen, sind in der Datei `mcmc.c` implementiert.

Um den Ablauf zu verdeutlichen, werden verschiedene Objekte (Arc-Box, Viereck) verwendet. Die Funktionen werden in einem Kästchen (Arc-Box) dargestellt; ein Viereck stellt Ausgabedateien dar. Durchgehende Pfeile dienen dazu, Funktionsaufrufe darzustellen. Wenn beispielsweise ein durchgehender Pfeil von der Funktion A zu der Funktion B geht, so bedeutet das, dass die Funktion B von der Funktion A aufgerufen wird. Ein Ergebnis wird in einer Ellipse dargestellt. Ausserdem werden gestrichelte Pfeile verwendet, wenn das Ergebnis entweder von einer anderen Funktion weiterverarbeitet oder in die Ausgabedateien geschrieben wird. Die Zahlen in Kreisen sind dazu da, den Programmablauf besser zu verdeutlichen. Die gestrichelt markierten Bereiche auf der linken und rechten Seite in Bild 2.1 stellen dar, wie die Zeichenkette `prinString` für die `.p-` und `.t-` Datei erstellt wird.

2.1.1 DoMcmc

Die Funktion `DoMcmc(void)` ist die Hauptfunktion für eine Analyse. Sie ruft mehrere Funktionen auf, bevor die Funktion `RunChain(long int *seed)` für die Stammbaumsuche aufgerufen wird. Hier werden nur einige dieser Funktionen erläutert, die für die Ausgabe relevant sind. Da in MrBayes 3.1 vor jeder Analyse eine Datenmatrix eingelesen werden muss, prüft `DoMcmc(void)` den Wert der Variable `defMatrix`, um herauszufinden, ob eine Datenmatrix eingelesen wurde. Wenn `defMatrix == NO` ist, so wird der Benutzer mit der Fehlermeldung ‘‘A character matrix must be defined first’’ informiert und er muss eine Datendatei (`*.nex`) über die Befehlszeile von MrBayes 3.1

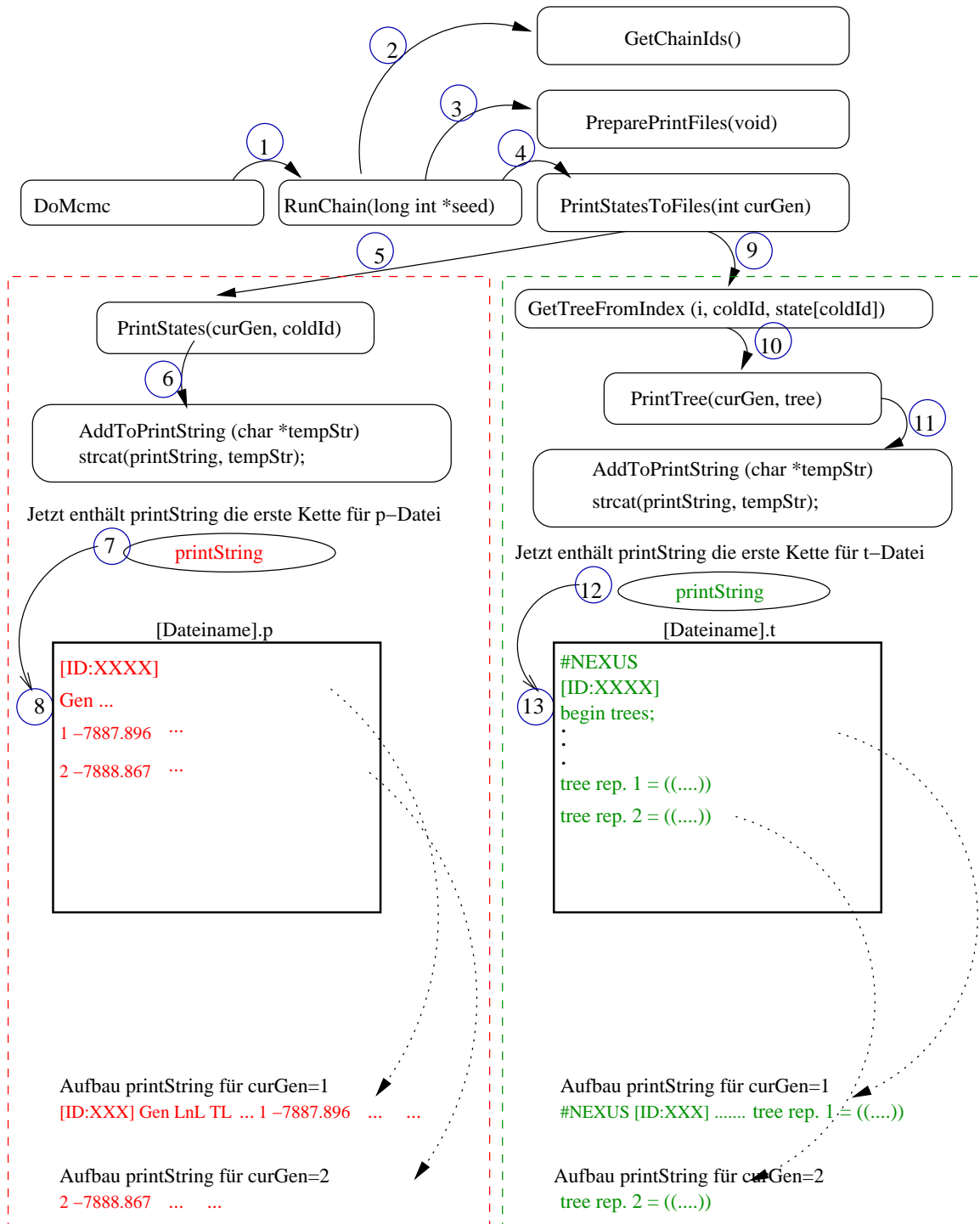


Abbildung 2.1: Bisherige Ausgabe

eingeben. Um für eine Analyse eine Kennung (ID) zu bekommen, ruft `DoMcmc(void)` die Funktion `GetStamp()` auf. Diese ID wird für jede Analyse berechnet und in die Dateien geschrieben. Dadurch kann später festgestellt werden, welche Daten zu einer

bestimmten Analyse gehören. Schließlich wird von ihr die Funktion `RunChain(&seed)` aufgerufen, um die Stammbaumsuche durchzuführen. Der Parameter `seed` wird am Anfang jeder Analyse zufällig (randomly) generiert und wird verwendet, um die Suchläufe untereinander zu unterscheiden.

Der Parameter `seed` wird für die MPI-Version wie folgt berechnet:

```
seed = chainParams.chainSeed + (proc_id + 1);
```

Dabei enthält `chainParams.chainSeed` den Wert, der dem Befehl `mcmc` als Parameter übergeben wird. Die Variable `proc_id` bezeichnet die Nummer (Rank) eines MPI-Prozesses.

2.1.2 RunChain

Die Funktion `RunChain(long int *seed)` führt die Stammbaumsuche durch und wird von der Funktion `DoMcmc(void)` aufgerufen.

Damit alle MPI-Prozesse die Suchläufe (Chains) bekommen, ruft sie die Funktion `GetChainIds(void)` auf. Auf diese Weise bekommt jeder Prozess die IDs seiner Suchläufe (Chains) zugewiesen und speichert diese in seinem `chainId[]`-Array. Die Funktionsweise von `GetChainIds(void)` wird unten ausführlich beschrieben.

`RunChain(long int *seed)` ruft dann die Funktion `PreparePrintFiles(void)` auf, um die Ausgabedateien für die Funktion `PrintStatesToFiles(int curGen)` vorzubereiten. Da MPI-Prozesse für jeden Schritt der Stammbaumanalyse den Zustand ihrer Suchläufe (Chains) untereinander tauschen, wird die Funktion `AttemptSwap(swapA, swapB, seed)` aufgerufen. Diese Funktion bekommt als Parameter die Ranks (`swapA` und `swapB`) von zwei Prozessen übergeben, die untereinander einen Austausch ihrer Suchläufe (Chains) durchführen. Nach dem Austausch ändert sich das `chainId[]`-Array. Ausserdem wird der Funktion der zufällig generierte Parameter `seed` übergeben.

2.1.3 GetChainIds

Die Funktion `GetChainIds(void)` wird von der Funktion `RunChain(long int *seed)` aufgerufen und füllt das `chainId[]`-Array von jedem Prozess mit `GlobalChains`. Die Anzahl der `GlobalChains` wird durch die Parameter `nchains` und `nruns` bestimmt. Wenn beispielsweise in der Analyse für `nchains` der Wert 4 und `nruns=3` definiert wurde, so gibt es insgesamt 12 `GlobalChains` (`nruns * nchains`). Diese Chains werden unter allen MPI-Prozessen gleichmäßig verteilt. Chains, die übrig bleiben, werden beginnend mit dem Prozess mit dem kleinsten Rank auf die Prozesse verteilt. Um herauszufinden, ob `GlobalChains` übrig bleiben wird der Restwert `remainder` wie folgt berechnet:

```
remainder = numGlobalChains % num_procs;
```

Nachdem die `GlobalChains` unter die Prozesse verteilt sind, sieht beispielsweise das `chainId[]`-Array für `num_procs=3`, `nchains=4` und `nruns=2` von jedem Prozess bei der Initialisierung wie folgt aus:

Prozess 0

0	1	2
---	---	---

Prozess 1

3	4	5
---	---	---

Prozess 2

6	7
---	---

Im Laufe der Analyse tauschen Prozesse die Zustände ihrer Chains untereinander aus, dadurch ändern sich auch die IDs, die in den `chainId[]`-Array gespeichert sind. Beispielsweise könnten die IDs nach einem Durchlauf wie folgt aussehen:

Prozess 0

4	2	0
---	---	---

Prozess 1

1	3	7
---	---	---

Prozess 2

6	5
---	---

Diese IDs werden später in der Funktion `PrintStatesToFiles` benötigt, um herauszufinden, welcher Prozess für das Schreiben einer Analyse zuständig ist.

2.1.4 PreparePrintFiles

Die Funktion `PreparePrintFiles(void)` wird von der Funktion `RunChain (long int *seed)` aufgerufen, um die Dateien `.t`, `.p` und `.mcmc` für die Ausgabe vorzubereiten. Sie reserviert Speicher für die Dateizeiger der Ausgabedateien. Außerdem wird in ihr der jeweilige Dateiname mit `sprintf (fileName, "%s.Dateiendung", localFileName)` in die Variable `fileName` gespeichert. Anschließend wird die Datei von Prozess 0 mit der Funktion `FILE *OpenNewMBPrintFile (char *fileName)` geöffnet. Die Funktion `FILE *OpenNewMBPrintFile (char *fileName)` liefert als Rückgabewert einen file pointer auf die Datei, die geöffnet werden soll. In dieser Funktion wird zunächst geprüft, ob die Datei `fileName` bereits vorhanden ist. Wenn sie bereits existiert, so wird der Benutzer darauf hingewiesen und er hat die Möglichkeit, entweder diese Datei zu löschen oder Daten am Ende dieser Datei anzuhängen. Dazu wird er aufgefordert die jeweilige Option zu wählen. Wenn die Datei überschrieben werden soll, wird die Funktion `OpenTextFileW(fileName)` aufgerufen. Wenn die neue Analyse in dieselbe Datei gespeichert werden soll, so wird die Funktion `OpenTextFileA(fileName)` aufgerufen.

2.1.5 PrintStatesToFiles

Die Funktion `PrintStatesToFiles(int curGen)` ist eine der wichtigsten Funktionen in MrBayes 3.1. Sie schreibt die Phylogenie-Bäume bzw. Model-Parameter in die Ausgabedateien und wird von der Funktion `RunChain(&seed)` aufgerufen.

Nachdem die Werte für den aktuellen Schritt `curGen` berechnet sind, rufen alle MPI-Prozesse aus der Funktion `RunChain(&seed)` die Funktion `int PrintStatesToFiles(int curGen)` auf, um die Ergebnisse in die Ausgabedateien zu schreiben. Die Funktion `PrintStatesToFiles(int curGen)` wird für den ersten bzw. letzten Schritt `curGen` der Stammbaumanalyse aufgerufen oder sie wird aufgerufen, wenn ein Schritt `curGen` durch `samplefreq` ohne Rest teilbar ist.

In `PrintStatesToFiles` rufen alle Prozesse die Funktion `MPI_Barrier` auf, um sich zu synchronisieren. Sie ruft die Funktionen `PrintStates`, `GetTreeFromIndex`, `PrintTree` innerhalb einer Schleife auf. Diese Schleife läuft über alle Durchläufe (`nruns`). Da nicht definiert ist, welcher Prozess für welchen Durchlauf zuständig ist, muss jeder Prozess für den aktuellen Schritt `curGen` sein lokales `ChainId[]`-Array überprüfen, ob er die ID für diese "cold chain" besitzt. Nur wenn er die ID besitzt, darf er die Funktion `PrintStates(curGen, coldId)` aufrufen und deklariert die Variable `doesThisProcHaveId` mit `YES`. Die Funktion `PrintStates(curGen, coldId)` liefert die Zeichenkette in `printString`. Diese Zeichenkette wird an Prozess 0 geschickt. Dann wird die Funktion `Tree *GetTreeFromIndex (int index, int chain, int state)` aufgerufen. Um den Phylogenie-Baum zu erhalten, ruft `PrintStatesToFiles(int curGen)` die Funktion `PrintTree (int curGen, Tree *tree)` auf. Diese liefert den Phylogenie-Baum in die Variable `printString` zurück. Dieser Baum wird dann an Prozess 0 gesendet, um in die `.t`-Datei zu schreiben.

2.1.6 PrintStates

Die Funktion `PrintStates(int curGen, int coldId)` wird von der Funktion `PrintStatesToFiles` mit den Parametern `curGen` und `coldId` aufgerufen. Sie liefert als Ergebnis die komplette Zeichenkette `printString`, die von der Funktion `PrintStatesToFiles` in die `.p`-Datei geschrieben wird. Wird `PrintStates` mit `curGen = 1` aufgerufen, so wird der Dateikopf der `.p`-Datei zusammen mit dem Ergebnis der ersten Zeile in den `printString` gespeichert. Um die Zeichenketten zusammen in `printString` zu schreiben, wird von der Funktion `PrintStates` die Funktion `AddToPrintString` aufgerufen.

2.1.7 AddToPrintString

Die Funktion `AddToPrintString (char *tempStr)` wird verwendet, um die Zeichenkette `tempStr` an die Zeichenkette `printString` zu konkatenieren. Der Parameter

`tempStr` wird der Funktion als Zeiger vom Typ `char *` übergeben. Beispielsweise wird diese Funktion von `PrintStates` und `PrintTree` aufgerufen, um den Dateikopf zusammen mit der ersten Zeile in `printString` zu speichern.

2.1.8 GetTreeFromIndex

Die Funktion `Tree *GetTreeFromIndex (int index, int chain, int state)` wird durch `PrintStatesToFiles` aufgerufen und liefert als Rückgabewert einen Zeiger vom Typ `Tree *`. Das ist ein Baum, der während der Analyse berechnet wurde. Dieser Baum wird der Funktion `PrintTree(int curGen, Tree *tree)` als Parameter übergeben.

2.1.9 PrintTree

Die Funktion `PrintTree (int curGen, Tree *tree)` wird von der Funktion `PrintStatesToFiles` aufgerufen und liefert als Ergebnis für jeden Schritt `curGen` der Stammbaumanalyse einen Phylogenie-Baum. Dieser Baum wird ebenfalls in `printString` gespeichert. Wird `PrintTree` mit `curGen = 1` aufgerufen, so wird der Dateikopf der `.t`-Datei zusammen mit dem Ergebnis der ersten Zeile in den `printString` gespeichert. Um die Zeichenketten `tempStr` und `printString` zu konkatenieren, wird die Funktion `AddToPrintString` verwendet.

2.1.10 MrBayesPrint

In MrBayes 3.1 wird im Quellcode nicht die übliche `printf` Funktion verwendet, sondern eine eigene implementierte Funktion `MrBayesPrint`. Diese Funktion erlaubt alle Bildschirmausgaben in eine Logdatei (`log.out`) auszugeben. Dafür wird der Befehl `log` an der Befehlszeile wie folgt eingegeben:

```
log start.
```

Kapitel 3

Implementierung

3.1 Partitionierung der parallelen Datei

Um die Daten der .p- und .t-Datei parallel in eine einzige Datei zu schreiben, wird die Datei in zwei Bereiche aufgeteilt. Der eine Bereich dieser Datei enthält nur Daten der .p-Dateien, der andere Bereich nur Daten der .t-Dateien. Da die parallele Datei später von MrBayes 3.1 eingelesen werden soll, ist dies die einfachste Variante. Die Abbildung 3.1 verdeutlicht, wie die Datei parallel geschrieben wird.

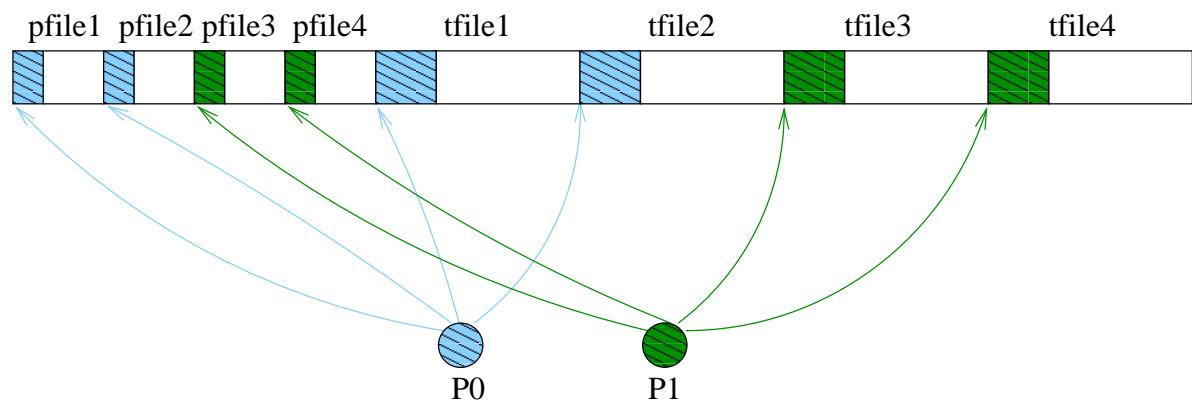


Abbildung 3.1: .pvs-Datei

Da die Datei von mehreren Prozessen parallel geschrieben wird, muss für jeden Prozess definiert werden, welche Bereiche der Datei er schreiben darf, d.h. für welche Durchläufe `nruns` er zuständig ist. Durch die Funktion `IORunAssignment(void)` werden die Durchläufe `nruns` auf die Prozesse verteilt. Wie diese Verteilung abläuft, wird im Abschnitt 3.2.1 beschrieben.

3.2 Einzelheiten zur Implementierung

3.2.1 Zuordnen von Analysen

Die Funktion `IORunAssignment(void)` verteilt die Durchläufe `nruns` auf die Prozesse. Die Verteilung geschieht ähnlich, wie bei der Funktion `GetChainIds(void)`. Zunächst werden alle Durchläufe `nruns` gleichmäßig auf die Prozesse verteilt. Wenn die Variable `remainder` größer als 0 ist, so werden die restlichen Durchläufe `nruns` beginnend mit dem Prozess 0, unter die Prozesse verteilt. Jeder Prozess speichert in sein `IORunId[]`-Array die IDs seiner Durchläufe (`nruns`). Das `IORunId[]`-Array sieht für `num_procs=3` und `nruns=8` nach der Verteilung wie folgt aus:

Prozess 0	0	1	2
Prozess 1	3	4	5
Prozess 2	6	7	

3.2.2 Ermitteln von Hauptspeichergröße eines Cluster-Knotens

Die Funktion `long MMSize(void)` ermittelt von einem Cluster-Knoten 1/4 seiner Hauptspeichergröße in Bytes. Sie wird von der Funktion `MemoryForProcs(void)` aufgerufen, um den kleinsten Hauptspeicher unter allen Cluster-Knoten zu ermitteln und diese den Prozessen mitzuteilen. Die Funktion lautet wie folgt:

```
long MMSize (void)
{
    long numofpages, sizeofpage;

    numofpages = sysconf (_SC_PHYS_PAGES);
    sizeofpage = sysconf (_SC_PAGESIZE);

    memsize = (numofpages * sizeofpage) / 4;

    return memsize;
}
```

3.2.3 Kleinsten Hauptspeicher den Prozessen mitteilen

Die Funktion `MemoryForProcs(void)` ruft die Funktion `long MMSize(void)` auf und übermittelt die kleinste Hauptspeichergröße eines Cluster-Knotens an alle Prozesse. Wie in der Abbildung 3.2 dargestellt, wird beispielsweise die Hauptspeichergröße von `node3` an alle Prozesse übermittelt.

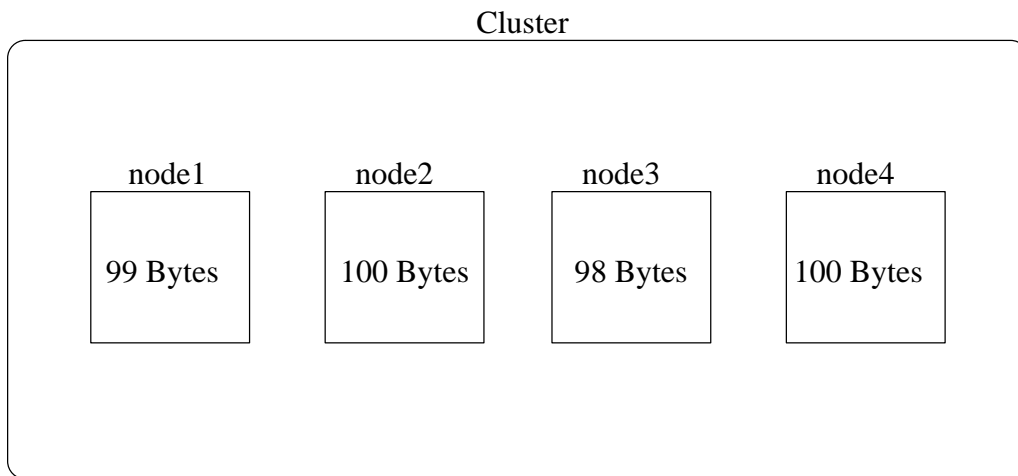


Abbildung 3.2: Kleinster Hauptspeicher

3.2.4 Ermitteln von Hauptspeicherbedarf einer Analyse

Damit alle MPI-Prozesse die .pvs-Datei parallel schreiben können, wird die exakte Größe der Daten für .p-Dateien und .t-Dateien benötigt. Diese Größe wird von der Funktion `int GetTotalOutputDataSetSize(int n)` berechnet. Sie liefert die exakte Größe der Daten für .p- und .t-Dateien in Bytes. Als Parameter wird der Funktion der aktuelle Schritt `n` der Stammbaumanalyse übergeben. Das Ergebnis wird in die Variable `totalforpandt` abgelegt.

Da die MPI-Prozesse ihre individuellen Bereiche der .pvs-Datei schreiben, muss für jeden Prozess seine Dateisicht (file view) definiert werden. Jeder Prozess sieht dann nur die Bereiche einer Datei, die er auch schreiben darf. Anhand der berechneten Größe wird für jeden Prozess mit der Funktion `MPI_File_set_view` sein `Displacement` definiert. Die Dateisicht (file view) für alle Prozesse wird in der Funktion `int PI02WriteOut(void)` definiert. Um die Größe zu berechnen, wird zunächst ermittelt, wieviele Zeilen in die .p und .t-Datei geschrieben werden. Die Anzahl der Zeilen wird in die Variable `datasamples` gespeichert. Der Wert für `datasamples` wird im Quellcode durch drei Fälle bestimmt:

```

:
:
if (((chainParams.numGen % 2) == 0) && ((chainParams.sampleFreq % 2) == 0)) ||
    (((chainParams.numGen % 2) != 0) && ((chainParams.sampleFreq % 2) != 0) &&
    (chainParams.sampleFreq != 1)))

datasamples = (chainParams.numGen / chainParams.sampleFreq) + 1;

if (((chainParams.numGen % 2) == 0) && ((chainParams.sampleFreq % 2) != 0) &&
    (chainParams.sampleFreq != 1)) ||
    (((chainParams.numGen % 2) != 0) && ((chainParams.sampleFreq % 2) == 0)))
  
```

```

datasamples = (chainParams.numGen / chainParams.sampleFreq) + 2;
if (chainParams.sampleFreq == 1)
datasamples = chainParams.numGen;
.
.

```

Dann ruft der Prozess `procWithChain` die Funktion `PrintStates(1, coldId)` mit `curGen=1` auf und erhält als Ergebnis die erste Zeile der `.p`-Datei (siehe Abbildung 3.3).

Dateikopf	1	-8204.947	2.130	0.250000	0.250000
-----------	---	-----------	-------	----------	----------	-----	-----

Abbildung 3.3: Zeile einer `.p`-Datei

Aus dieser Zeile wird die Länge der Teilzeichenkette ermittelt, diese Teilzeichenkette beginnt oben in der Abbildung 3.3 mit `-8204.947`. Das Ergebnis wird in die Variable `pStringsize` gespeichert. Der Speicherbedarf aller `.p`-Dateien lässt sich dann wie folgt berechnen:

```
memforPData = datasamples * pStringsize * chainParams.numRuns;
```

Um die erste Zeile der `.t`-Datei zu berechnen, wird von dem Prozess `procWithChain` zunächst die Funktion `GetTreeFromIndex(0, coldId, state[coldId])` aufgerufen. Diese liefert als Rückgabewert den Parameter `tree` für die Funktion `PrintTree(1, tree)`. Dann wird die Funktion `PrintTree(1, tree)` mit `curGen=1` aufgerufen und bekommt als Parameter `tree` übergeben. Die Funktion `PrintTree(1, tree)` liefert als Ergebnis die erste Zeile der `.t`-Datei (siehe Abbildung 3.4).

Dateikopf	tree rep.1 =	((((9:0.100000,4:0.100000):0.100000..
-----------	--------------	--

Abbildung 3.4: Zeile einer `.t`-Datei

Aus dieser Zeile wird die Länge der Teilzeichenkette ermittelt, diese Teilzeichenkette beginnt oben in der Abbildung 3.4 mit `((((9:0`. Das Ergebnis wird in die Variable `tStringsize` gespeichert. Der Speicherbedarf für die Zeilen aller `.t`-Dateien lässt sich dann wie folgt berechnen:

```
memforTData = datasamples * tStringsize * chainParams.numRuns;
```

Die gesamte Größe für die `.p`- und `.t`-Dateien ist dann:

```
totalforpandt = memforPData + memforTData;
```

3.2.5 Typ eines Zyklus

Die Funktion `int TypeOfCycle(void)` liefert als Rückgabewert den Typ des jeweiligen Zyklus. Wenn der benötigte Speicher `totalforpandt` für die `.p`- und `.t`-Dateien größer oder gleich der übermittelten Speichergröße `memoryforall` ist, so liefert sie als Rückgabewert 1, ansonsten 0 zurück. Wenn als Rückgabewert 1 zurückgeliefert wird, so wird 1/4 des Hauptspeichers für die Analyse reserviert. Wenn als Rückgabewert eine 0 zurückgeliefert wird, so wird für die Analyse soviel Speicher reserviert, wieviel angefordert wurde, nämlich `totalforpandt`. Diese Speichergröße wird dann in der Funktion `BuffersForRuns` verwendet, um die Puffer für `.p`- und `.t`-Dateien zu reservieren.

3.2.6 Reservieren von Puffern

Die Funktion `BuffersForRuns(void)` reserviert im Hauptspeicher Puffer für die Ergebnisse der `.p`- und `.t`-Dateien. Sie wird nur von Prozessen aufgerufen, die von der Funktion `IORunAssignment(void)` Durchläufe `nruns` zugewiesen bekommen haben.

In dieser Arbeit wurde das Programm so geändert, dass die berechneten Ergebnisse zunächst in Puffern aufbewahrt, dann parallel in die Datei geschrieben werden.

3.2.7 Kommunikation zwischen Prozessen

Die Funktion `int PIOCommunication(int curGen)` berechnet die Werte der `.p`- und `.t`-Dateien und speichert diese in die jeweiligen Puffer. Als Parameter wird der Funktion der aktuelle Schritt `curGen` der Stammbaumanalyse übergeben. Genau wie die Funktion `PrintStatesToFiles(int curGen)` wird auch diese Funktion von allen MPI-Prozessen aufgerufen. Innerhalb einer Schleife, die über alle Durchläufe `nruns` geht, wird mit der Funktion `int ProcForThisRun (int runId)` geprüft, welcher Prozess für den aktuellen Durchlauf zuständig ist. Der Rückgabewert dieser Funktion wird in die Variable `owner` gespeichert.

Der Prozess, der die ID für ein "cold chain" für den aktuellen Durchlauf besitzt, ruft die Funktion `PrintStates(curGen, coldId)` auf. Als Ergebnis bekommt er die komplette Zeile als Zeichenkette in `printString` zurück. Aus dieser Zeile wird die Teilzeichenkette in die Variable `newprintString` gespeichert. Danach wird geprüft, ob der Prozess, der das Ergebnis berechnet hat, auch für den aktuellen Durchlauf zuständig ist. Wenn also der Prozess `procWithChain` die Bedingung `procWithChain == owner` erfüllt, so speichert er das Ergebnis im eigenen Puffer, ansonsten sendet er das Ergebnis an den zuständigen Prozess `owner`. Bevor ein Prozess das Ergebnis in seinem Puffer speichert, ermittelt er zunächst, in welche Pufferzeile das Ergebnis gespeichert werden soll.

Da ein Puffer mehrere Ergebnisse (Zeilen einer `.p`- oder `.t`-Datei) speichern kann, muss für jeden Puffer ein globaler Zeiger definiert werden. Dieser Zeiger referenziert immer auf das Ende der zuletzt gespeicherten Zeile (siehe Abbildung 3.5).

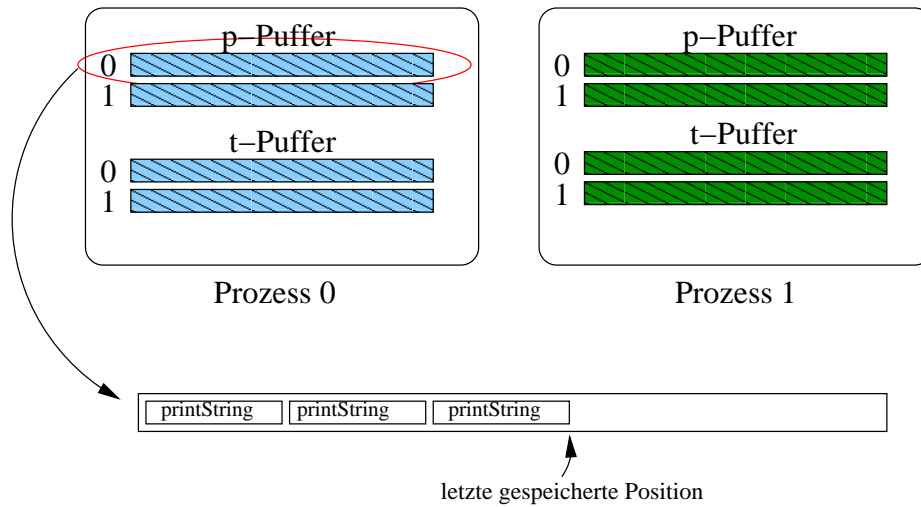


Abbildung 3.5: Zeiger für Puffer

3.2.8 Ermittlung von Durchläufen

Die Funktion `int ProcForThisRun(int run)` liefert als Rückgabewert den zuständigen Prozess für einen Durchlauf. Als Parameter wird der Funktion die ID des jeweiligen Durchlaufs übergeben. Von der Funktion wird geprüft, ob ein Prozess für einen Durchlauf zuständig ist. Wenn ein Prozess diese Bedingung erfüllt, so speichert er seine `proc_id` in die Variable `procRun`. Da für einen Durchlauf immer nur ein Prozess zuständig ist, wird auch nur ein Prozess die obige Bedingung erfüllen, die anderen Prozesse setzen `procRun=0`.

Der Quelltext sieht dann so aus:

```
int ProcForThisRun(int run)
{
    int i;
    procRun=0;

    for (i=0; i<RunsForProc; i++)
    {
        if (IORunId[i] == run)
        {
            procRun = proc_id;
        }
    }
    MPI_Allreduce (&procRun, &proc_owner, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    return proc_owner;
}
```

3.2.9 Erstellen der parallelen Datei

Die Funktion `int PreparePIOFile(void)` erstellt die `.pvs`-Datei für die parallele Ausgabe. Sie prüft zuerst, ob für die Prozesse ein neuer MPI-Kommunikator benötigt wird. Wenn die Variable `NewCommSwitch` den Wert 1 hat, so wird mit `MPI_Comm_create(MPI_COMM_WORLD, newgroup, &newcomm)`; ein neuer Kommunikator erstellt. Die `.pvs`-Datei wird dann nur von den Prozessen parallel geöffnet, die dem neuen Kommunikator gehören.

Vor der Erstellung der `.pvs`-Datei wird geprüft, ob sie bereits vorhanden ist. Wenn diese Datei existiert, so wird der Benutzer darauf hingewiesen und hat die Möglichkeit entweder sie zu löschen oder die neue Analyse in dieselbe Datei anzuhängen. Der Code-Abschnitt sieht wie folgt aus:

```
MPI_File_open(MPI_COMM_WORLD,
              fileName,
              MPI_MODE_RDWR | MPI_MODE_APPEND,
              MPI_INFO_NULL,
              &fhandle);

MPI_File_get_size(fhandle, &fsize);
```

Mit der Funktion `MPI_File_get_size` wird die Größe der vorhandenen Datei in Bytes ermittelt und in die Variable `fsize` gespeichert. Der Wert von `fsize` wird dann in der Funktion `PIO2WriteOut` zu `Displacement` addiert, um alle individuellen Dateizeiger der Prozesse richtig zu positionieren.

3.2.10 Ausgabe in die parallele Datei

Die Funktion `int PIO2WriteOut(void)` wurde implementiert, um die Puffer der `.p`- und `.t`-Dateien parallel in die `.pvs`-Datei zu schreiben. Da die `.p`- und `.t`-Dateien unterschiedlich gross sind und in getrennte Bereiche geschrieben werden, werden zwei abstrakte Datentypen `filetype1` und `filetype2` für den Bereich der `.p`- und `.t`-Dateien erstellt, die dann als `etype` in die parallele Datei geschrieben werden. Es gibt verschiedene MPI-IO Funktionen, um einen abgeleiteten Datentyp zu erstellen. Für diese Arbeit wird die Funktion `MPI_Type_vector` verwendet. Diese liefert für den abgeleiteten Datentyp ein `handle` zurück, welcher aus einer Anzahl von Blöcken eines vorhandenen Datentyps besteht. Da die Daten der `.p`- und `.t`-Dateien nicht zusammenhängend in die Datei geschrieben werden, eignet sich diese Funktion am Besten für einen abgeleiteten Datentyp. Die Abbildung 3.6 zeigt beide Datentypen `filetype1` und `filetype2`.

Da die `.p`- und `.t`-Dateien unterschiedlich groß sind, werden für die parallele Datei zwei Dateisichten (file views) definiert. Die erste Dateisicht (siehe Abbildung 3.7) wird

für den Bereich der .p-Dateien definiert, die zweite Dateisicht (siehe Abbildung 3.8) für den Bereich der .t-Dateien.

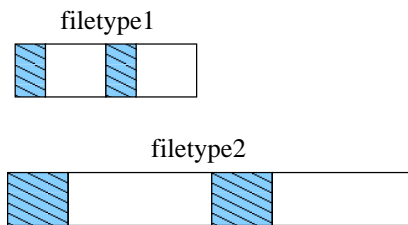


Abbildung 3.6: Abgeleitete Datentypen

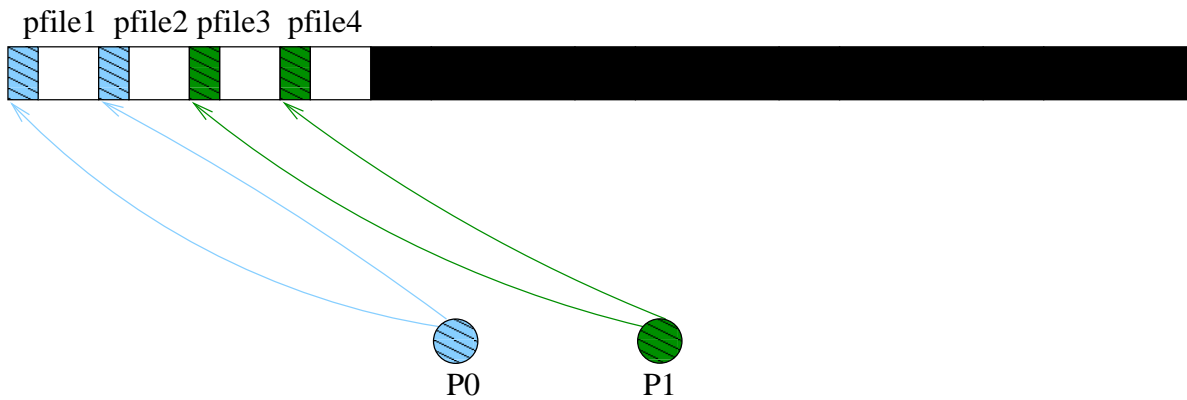


Abbildung 3.7: file view für .p-Dateien

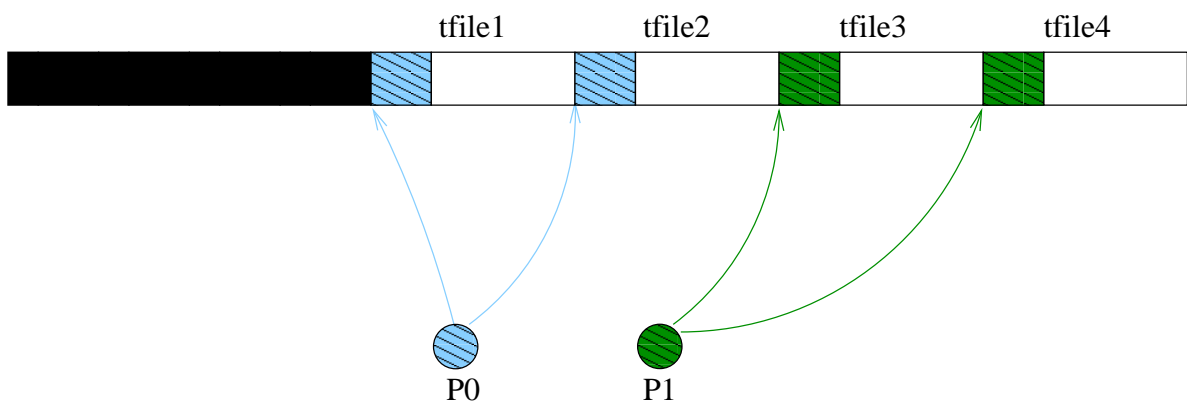


Abbildung 3.8: file view für .t-Dateien

Kapitel 4

Systemumgebung und Tools

Für die Arbeit wurde die Opensource-Software TreeView X [06] verwendet, um die Phylogenie-Bäume grafisch anzuzeigen. Um den Source-Code von MrBayes 3.1 zu analysieren, wurde die Opensource-Software snavigator [07] verwendet.

Kapitel 5

Zusammenfassung

In der Zusammenfassung soll zunächst erläutert werden, welche Ziele in diese Bachelor-Arbeit erreicht worden sind bzw. welche Probleme bei der Implementierung entstanden sind.

MrBayes 3.1 ist eine Anwendung, die zur Laufzeit viele Daten erzeugt und diese in mehrere Dateien schreibt. Während einer Analyse werden Phylogenie-Bäume in die `.t`-Dateien und Wahrscheinlichkeiten in die `.p`-Dateien geschrieben. Das Ziel dieser Bachelor-Arbeit war es, eine Lösung zu implementieren, welche die Ausgabedaten von MrBayes 3.1 parallel in eine einzige Datei schreibt. Dabei sollten die Funktionen der Software-Schnittstelle MPI-IO zum Einsatz kommen.

Um die Daten der `.p`- und `.t`-Dateien parallel in eine einzige Datei zu schreiben, wurde die Datei in zwei Bereiche aufgeteilt. In einem Bereich der Datei werden Daten der `.p`- und im anderen Daten der `.t`-Dateien geschrieben. Dafür wurden zwei abstrakte Datentypen `filetype1` und `filetype2` implementiert. Darüber hinaus wurden für diese Datei zwei verschiedene Dateisichten (file views) definiert.

Momentan werden in die Datei nur einige Wahrscheinlichkeiten und viele ungewollte Zeichen geschrieben, die nicht zur Ausgabe einer Analyse gehören. Diese Zeichen sind unter anderem Umlaute, At-Zeichen usw., die an verschiedenen Stellen in die Datei geschrieben werden. Darüber hinaus werden nur gelegentlich einige Phylogenie-Bäume in die Datei geschrieben.

Ausserdem wird bei einer Analyse nicht immer die gleiche Anzahl von Zeilen in die Datei geschrieben. Momentan wird die beschriebene Ausgabe nur dann in die Datei geschrieben, wenn die Anzahl der `nruns` gleich der Anzahl von Prozessen ist.

Kapitel 6

Ausblick

An dieser Stelle sollen einige Schritte beschrieben werden, wie die Arbeit fortgeführt werden kann.

Als nächstes sollen die implementierten Funktionen verfeinert werden, so dass in die parallele Datei, ausser den Wahrscheinlichkeiten und Phylogenie-Bäumen, keine andere Zeichen geschrieben werden. Dazu sollte die Implementierung der abstrakten Datentypen `filetype1` und `filetype2` und die Prozessbezogene Dateisicht (file view) überprüft werden. Darüber hinaus sollte untersucht werden, ob die Funktion `PI02WriteOut` an der richtigen Stelle aufgerufen wird.

Wie bereits in der Zusammenfassung beschrieben, wird gelegentlich in die Datei geschrieben und das Schreiben geschieht nur dann, wenn die Anzahl von `nruns` gleich der Anzahl von Prozessen ist. Dieses Problem sollte gelöst werden, so dass auch dann in die Datei geschrieben wird, wenn die Anzahl von `nruns` nicht gleich der Anzahl von Prozessen ist.

Literaturverzeichnis

- [01] MrBayes Webseite. <http://mrbayes.csit.fsu.edu/index.php>.
- [02] Message Passing Interface Forum Webseite. <http://www.mpi-forum.org/>.
- [03] Nexus-Format. <http://lms00.psi.ch/NeXus/>.
- [06] TreeView X. <http://taxonomy.zoology.gla.ac.uk/rod/treeview.html>.
- [07] Source-Navigator. <http://sourcnav.sourceforge.net/download.html>.
- [FRvdM05] J. P. Huelsenbeck F. Ronquist and P. van der Mark. *MrBayes 3.1 Manual*. 2005.
- [GTL99] William Gropp, Rajeev Thakur, and Ewing Lusk. *Using MPI-2: Advanced Features of the Message Passing Interface*. MIT Press, Cambridge, MA, USA, 1999.
- [Pri] Christian Printzen. Praktikumsteil Molekulare Systematik. <http://www.bio.uni-frankfurt.de/botanik/MolSystematik.pdf>.
- [Sei] Stefan Seichter. MPI I/O: Die Software-Schnittstelle zum parallelen I/O in MPI-2. http://www2.inf.fh-bonn-rhein-sieg.de/~rberre2m/lehre/ws0102/vps2/Semin%ar_MPI_IO_Seichter.pdf.