Informe

Hillary Cañas, Eric Bellet & Jean Garcia

PARTE 1: PRODUCTO MATRIZ-VECTOR

La función productmv recibe los siguientes parámetros:

#-matriz(string): ruta del archivo que contiene la matriz(cuadrada) a multiplicar, #por ejemplo: '. . . /tblAkv10x10.csv'.

#-vector(string): ruta del archivo que contiene el vector a multiplicar, #por ejemplo: . . . /tblxkv10.csv'.

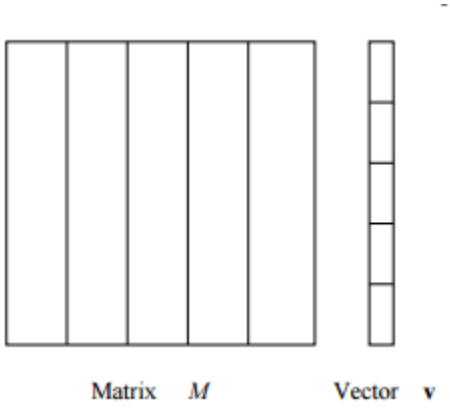
#-N(int): dimensión asociada a la multiplicación, por ejemplo: 10.

#-memorylimit(int): tamaño límite (en bytes) que puede ocupar en la sesión de r #donde ejecutará su función, por ejemplo: 480.

Cuando el memorylimit es -1, se trabaja con toda la memoria de la maquina. En caso contrario, se trabaja la multiplicacion con un manejo de memoria.

El manejo de memoria se encarga de dividir el vector y la matriz en pequeños trozos o pedazos a los cuales llamamos 'chunks', de acuerdo al tamaño de memoria disponible para luego operar con la matriz.

Según el profesor Jeffrey D. Ullman de la Universidad de Stanford[1] se pueden ver los 'chunks' de la siguiente forma:



```
chunks <- function(A, x, N, chunksF, indice){
  #*******************************************************
  #Division de chunks utilizando la memoria.
  #*******************************************************

  #tamres es el tamano necesario para guardar un resultado.
  tamres <- object.size(max(A[,ncol(A)]) * max(x[,2]))

  #tamvector es el size de un valor del vector.
  tamvector <- object.size(x[N,])
  #Acumula la cantidad de memoria que se puede utilizar
  acum <- 0
  #Cuenta cuantas operaciones se pueden hacer
  cont <- 0
  for (i in indice:((indice + chunksF)-1)) {

    acum <- object.size(A[i,]) + tamres + tamvector + acum

    if (acum > memoria){

      break
    }
    #Cuento cuantos valores de la matriz puedo utilizar.
    cont <- cont + 1

  }
  return(cont)
}
```

En la función map se opera el producto de una fila de la matriz por el vector. Los resultados se guardan en un archivo temporal. De esta manera, no se ocupa espacio en memoria ram.

```
#*************************************************************************
*****
#                                MAP
#*************************************************************************
*****
map <- function(A, x, N, i, j, cont){
  #i<-7 #4= 5-1,8-2=6,11-3=8 ,
  for (k in i:((i+cont)-1)) {
    resultado <- A[k, ncol(A)] * x[j, ncol(x)]
    write.table(resultado, file = "tmp/archivotemporal.csv", row.names = FALS
E,
                col.names = FALSE, sep = ",", append = T)

  }
```

```
}
```

Al finalizar la operación anterior se libera la memoria. Con esto se asegura que se tiene suficiente memoria para el reduce.

```
#Liberamos memoria para hacer el reduce
  remove(A)
  remove(x)
```

El reduce se encarga de sumar los valores que se producen en el map para asi entregar la respuesta final.

```
#**************************************************************************
*****
#                                  REDUCE
#**************************************************************************
*****
reduce <- function(N){
  z <- read.csv("tmp/archivotemporal.csv", header = FALSE)
  m <- matrix(0,nrow = N)
  #Calculamos cuantos reducers se necesitan
  reducers <- 1:N

  for (i in 1:N) {
    k <- reducers[i]
    for (h in 1:N) {
      m[i] <- z[k,1] + m[i]
      k <- k + N
    }
  }
  remove(z)
  return(m)

}
```

PARTE 2: PRODUCTO MATRIZ-MATRIZ

La función productmm recibe los siguientes parámetros:

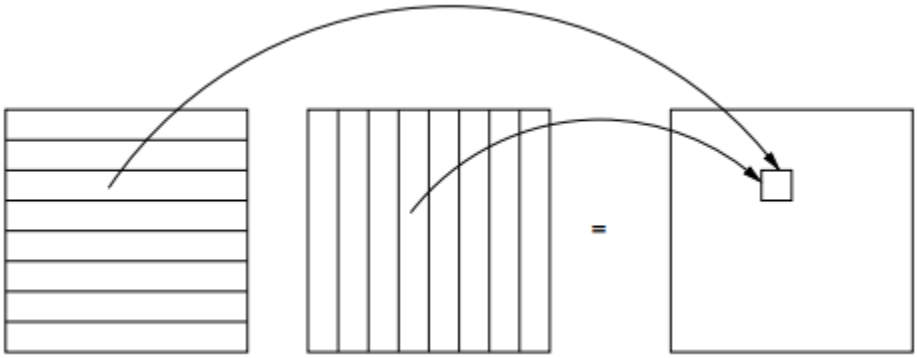
#-matrizA(string): ruta del archivo que contiene la matriz(cuadrada) a multiplicar, #por ejemplo: '. . . /tblAkv10x10.csv'.

#-matrizB(string): ruta del archivo que contiene la matriz(cuadrada) a multiplicar, #por ejemplo: '. . . /tblAkv10x10ident.csv'.

#-N(int): dimension asociada a la multiplicación, por ejemplo: 10.

#-límite de memoria(int): tamaño límite (en bytes) que puede ocupar en la sesión de r #donde ejecutará su función, por ejemplo: 480.

Se aplicó una técnica llamanda "lower-bound" para multiplicar matrices en una sola pasada. Aqui el reducer corresponde a un solo elemento del la matriz final. Primero agrupamos las filas y columnas en bandas. Cada par consiste en una banda de filas de una matriz y una banda de columnas de la otra matriz, esto es utilizado por el reducer para producir un cuadro de elementos de la matriz. Se observa en la siguiente figura:



Al igual que en el caso matriz-vector, se realiza un manejo de memoria por 'chunks', que son las bandas previamente mencionadas.

```
#*************************************************************************
*****
#Calculamos cuantas operaciones podemos hacer utilizando la memoria indicada.
#*************************************************************************
*****
chunks <- function(A, B, N, chunksF, indice){
  #******************************************************
  #Division de chunks utilizando la memoria.
  #******************************************************

  #tamres es el tamano necesario para guardar un resultado.
  tamres <- (object.size(max(A[,ncol(A)]) * max(B[,ncol(B)])))*N

  #tamcol es el size de la matriz
  tamcol <- object.size(B)
```

```
  #Acumula la cantidad de memoria que se puede utilizar
  acum <- 0
  #Cuenta cuantas operaciones se pueden hacer
  cont <- 0
  for (i in indice:((indice + chunksF)-1)) {

    acum <- object.size(A[i,])*N  + tamres  + tamcol + acum

    if (acum > memoria){

      break
    }
    #Cuento cuantos valores  puedo utilizar.
    cont <- cont + 1

  }
  return(cont)
}
```

Se multiplican las filas y columnas y los datos se almacenan en un archive temporal. Para que luego sean utilizados por el reduce.

```
#*************************************************************************
*****
#                                 MAP
#*************************************************************************
*****
map <- function(chunk){
  resultado <- unlist(chunk[1]) * unlist(chunk[2])
  write.table(resultado, file = "tmp/archivotemporal.csv", row.names = FALSE,
              col.names = FALSE, sep = ",", append = T)

}
```

Al finalizar la operación anterior se libera la memoria.

```
#Liberamos memoria para hacer el reduce
  remove(A)
  remove(x)
```

En el reduce se lee el archive temporal para sumar los resultados de los vectores que se obtuvieron en el map y así dar la solución final

```
#**************************************************************************
*****
#                                   REDUCE
#**************************************************************************
*****
reduce <- function(N){
  z <- read.csv("tmp/archivotemporal.csv", header = FALSE)
  m <- matrix(0,nrow = N, ncol = N)
  #Calculamos cuantos reducers se necesitan
  reducers <- 1:N
  k <- 1
  j <- 1
  for (i in 1:N) {
    for (g in 1:N) {
      if (j == (N+1)){
        j <- 1
      }

      for (h in 1:N) {
        m[i, j] <- z[k,1] +  m[i, j]
        k <- k + 1

      }
      j <- j + 1
    }
  }
  remove(z)
  return(m)

}
```

Referencia bibliográfica:

[1]Jure Leskovec, Anand Rajaraman & Jeff Ullman, “The Mining of Massive Datasets,” Chapter 2, pp. 20–71 http://infolab.stanford.edu/~ullman/mmds/ch2.pdf