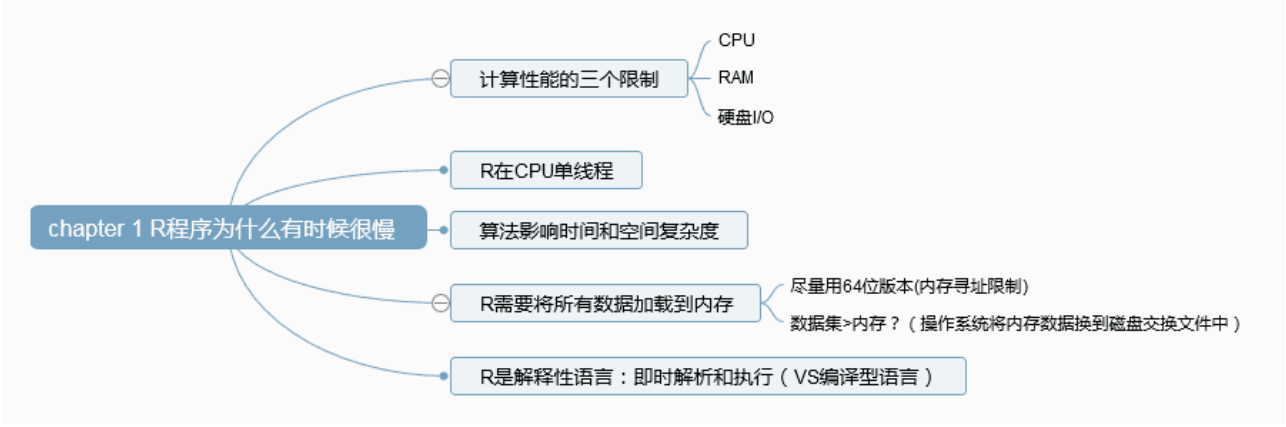
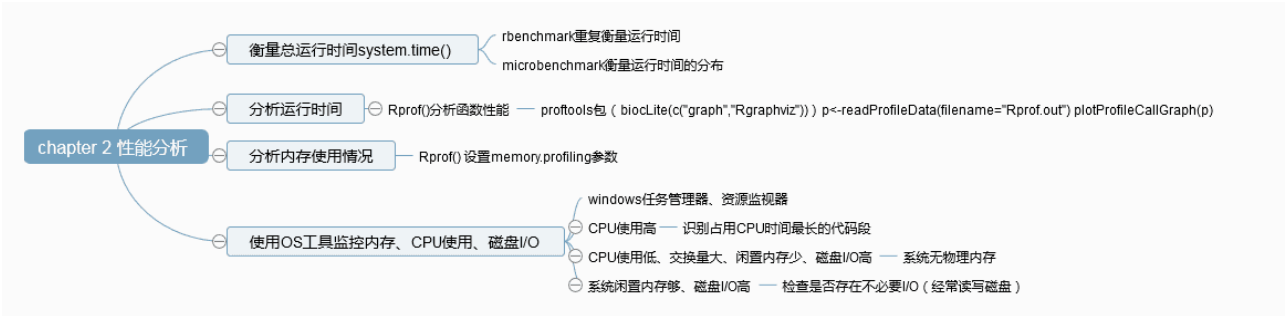
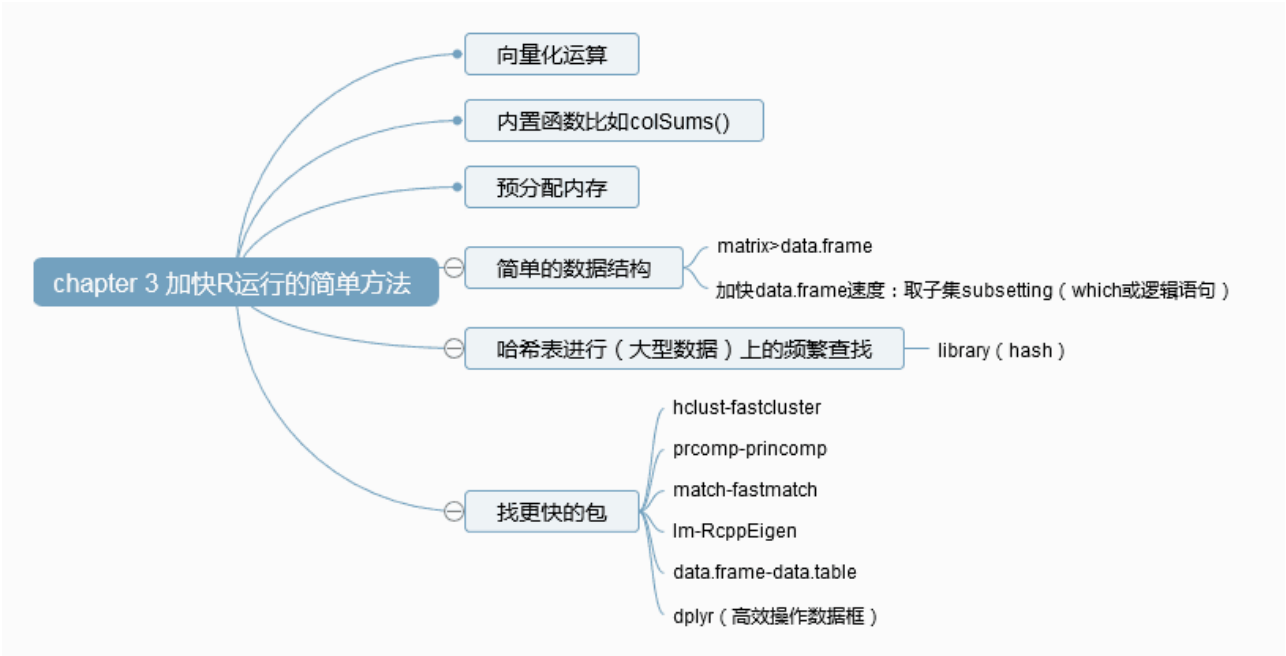
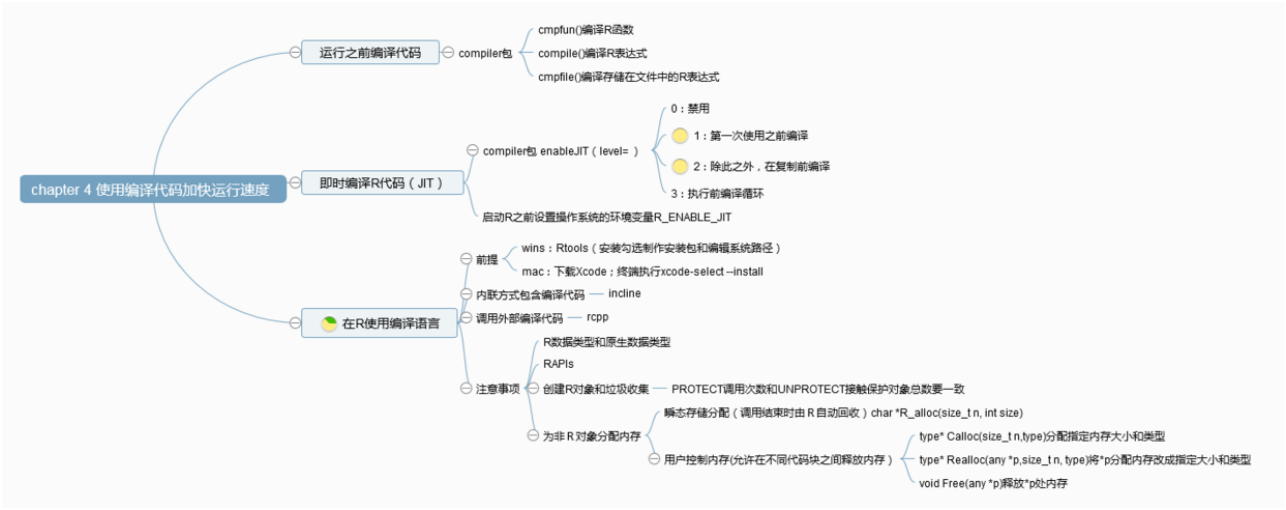
R high performance programming

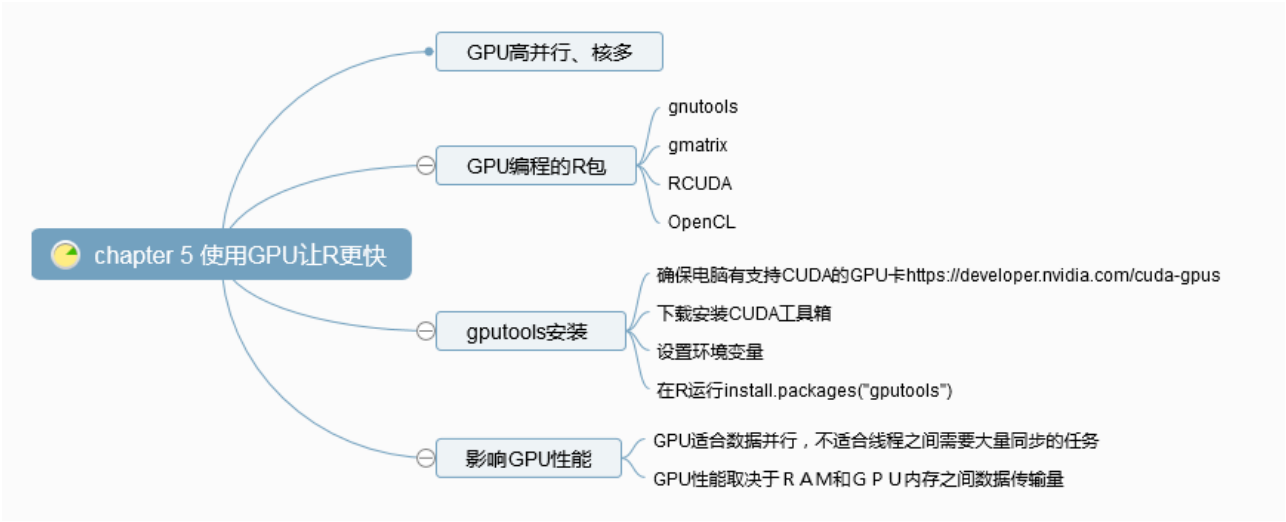
Aloysius Lim， William Tjhi（著）
唐李洋（译）

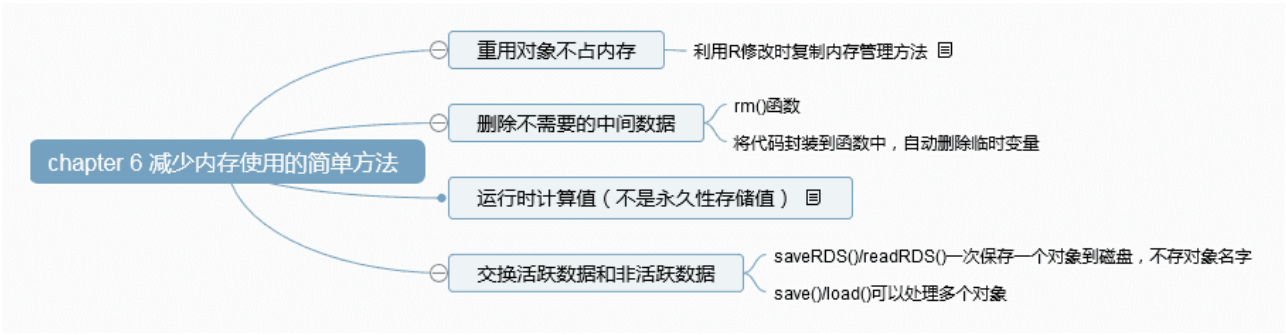












注解：
修改时复制copy-on-modification：
即从已有对象创建新的对象有时并不需要占用额外内存

```
//检查对象大小
object.size()
pryr包的object_size()
//查看对象所指内存块
pryr包的address()
//追踪对象复制方法
tracemem()
```

运行时计算值部分 实例 -- 层次聚类

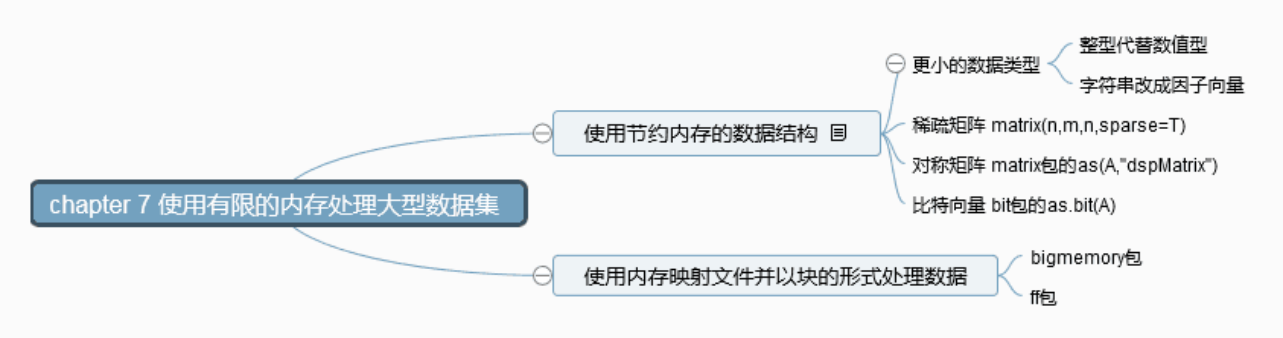
```
//方法1：计算每对观察值的距离矩阵然后决定哪一对最近
```

```
A<-matrix(rnorm(1E5),1E4,10)
dist_mat<-as.matrix(dist(A))
diag(dist_mat)<-NA
res1<-which(dist_mat==min(dist_mat,na.rm=T),arr.ind=T)[1,]
object_size(A)
##800k
object_size(dist_mat)
##801MB
//距离矩阵需要成倍的内存空间存储所有观察对的距离
```

```
//方法二：可以逐对计算，需要内存少,但时间长
library(pdist)
temp_res<-lapply(1:nrow(A),function(x){
temp<-as.matrix(pdist(X=A,Y=A[x,]));
temp[x]<-NA;
output_val<-min(temp,na.rm=T);
output_ind<-c(x,which(temp==output_val));
output<-list(val=output_val,ind=output_ind);
})
val_vec<-sapply(temp_res,FUN=function(x) x$val)
ind_vec<-sapply(temp_res,FUN=function(x) x$ind)
res2<-ind_vec[,which.min(val_vec)]
object_size(temp_res)
##2.72MB
object_size(val_vec)
##80kB
object_size(ind_vec)
##80.2kB
```

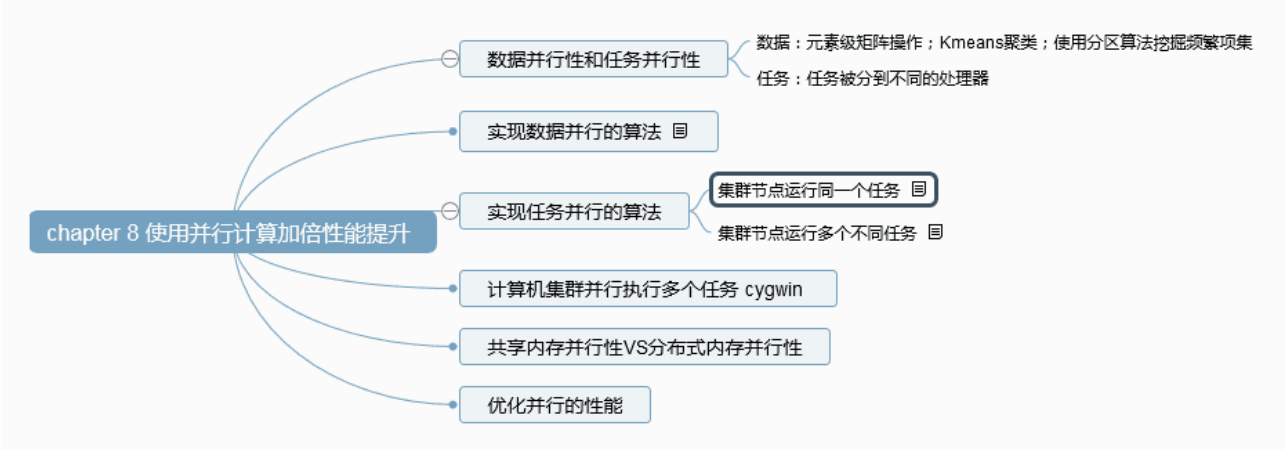
（1）方法2在实际应用可以采用FNN包的knn()；

（2）代码并行化：用parallel包的parLApply()代替lapply()




```
rep.int(x,times):整型
object.size(rep.int("0123456789",1e6))
##8000096 bytes
object.size(rep.int(formatC(seq_len(1e3),width=10),1e3))
##8056040 bytes
object.size(formatC(seq_len(1e6),width=10))
##64000040 bytes
```

字符型向量存储指向包含实际数据的其他向量的指针；需要的存储量取决于向量中唯一字符串的个数。



集群节点运行同一任务

```
#衡量串行算法的运行时间
#L'Ecuyer组合多递归生成器
RNGkind("L'Ecuyer-CMRG'")
nsamples<-5e8
lambda<-10
system.time(random1<-rpois(nsamples,lambda)
```

```
#在集群上生成随机数
#将这个任务平均分配到worker上
cores<-detectCores()
cl<-makeCluster(ncores)
samples.per.process<-
diff(round(seq(0,nsamples,length.out=ncores+1)))
#在基于socket的集群上生成随机数之前，每个worker需要不同的种子来生成随机数流
clusterSetRNGStream(cl)
system.time(random2<-unlist(
parLapply(cl,samples.per.process,rpois,lambda)))
stopCluster(cl)
```

集群节点运行不同任务

```
RNGkind("L'Ecuyer-CMRG'")
nsamples<-5e7
pois.lambda<-10
system.time(random1<-
list(pois=rpois(nsamples,pois.lambda),unif=runif(nsamples),norm=rn
orm(nsamples),exp=rexp(nsamples)))
```

```
cores<-detectCores()
cl<-makeCluster(cores)
calls<-
list(pois=list("rpois",list(n=nsamples,lambda=pois.lambda)),unif=l
ist("runif",list(n=nsamples)),norm=list("rnorm",list(n=nsamples)),
exp=list("rexp",list(n=nsamples)))
clusterSetRNGStream(cl)
system.time(random2<-parLapply(cl,calls,function(call)
{do.call(call[[1]],call[[2]])}))
stopCluster(c)
```


```
dplyr包
library(dplyr)
db.conn<-
src_postgres(dbname="rdb",host="hostname",port=5432,user="ruser",p
assword="rpassword")

#创建两个到数据表sales和trans_items的引用
sales.tb<-tbl(db.conn,"sales")
trans_items.tb<-tbl(db.conn,"trans_items")

#inner_join()联结sales和trans_items表
joined.tb<-inner_join(sales.tb,trans_items.tb,by="trans_id")

#group_by()根据客户ID对项目分组
cust.items<-group_by(joined.tb,cust_id)
cust.spending<-summarize(cust.items,spending=sum(price))
cust.spending<-arrange(cust.spending,desc(spending))
cust.spending<-select(cust.spending,cust_id,spending)

#collect()用于运行SQL语句并获取结果
custs.by.spending<-collect(cust.spending)
top.custs<-head(cust.spending,10L)

#dplyr包提供%>%将操作联结起来，前面可以写为
top.custs<-sales.tb%>%inner_join(trans_items.tb,by="trans_id")
%>%grouped_by(cust_id)%>%summarise(spending=sum(price))
%>%arrange(desc(spending))%>%select(cust_id,spending)%>%head(10L)

PivotalR包
library(PibotalR)
db.conn<-
db.connect(host="hostname",port=5432,dbname="rdb",user="ruser",pas
sword="rpassword")
sales.tb<-db.data.frame("sales",db.conn)
trans_items.tb<-db.data.frame("trans_items",db.conn)
#执行SQL并获取结果
lookat(count(sales.tb$cust_id))
#content方法查看数据库服务器执行的SQL查询
```

```
content(max(trans_items.tb$price))
trans<-by(trans_items.tb['price'],trans_items.tb$trans_id,sum)
sales.value<-
merge(sales.tb[c("trans_id","cust_id","store_id")],trans,by="tran-
s_id")
cust.sales<-by(sales.value,sales.value$cust_id,function(x){
trans_count<-count(x$trans_id)
total_spend<-sum(x$price_sum)
stores_visited<-count(x$store_id)
cbind(trans_count,total_spend,stores_visited)})
names(cust.sales)<-
c("cust_id","trans_count","total_spend","stores_visited")
lookat(cust.sales,5)
使用列式数据提升性能
MonetDB（https://www.monetdb.org/Downloads）
windows选择开始|程序|MonetDB|启动服务器，初始化并启动服务器。
library(MonetDB.R)
db.drv<-MonetDB.R()
db.conn<-
dbConnect(db.drv,host="hostname",post=50000,dbname="rdb",user="mon
etdb",password="monetdb")
dbWriteTable(db.conn,"sales",sales)
dbWriteTable(db.conn,"trans_items",trans.items)
library(microbenchmark)
microbenchmark({res<-dbGetQuery(db.conn,'SELECT
store_id,SUM(proce) as total_sales FROM sales INNER JOIN tran-
s_items USING (trans_id) GROUP BY strore_id;')},times=10)
使用数据库阵列最大化科学计算的性能
1、下载安装SCIDB
2、在SCIDB服务器安装shim
3、从CRAN安装scidb包

library(scidb)
scidbconnect(host="hostname",port=8080)
#使用as.scidb()将数据装载到数据库
A<-as.scidb(matrix(rnorm(1200),40,30),name="A")
B<-as.scidb(matrix(rnorm(1200),30,40),name="B")
#scidb提供类似r的语法来操纵SCIDB矩阵和数组
```