SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

# Seminarski rad

# Paralelno programiranje u programskom jeziku R

**Dragana Čolić**

**voditelj: prof. dr. sc. Branko Jeren**

Zagreb, travanj 2009.

## Sadržaj

# 1. Uvod

Cilj ovog seminarskog rada je upoznavanje s metodama paralelnog programiranja u programskom paketu R. Riječ je o istoimenom programskom jeziku i radnom okruženju za statističko računarstvo i grafiku. Za R je izgrađen niz programskih paketa za ostvarenje različitih specifičnih namjena, kao što su implementacija MPI (eng. Message Passing Interface) standarda i algoritama za obradu velikih skupova podataka (klasifikacijski algoritmi). Budući da je jedna od glavnih osobitosti R jezika i razvojne okoline pogodnost za programsku obradu velike količine podataka što je vremenski zahtjevno, kao mogućnost poboljšanja i ubrzanja izvedbe, nameće se paralelno izvođenje programa.

Ideja paralelnog programiranja zasniva se na usporavanju i ograničenjima mogućeg rasta računalne moći pojedinog računala, ali na sve većem broju dostupnih umreženih računala. Raspodjela poslova između računala, ubrzava izvedbu. Postoji nekoliko različitih pristupa razvoju paralelnih programa, od kojih će se ovdje pažnja pridati MPI standardu. U pravilu je kod MPI programa paralelizacija i raspodjela poslova izvedena unutar jednog programa, koji se izvodi na svim računalima, a ona međusobno komuniciraju porukama.

Dostupne implementacija algoritama za obradu velikih skupova podataka za R su slijedne te je ideja, u konačnici, korisnicima R paketa omogućiti paralelno izvođenje tih algoritama.

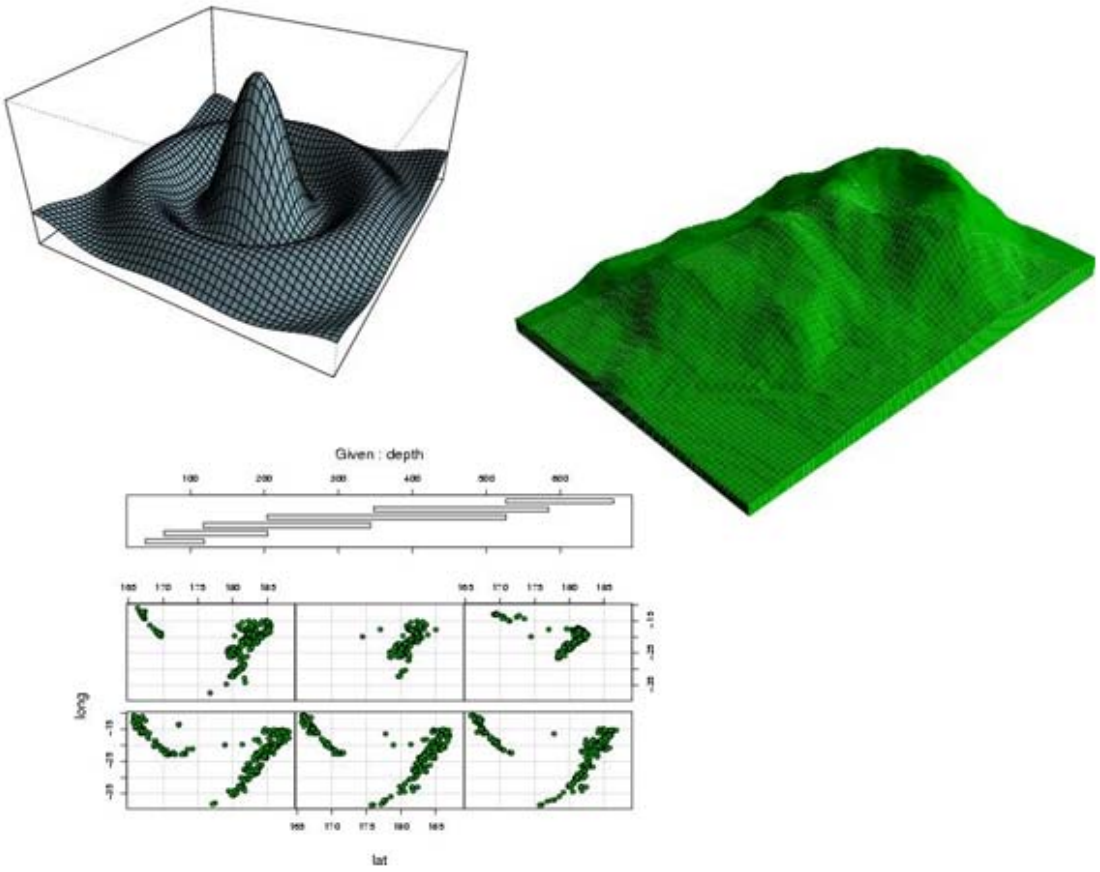
# 2. O programskom paketu i jeziku R

Kao što je rečeno u uvodu, R je programski jezik za statističko programiranje i grafiku. Riječ je o programu s GNU licencom što znači da je originalno namijenjen Unix operativnim sustavima i potpuno otvorenog koda te ga svaki korisnik može nadograđivati, kopirati, mijenjati i distribuirati. R je nasljednik S programskog jezika, razvijenog u Bell Laboratoriesa 70-ih godina prošlog stoljeća. Naziv S odabran je na temelju glavne motivacije za oblikovanje tog jezika, a to je statističko programiranje. Značajan dio programskog koda napisanog za S programski jezik, koristi se nepromijenjen i unutar R paketa.

Osim standardnih mogućnosti programskih jezika, kao što su petlje, uvjetni izrazi, korisničke funkcije, ulazno/izlazne operacije, R omogućuje i izvođenje računskih operacija nad matricama te alate za grafičku obradu i izuzetno kvalitetan prikaz podataka, koji korisnik u velikoj mjeri može sam oblikovati. Metode statističkog programiranja, dostupne u R-u uključuju linearno i nelinearno oblikovanje, klasične statističke testove, analizu vremenskih nizova, klasifikaciju podataka i dr.

Osim R okruženja unutar kojeg se interaktivno naredbama mogu pokretati ugrađene, nadograđene ili korisničke funkcije i skripte, dostupan je i niz korisničkih grafičkih sučelja za R. Neki od njih su:

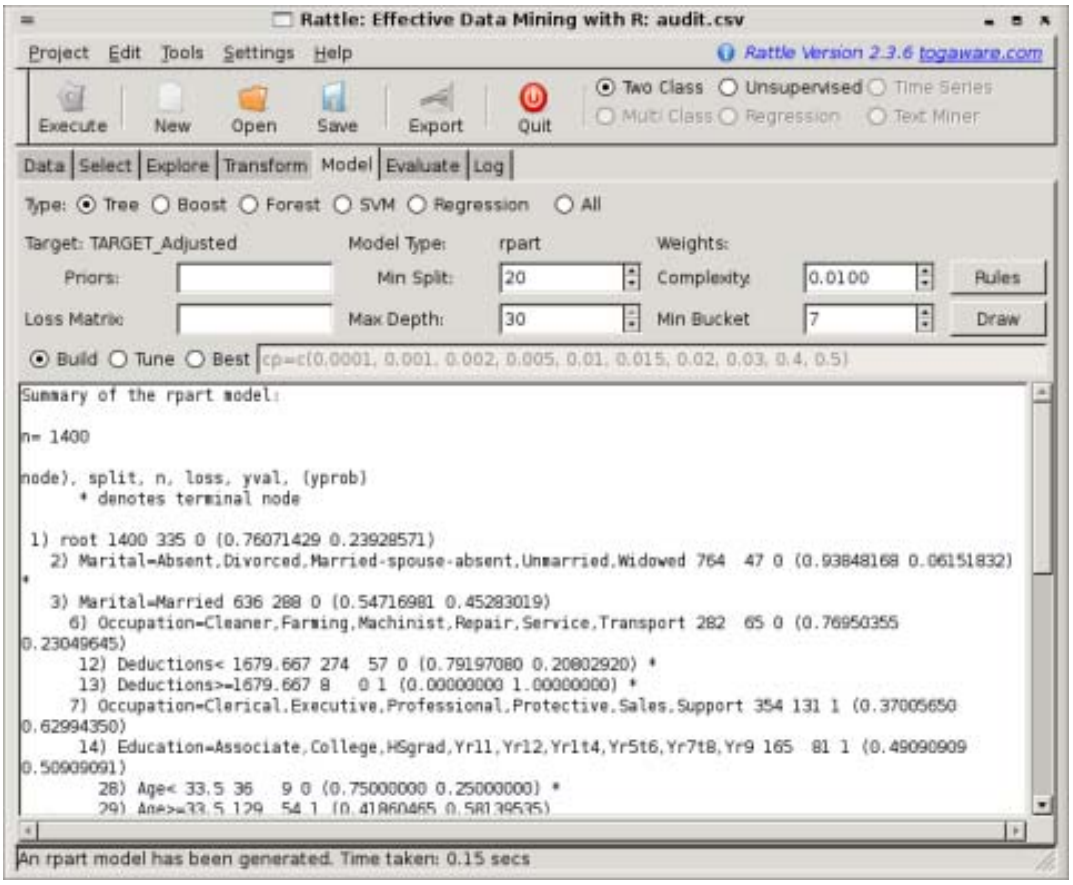
- Rattle,
- Statistical Lab,
- PMG,
- Java GUI for R,
- StatET i dr.



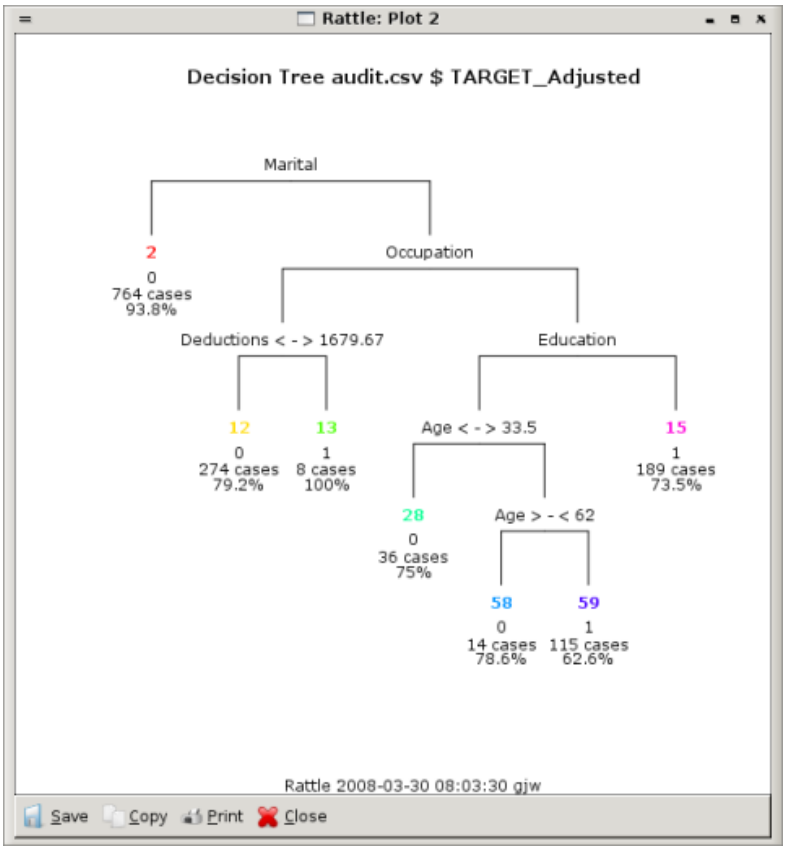

***Slika 1.*** *R grafički objekti: 3D grafovi, usporedni dijagrami*

Osim programskih paketa koje uključuje osnovni R paket, dostupan je čitav niz besplatnih programskih dodataka za R kojima se on može nadograditi u bilo kojem trenutku. Oni su dostupni putem CRAN (eng. The Comprehensive R Archive Network) projekta koji uključuje mrežu FTP i web poslužitelja na kojima se nalaze identične i ažurne inačice programskih paketa i pripadne dokumentacije za R paket.

Valja napomenuti da je značajan dio programskog koda za R napisan u Fortranu i C/C++-u, a dijelovi programskog koda napisani u nekom od tih jezika mogu biti direktno pozvani prilikom izvođenja R programa. C jezik može se koristiti i za manipuliranje R objektima.




***Slika 2.*** *Rattle grafičko korisničko sučelje*




***Slika 3.*** *Graf oblikovan pomoću Rattle GUI-a*

R je danas dostupan za operacijske sustave Linux, MacOSX i Windows, a skida se s najbližeg CRAN web/ftp poslužitelja.

# 3. Paralelno programiranje

Paralelno programiranje je pristup u oblikovanju programa koji se temelji na ideji iskorištavanja računalne moći većeg broja umreženih računala. Budući da je rast brzine rada procesora ograničen fizikalnim svojstvima i zakonima, a danas se tim ograničenjima sve više približavamo, potrebno je naći druge načine ubrzavanja izvedbe programskih zadataka. Paralelizacija posla na razini sklopovlja izvodi se danas na više načina:

- cjevovodi,
- višedretvenost,
- višejezgrenost procesora,
- GP/GPU.

Računala na kojima se izvode paralelni programi mogu biti umrežena računala lokalne mreže (grozd) ili raspršena računala u Internetu(splet računala). Za razliku od grozda računala koji je danas uobičajen, splet računala je model koji će zaživjeti tek budućnosti.
Osim modela s više računala, program se može izvoditi na jednom računalu s više procesora – to je model zajedničke memorije. U slučaju izvođenja programa na većem broju računala, govori se o modelu raspodijeljene memorije. To znači da svako računalo ima svoju lokalnu memoriju, a razmjena podataka odvija se preko poruka. Takav način komunikacije kod paralelnih programa opisan je MPI standardom.
Prije oblikovanja paralelnih programa važno je znati kakav treba biti dobar paralelni program, koji su koraci u njegovu oblikovanju i što sve potrebno za paralelno računanje:

1) Dobar paralelni program mora:
    a) izvoditi više radnji istovremeno,
    b) glavninu vremena trošiti na lokalni rad, a manji dio na udaljenu komunikaciju,
    c) biti fleksibilan u odnosu na broj računala na kojima se izvodi i
    d) imati dijelove koji su uporabivi u okviru nekih drugih algoritama(modularnost).
2) Koraci u oblikovanju paralelnih algoritama su:
    a) detaljno analizirati slijedni algoritam i pronaći dijelove pogodne za paralelizaciju
    b) rastaviti algoritam funkcionalnom i podatkovnom dekompozicijom,
    c) ostvariti program u odabranoj programskoj paradigmi i sklopovskom okruženju,
    d) ispravljanje grešaka i optimiranje izvođenja.

3) Za paralelno računanje potrebno je imati:

   a) paralelni algoritam,

   b) okolinu za paralelni rad,

   c) arhitekturu više računala/procesora i

   d) vezu između procesora/računala.

Algoritmi pogodni za paralelizaciju su algoritmi koji često obavljaju neovisne računske operacije ili obavljaju operacije nad skupom neovisnih podataka. Dekompozicija slijednog algoritma u paralelni pritom zahtjeva funkcionalnu, odnosno podatkovnu dekompoziciju, odnosno rastavljanje algoritma na dijelove koji su međusobno neovisni s obzirom na podatke koje koriste ili funkcije koje izvode.Upravo je podatkovna neovisnost česta za statističke algoritme, jer oni provode sličan niz operacija nad izuzetno velikim skupovima podataka što je glavni uzrok njihove sporosti.

# 4. Paralelno programiranje u R-u

R programiranje posebno je pogodno za paralelizaciju jer se kod statističkih obrada podataka često radi o trivijalno paralelnim zadacima. To znači da se:

- ista operacija obavlja na manjim, neovisnim podskupovima podataka – podatkovna neovisnost,
- ili se podaci obrađuju na različite načine (ispituju se različita statistička svojstva istog skupa podataka) – funkcionalna nezavisnost.

Budući da je statističke programe često vrlo lako dekomponirati na skupove nezavisnih zadataka, dolazi se do zaključka da su R programi izuzetno pogodan za paralelizaciju. Osim toga, paralelizacija R programa ima smisla jer za njom postoji i realna potreba. Naime, statističke obrade podataka zahtijevaju velike skupove podataka što znači da je zbog njihove kvantitete obrada vremenski relativno zahtjevna. S druge strane, paralelizacijom se ona može ubrzati za određeni faktor.
Taj se faktor može opisati Amdahlovim zakonom:

$$ubrzanje = \frac{1}{S + \frac{P}{N}}$$

Gdje je S udio slijednog dijela u programu, P udio paralelnog dijela, a N broj dostupnih procesora. Nekoliko primjera ubrzanja dano je u slijedećoj tablici:

| Trajanje slijednog programa | Udio slijednog dijela | Broj procesora | Ubrzanje | Trajanje paralelnog programa |
|---|---|---|---|---|
| 30h | 0,5 | 10 | 1,8 | 16,5h |
| 30h | 0,2 | 20 | 4,2 | 7,2h |

***Tablica 1.*** *Primjeri ubrzanja programa*

Paralelno izvođenje programa podrazumijeva infrastrukturu koja omogućuje koja omogućuje komunikaciju između različitih računalnih čvorova (jezgre, procesori, računala). Dva su dostupna oblika ovakve podrške za mrežu računala:

1. PVM (eng. Parallel Virtual Machine) – koji ostvaruje komunikaciju između računala u heterogenoj mreži i pruža sučelje putem kojeg se takva računala mogu koristiti kao jedno računalo.

2. MPI (eng. Message Passing Interface) – standard koji definira komunikaciju porukama između računala povezanih lokalnom mrežom.

Prednost PVM pristupa je ta što je on pogodan za heterogene mreže računala, odnosno za programiranje u spletu (eng. grid) poput Interneta koji uključuje računala s različitim OS-ovim, različitim protokolima i sl. PVM također karakterizira otpornost na pogreške i mogućnost dinamičkog upravljanja resursima i mrežom.
Budući da je paralelno izvođenje programa u spletu računala još uvijek napredna ideja čije se puno ostvarenje očekuje tek u budućnosti, u ovom trenutku MPI se nameće kao bolji izbor što se može opravdati slijedećim prednostima:

- MPI je formalno specificiran standard,
- njegove implementacije su optimizirane za pojedine arhitekture,
- ima više dostupnih implementacija i bogatije mogućnosti komunikacije od PVM-a.

MPI je pogodan za paralelno izvođenje programa na grozdovima računala koji predstavljaju optimalan pristup jer omogućuju uporabu više računalnih čvorova od višejezgrenih i višeprocesorskih računala, a lakše su izvedivi od spleta računala.
Iako su u obliku R paketa dostupne implementacije PVM-a (rpvm) i MPI standarda (rmpi), dalje u dokumentu razmatrane metode podrazumijevaju primjenu MPI standarda.
Korištenje R paketa za paralelizaciju programa podrazumijeva postojanje paralelne infrastrukture na svim računalima u spletu. Za MPI dostupno je nekoliko takvih besplatnih implementacija:

- MPICH, MPICH2,
- LAM/MPI,
- OpenMPI i
- DeinoMPI.

## 4.1 Rmpi

Rmpi paket je implementacija MPI standarda za R, a može se koristiti na Windows, Linux i MacOS grozodovima.. Budući da je MPI dobar i danas najčešći način paralelne izvedbe programa, upravo je on odabran za paralelizaciju R programa.
MPI standard definira komunikaciju porukama i razmjenu podataka između računala koja paralelno izvode isti program. U nastavku se navode neke osnovne MPI funkcije, prikazane u R implementaciji:

- **lamhosts()** – vraća brojeve procesora i imena njihovih poslužitelja

- **mpi.is.master()** – vraća TRUE ukoliko je računalu koje ga poziva dodijeljena uloga upravitelja paralelnog izvođenja, FALSE inače.
- **mpi.hostinfo()** – ispisuje broj, ulogu, grupu u kojoj se nalazi i ime poslužitelja pozivajućeg procesa.
- **mpi.bcast(x, type, rank = 0, comm = 1)** – proces s brojem *rank* (pretpostavljena vrijednost je 0) šalje podatke tipa *type* sadržane u varijabli *x* svim procesima u grupi *comm* (pretpostavljena je grupa svih postojećih procesa). Kako bi poziv uspio moraju ga pozvati svi procesi iz grupe *comm.*
- **mpi.scatter (x, type, rdata, root = 0, comm = 1)** – proces *root* šalje podatke tipa *type* iz spremnika *x* svim procesima iz grupe *comm*, a oni ga spremaju u spremnik *rdata*. Važno je napomenuti da tipovi type moraju odgovarati u pozivu funkcije kod procesa koji primaju podatak i kod onog koji ga šalje.
- **mpi.gather(x, type, rdata, rcounts, root = 0, comm = 1)** – svi procesi iz grupe *comm* šalju svoj podatak *x*, koji je tipa *type* i duljjine *rcounts*. Proces *root* sakuplja te podatke u svoj spremnik *rdata* koji mora biti dovoljno velik i tipa *type*. Za ovu funkciju može se reći da je suprotna funkciji scatter.
- **mpi.reduce(x, type,op = c(„sum“, “prod“, „max“, „min“, “maxloc“, „minloc“), dest = 0, comm = 1)** – funkcija sakuplja podatak *x* sa svih procesa iz grupe *comm* izvodi nad njima operaciju *op*, rezultat se sprema u procesu s oznakom *dest*.
- **mpi.send(x,type,dest,tag, comm = 1)** – šalje se poruka *x*, tipa *type*, s oznakom *tag*, računalu *dest* iz grupe *comm.*
- **mpi.recv(x, type, source, tag, comm = 1, status = 0)** – računalo očekuje poruku od procesa *source* iz grupe *comm*, a status se zapisuje u varijabli *status*. Da bi razmjena poruke uspjela, parametri *tag* i *type* moraju biti isti u pozivu *send* i *recv*, a *source/dest* parametri moraju odgovarati brojevima procesa koji pozivaju *send/recv* funkcije.
- **mpi.finalize()** – završava paralelno izvođenje i moraju je pozvati sva sudjelujuća računala prije završetka programa.

Iako je ovo samo dio osnovnih MPI funkcija, one omogućuju izvedbu velike većine paralelnih programa.

## 4.2 R paketi za paralelno programiranje na grozdu računala

Osim već spomenutih rpvm i rmpi paketa, dostupno je nekoliko drugih paketa koji omogućuju i olakšavaju paralelno programiranje na grozdu računala (koji može biti jedno računalo ili više njih). Neki od njih su:

- Papply – riječ je o paketu koji sadrži jednu funkciju čijim se pozivom paralelno pokreće ista operacija nad zadanim podacima (mogu biti različiti za svaki proces). Ukoliko nije dostupna arhitektura više računala, program se izvodi slijedno. Papply koristi Rmpi funkcije pa je preduvjet za njegovo korištenje instalacija Rmpi paketa.
- Biopara – omogućuje raspodjelu poslova između čvorova putem ssh veza. Veze se uspostavljaju eksplicitno zadavanjem imena računala i priključnica (eng. port) direktno funkciji. Dakle ne koristit se nikakva paralelna infrastruktura. Nedostatak paketa je osjetljivost na pogreške i slabo ujednačavanje opterećenja među računalima.
- snow – uključuje funkcije slične papply funkciji, ali i šire mogućnosti za upravljanje paralelnim izvođenjem na grozdu. Može se izvoditi na MPI, PVM i NWS sustavima, a pretpostavljeni je sustav MPI.
- snowFT i snowfall - riječ je o ekstenzijama snow paketa koje osiguravaju otpornost na greške(snowFT) i dodatne mogućnosti poput slijednog izvođenja u slučaju nedostupnosti paralelne arhitekture (snowfall). Snow paket karakterizira jednostavnost uporabe i visoka učinkovitost izvedbe.
- Nws (eng. NetWorkSpaces) - omogućuje koordinaciju i razmjenu podataka između programa, a namijenjen je skriptnim jezicima (zasad podržava Matlab, Python i R).
- Task-pR – zahtijeva postojanje LAM/MPI implementacije kako bi paralelno izvodio programe i ne podržava paralelne funkcije visoke razine poput papply-a.

Analiza navedenih paketa pokazala je kako se najučinkovitije paralelno izvođenje programa postiže primjenom snow paketa i njegovih ekstenzija snowFT i snowfall [2].

## 4.3 Ostali dostupni paketi

Osim navedenih paketa za programiranje na grozdu računala dostupni su i paketi za višejezgrene sustave:

- pnmath i pnmath0 – paketi koji osnovne R numeričke metode zamjenjuju s optimiziranim paralelnim inačicama,

- fork – uključuje R programske funkcije koje omogućuju korištenje fork() i sličnih poziva sustava kojima se može upravljati većim brojem procesora,
- R/parallel – omogućuju paralelizaciju podatkovno nezavisnih petlji.

Također dostupni su i R paketi za grid programiranje: gridR, multiR i Biocep-R. Riječ je o izuzetno mladim rješenjima (2007., 2008.) koji iskorištavaju različite mogućnosti računala u infrastrukturi spleta.

# 5. Izvedba programa

Za primjer primjene i ostvarenja paralelnih programa u R-u odabrana je MPI implementacija te papply funkcija. Pisanje i izvođenje paralelnih R programa može se izvoditi direktno unutar R-a, ali moguće je koristiti i slijedni programski kod pisan u C-u. Takav se kod omata R funkcijom te se unutar R skripte može pozivati za slijedno ili paralelno izvođenje. U svrhu paralelizacije moguće je koristiti osnovne MPI funkcije iz Rmpi paketa ili funkcije više razine dostupne unutar drugih paketa, npr. snow ili papply.

Uopće korištenje MPI implementacije zahtijeva paralelno pokretanje R-a na željenim računalima u grozdu. Za OpenMPI to se može učiniti pokretanjem sljedeće naredbe:

*mpirun -np 1 -machinefile machines R –vanilla*

Što znači da se pokreće jedan glavni R ili „master“ proces, a broj i lokacija dostupnih „slave“ (pomoćnih) procesa dani su u datoteci „machines“ čiji sadržaj može biti slijedeći:

```
host1  slots=1 max_slots=5
host2 slots=1 max_slots=5
host3 slots=2 max_slots=5
```

*HostX* je pritom ime računala u grozdu, *slots* broj procesora dostupnih na tom računalu, a *max_slots* maksimalni broj procesa koji se može pokrenuti.

Za primjer koristimo funkciju napisanu u C-u koja računa informacijsku dobit atributa u odnosu na klasu:

```
#include<R.h>
#include<Rmath.h>

//v - ulazni vektor atributa
//s - veličina ulaznog vektora
//k - vektor klasa
//k_p - moguće vrijednosti klase
//v_p - moguće vrijednosti ulaznog atributa
//v_p_s - broj različitih mogućih vrijednosti atributa
//k_p_s - broj različitih vrijednosti klase
//rez - varijabla u koju spremamo rezultat
void info_gain(int *v, int *s, int *k, int *k_p, int *v_p, int *v_p_s, int *k_p_s, double *rez)
{
  int i,j,l,proba;
  int count;
  double p;
  double entropy1 = 0, entropy2 = 0, entropy3 = 0;
 for(proba = 0; proba < 1; proba++)
 {
  ///H(atribut)
  for(i = 0; i < *v_p_s; i++)
  {
    count = 0;
    for(j = 0; j < *s; j++)
{
     if(v[j] == v_p[i])
```

```
    count++;
  }
  p = count/(double)(*s);
  if(p != 0)
  entropy1 -= p*log(p)/log(2.f);
 }
 *rez = entropy1;

 //H(klase)
  for(i = 0; i < *k_p_s; i++)
          {
                  count = 0;
                  for(j = 0; j < *s; j++)
                  {
                    if(k[j] == k_p[i])
                            count++;
                  }
                  p = count/(double)(*s);
  if(p != 0)
                  entropy2 -= p*log(p)/log(2.f);
           }
          *rez += entropy2;

          //H(klase&atr)
          for(i = 0; i < *k_p_s; i++)
                  for(l = 0; l < *v_p_s; l++)
                  {
                          count = 0;
                          for(j = 0; j < *s; j++)
                          {
                            if(k[j] == k_p[i] && v[j] == v_p[l])
                                    count++;
                          }
                          p = count/(double)(*s);
    if(p != 0)
                          entropy3 -= p*log(p)/log(2.f);
                   }
                  *rez -= entropy3;
}
}
```

Funkcije koje se pišu za primjenu u R-u obavezno moraju biti tipa *void*, a sve argumente primaju preko pokazivača te će se u njih pisati i povratni rezultati. Naredbom *R CMD SHLIB info_gain.c* pokreće se R prevodilac C koda koji stvara datoteku *info_gain.so*. Ta se funkcija može koristiti za pozivanje iz R-a, nakon što se učita naredbom *dyn.load(„info_gain.so“)*. Kako bi se olakšala uporaba funkcija se može omotati u R funkciju na sljedeći način:

```
   info_gain <- function(x)
{
 dyn.load("info_gain.so")
 rez = 0.0
 out <- .C("info_gain",
         vektor = as.integer(x[[1]]),
         size = as.integer(length(x[[1]])),
         klase = as.integer(x[[3]]),
         klase_poss = as.integer(x[[4]]),
```

```
        poss = as.integer(x[[2]]),
        poss_size = as.integer(length(x[[2]])),
        klase_poss_size = as.integer(length(x[[4]])),
        rez = as.numeric(rez))
        return(out$rez)
}
```

Ovakva se funkcija može pozivati direktno iz R sučelja. Za ilustraciju koristit ćemo skup podataka iz „UCI adult“ baze kojima se iznos prihoda osobe povezuje s atributima kao što su životna dob, rasa, bračni status, obrazovanje, spol, nacionalnost i sl. Ukupan broj atributa je 15, a ¸broj uzoraka oko četrdeset tisuća. Takva struktura u R „data.frame“ formatu izgleda ovako

```
25 Private 226802 11th 7 Never-married Machine-op-inspct Own-child Black Male 0 0 40 United-States <=50K
38 Private 89814 HS-grad 9 Married-civ-spouse Farming-fishing Husband White Male 0 0 50 United-States
<=50K
28 Local-gov 336951 Assoc-acdm 12 Married-civ-spouse Protective-serv Husband White Male 0 0 40 United-
States >50K
….
```

Cilj je dakle izračunati koji atribut daje najviše informacije o klasi. Ovaj se postupak inače više puta ponavlja kod izgradnje klasifikatora stablo odlučivanja tako da ponavljanjem ovog izračuna možemo napraviti grubu simulaciju rada tog klasifikatora. Izračunavanje informacijske dobiti za različite atribute u istom koraku može se obavljati neovisno, a definicija informacijske dobiti iz teorije informacije je slijedeća

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

Gdje je H(X) entropija atributa X koja se izračunava kao negativna suma umnožaka vjerojatnosti pojave različitih vrijednosti atributa X i logaritma po bazi 2 tih vjerojatnosti, odnosno formulom

$$H(X) = -\sum_{x_i \in X} P(X = x_i) \log P(X = x_i)$$

Očito je riječ o trivijalno paralelnom zadatku za kojeg možemo iskoristiti prethodno napisanu funkciju *info_gain*. Funkcija pogodna za paralelnu implementaciju ovog računa je *papply*(), a prima kao argumente listu ulaznih podataka te naziv funkcije koja će se provesti nad tim podacima. Povratna vrijednost *papply* funkcije je lista povratnih vrijednosti paralelno pozvanih funkcija. Jedan od načina na koje možemo oblikovat ulaznu listu i papply poziv je:

```
library(papply)
source("info_gain.R")
table = read.table("Dataset.data")
f = make.names(c("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o"),unique = TRUE)
names(table) <- f

a = as.vector(table$a,"integer")
b = as.vector(table$b,"integer")
d = as.vector(table$d,"integer")
e = as.vector(table$e,"integer")
f = as.vector(table$f,"integer")
```

```
g = as.vector(table$g,"integer")
h = as.vector(table$h,"integer")
i = as.vector(table$i,"integer")
j = as.vector(table$j,"integer")
o = as.vector(table$o,"integer")
n = as.vector(table$n,"integer")


ulaz = list(
        list(a, c(min(a):max(a)),o, c(min(o):max(o))),
        list(b, c(min(b):max(b)),o, c(min(o):max(o))),
        list(d, c(min(d):max(d)),o, c(min(o):max(o))),
        list(e, c(min(e):max(e)),o, c(min(o):max(o))),
        list(f, c(min(f):max(f)),o, c(min(o):max(o))),
        list(g, c(min(g):max(g)),o, c(min(o):max(o))),
        list(h, c(min(h):max(h)),o, c(min(o):max(o))),
        list(i, c(min(i):max(i)),o, c(min(o):max(o))),
        list(j, c(min(j):max(j)),o, c(min(o):max(o))),
        list(n, c(min(n):max(n)),o, c(min(o):max(o))))

c1 = proc.time()[3]
rez = papply(ulaz,info_gain)
c2 = proc.time()[3] - c1
```

Pritom nismo koristili sve atribute, samo njih 10. To je učinjeno zbog prevelikih raspona različitih vrijednosti ostalih atributa koji nisu pogodni za napisanu *info_gain.c* funkciju jer ona računa entropije prema svim mogućim vrijednostima varijable što je za slučaj numeričkih vrijednosti često prevelik raspon. U slučaju numeričkih varijabli valjalo bi postaviti ograničenja i koristiti operatore „<“,“>“ umjesto operatora „==“.

Rezultati mjerenja vremena izvođenja za paralelne i slijedne pozive funkcije *info_gain* dani su u sljedećoj tablici:

| **Broj procesa** | **Veličina petlje** | **Vrijeme izvođenja** | **Ubrzanje u odnosu na slijednu inačicu** |
|---|---|---|---|
| 1 | 1 | 0.81 | 1 |
| 2 | 1 | 2.12 | 0.38 |
| 3 | 1 | 2.62 | 0.309 |
| 4 | 1 | 4.19 | 0.20 |
| 1 | 500 | 45,73 | 1 |

| | | | |
|---|---|---|---|
| 2 | 500 | 29,34 | 1,56 |
| 3 | 500 | 22,71 | 2,01 |
| 4 | 500 | 31,57 | 1,45 |
| 1 | 3000 | 269.69 | 1 |
| 2 | 3000 | 172.28 | 1,56 |
| 3 | 3000 | 168.85 | 1,60 |
| 4 | 3000 | 180.02 | 1,50 |

***Tablica 2.*** *Rezultati mjerenja vremena izvođenja programa*

Rezultati mjerenja vremena izvođenja programa dali su navedene rezultate iz kojih se može zaključiti kako paralelizacija same *info_gain()* funkcije nema smisla jer je trošak vremena na međuprocesnu komunikaciju bitno veći od troškova računanja što rezultira paralelnom inačicom programa koja je sporija od slijedne. U slučaju simulacija koje su u istom pozivu funkcije više puta računale informacijsku dobit dobiveni su bolji rezultati. Ovakvu simulaciju može se opravdati time što se informacijska dobit u više navrata računa kod izgradnje stabla odlučivanja, a broj računanja ovisi o broju atributa (dubina stabla) i o različitim vrijednostima svakog atributa (broj novih grana na danoj dubini). Osim toga, kod klasifikatora koji se temelje na klasifikatoru stabla odlučivanja, kao što je primjerice metoda slučajnih šuma (eng. random forests), gradi se velik broj stabala ( >100, 300, 400). Zbog toga su realni slučajevi u kojima se unutar istog poziva funkcije informacijska dobit može računati i nekoliko stotina puta. Ovdje smo simulirali računanje informacijske dobiti u 500 i 3000 puta. Na takvim paralelnim programima dobivena su ubrzanja od oko 1,5 do 2 puta. Zanimljivo je za uočiti kako manji broj procesora (2) za ovaj slučaj daje otprilike isto ili bolje ubrzanje od većeg broja procesora (4). Odnosno, vrijeme utrošeno na rad po svakom procesu je bitno manje.

Konačno, može se zaključiti kako paralelizacija programa ima smisla onda kada su troškovi računanja bitno veći od komunikacijskih. Inače može doći do izostanka poboljšanja ili čak do pogoršanja vremena izvođenja programa. Osim toga nije nužno uvijek poželjan što veći broj procesa jer se može dogoditi da se uvedu dodatni komunikacijski troškovi uz manja računska poboljšanja. U svakom slučaju svaki problem vezan uz paralelizaciju mora se prethodno analizirati i razmotriti te na kraju testirati kako bi se ispitale optimalne okolnosti izvođenja jer one ovise o arhitekturi, MPI implementaciji te o izvedbi drugih funkcija koje se koriste što dovodi do toga da često nisu analitički i teoretski očite.

# 6. Zaključak

Za paralelizacijom programa u R-u postoji realna potreba zbog prirode zadataka za čije je rješavanje on posebno namijenjen (statistička obrada podataka). Osim toga, često se radi o metodama koje je relativno lako paralelizirati. Razmatranjem dostupnih programskih paketa za paralelizaciju programa, dolazi se do zaključka kako su rješenja koja pretpostavljaju arhitekturu grozda trenutno najprihvatljivija. Grozd naspram višejezgrenih i višeprocesorskih računala za jeftiniju cijenu pruža veći broj dostupnih CPU čvorova. S druge strane, naspram spleta, grozd je tehnički lakše izvedivo rješenje trenutno. R biblioteke namijenjene paralelnom izvođenju programa na grozdu su snow, Rmpi, Rpvm, papply, biopara i dr. Među njima se po mogućnostima i optimiziranosti izdvaja snow biblioteka sa svojim ekstenzijama snowFT i snowfall.

Osim dostupnih biblioteka, kod paralelizacije preporučljivo je koristiti dijelove brzog koda napisane u C-u. R je jezik visoke razine i zbog toga je znatno sporiji u odnosu na C kod, ali dostupni su relativno jednostavni načini omotavanja C funkcija R kodom. Jednostavan primjer paralelizacije u ovom seminaru prikazan je uz pomoć funkcije papply, a uključuje prikaz omotavanja C funkcija te njihovo paralelno pozivanje iz R sučelja.

# 7. Literatura

[1] Doc. dr. sc. Domagoj Jakobović, Predavanja iz kolegija Paralelno programiranje, veljača 2009.

[2] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, Ulrich Mansmann, State-of-the-art in Parallel Computing with R Technical Report Number 47, Department of Statistics University of Munich, svibanj 2009.

[3] The R Project for Statistical Computing, http://www.r-project.org/, travanj 2009.

# 8. Sažetak

R je programski paket otvorenog koda, a uključuje programski jezik i razvojnu okolinu. Posebno je namijenjen statističkom programiranju te ima izuzetne mogućnosti grafičkog prikaza rješenja i karakteristika podatkovnih skupova. Osim toga, poput Matlaba R je objektno-orijentirani, skriptni jezik visoke razine.

Potreba za paralelnim programiranjem u R-u nameće se zbog prirode statističkih algoritama koji često dugo traju (jer statistika ima smisla samo kad je uključen dovoljno velik, reprezentativan skup uzoraka), a trivijalni su za paralelizaciju (jer su operacije koje se obavljaju često podatkovno ili funkcionalno nezavisne).

U R-u je dostupan veći broj paketa za paralelizaciju koda, za višejezgrena računala, za grozd i za splet računala. Za grozd, kao optimalno rješenje nameće se MPI standard i snow biblioteka. Osim primjene paralelne komunikacije, R kod koji se paralelizira poželjno je pisati u C-u zbog brzine izvođenja. Na kraju seminara dan je jednostavan primjer C koda koji je omotan R funkcijom i paraleliziran u R-u.