pbdR: Input, PCA, and Movies

George Ostrouchov

NIMBioS Workshop, April 5-7, 2014

The pbdR Core Team

Wei-Chen Chen[1]
George Ostrouchov[2,3]
Pragneshkumar Patel[3]
Drew Schmidt[3]

pbdR
Programming with Big Data in R

Support

This work used resources of National Institute for Computational Sciences at the University of Tennessee, Knoxville, which is supported by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center. This work also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

[1] Department of Ecology and Evolutionary Biology
University of Tennessee, Knoxville TN, USA

[2] Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge TN, USA

[3] Joint Institute for Computational Sciences
University of Tennessee, Knoxville TN, USA

Contents

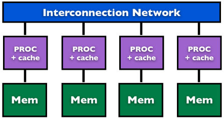
1. Hardware and Software Landscape
2. Data Input
3. Principal Components Analysis For Spatio-Temporal Data
4. Plot Ensembles in Parallel
5. Rearranging ddmatrix Data

Contents

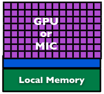
1. Hardware and Software Landscape
   - Quick Overview of Parallel Hardware
   - A Quick Overview of Parallel Software
   - pbdR Connects R to HPC Libraries

Three Basic Flavors of Hardware

Distributed Memory



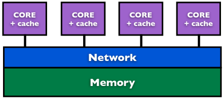
Co-Processor



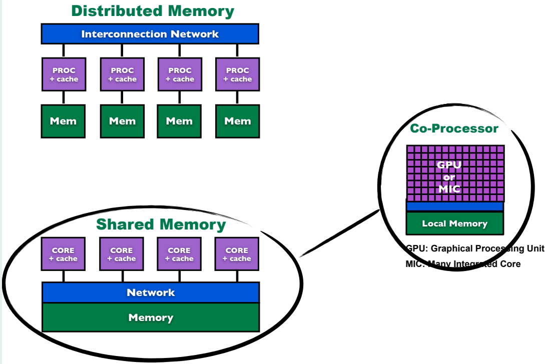
GPU: Graphical Processing Unit

MIC: Many Integrated Core

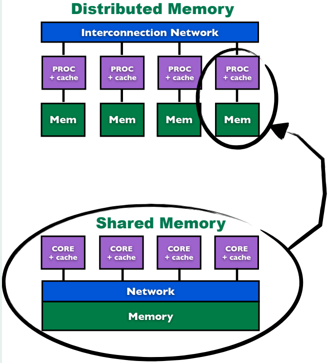
Shared Memory

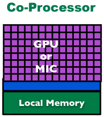


Your Laptop or Desktop



A Server or Cluster

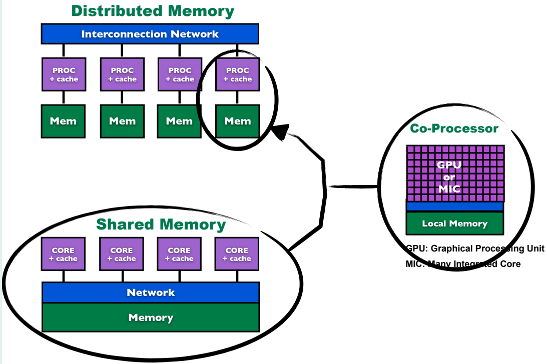




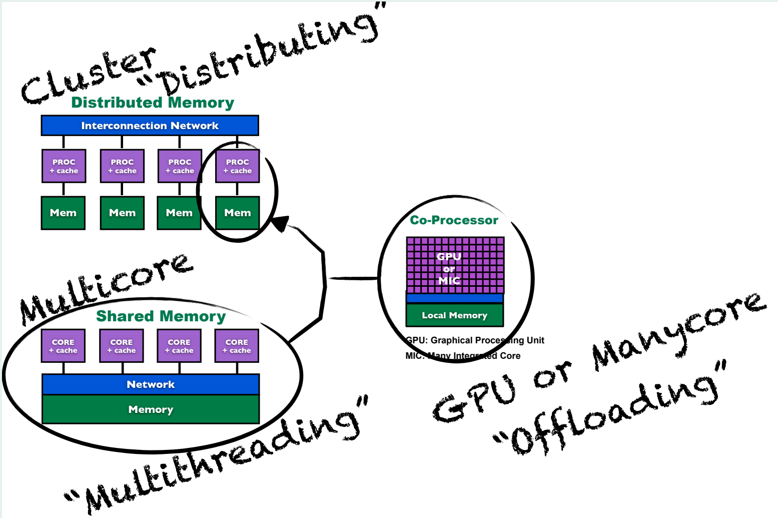
GPU: Graphical Processing Unit

MIC: Many Integrated Core

Server to Supercomputer



Knowing the Right Words



Distributed Memory

Interconnection Network

PROC + cache | PROC + cache | PROC + cache | PROC + cache

Mem | Mem | Mem | Mem

Shared Memory

CORE + cache | CORE + cache | CORE + cache | CORE + cache

Network

Memory

Co-Processor

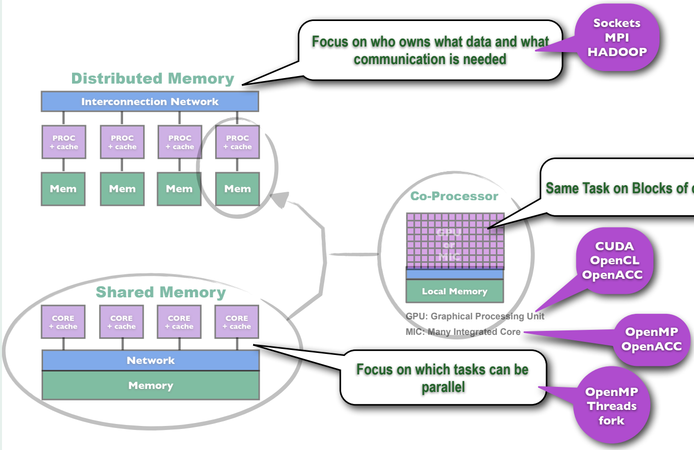
GPU or MIC

Local Memory

GPU: Graphical Processing Unit

MIC: Many Integrated Core

“Native” Programming Models and Tools



Focus on who owns what data and what communication is needed

Sockets
MPI
HADOOP

Same Task on Blocks of data

CUDA
OpenCL
OpenACC

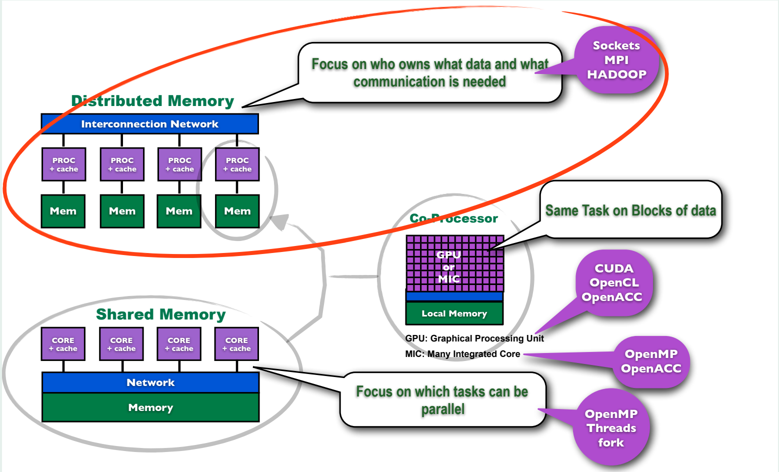
GPU: Graphical Processing Unit
MIC: Many Integrated Core

OpenMP
OpenACC

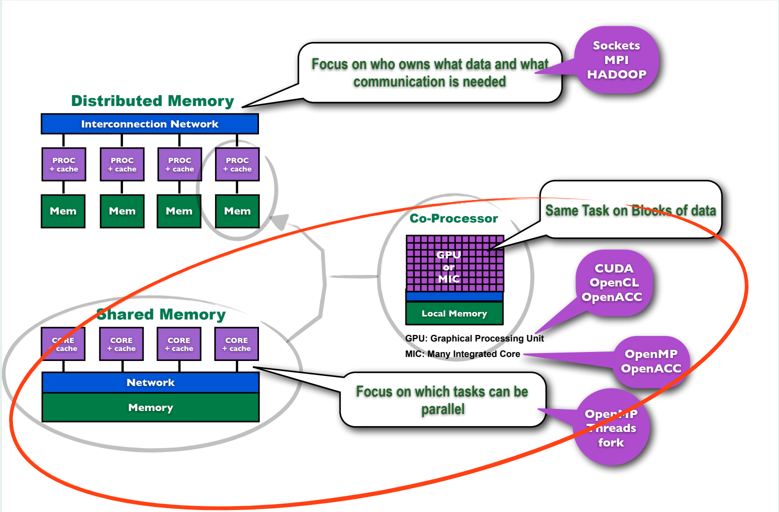
Focus on which tasks can be parallel

OpenMP
Threads
fork

30+ Years of Parallel Computing Research



Last 10 years of Advances



Focus on who owns what data and what communication is needed

Sockets
MPI
HADOOP

Same Task on Blocks of data

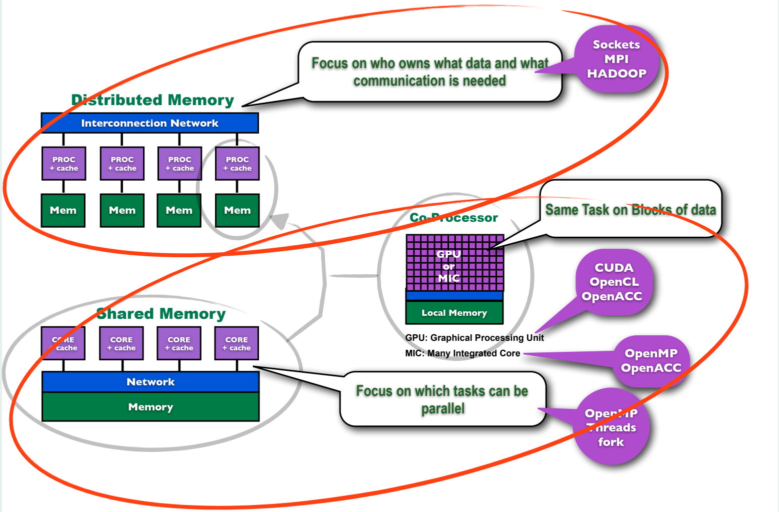
CUDA
OpenCL
OpenACC

OpenMP
OpenACC

Focus on which tasks can be parallel

OpenMP
Threads
fork

Putting It All Together Challenge



Focus on who owns what data and what communication is needed

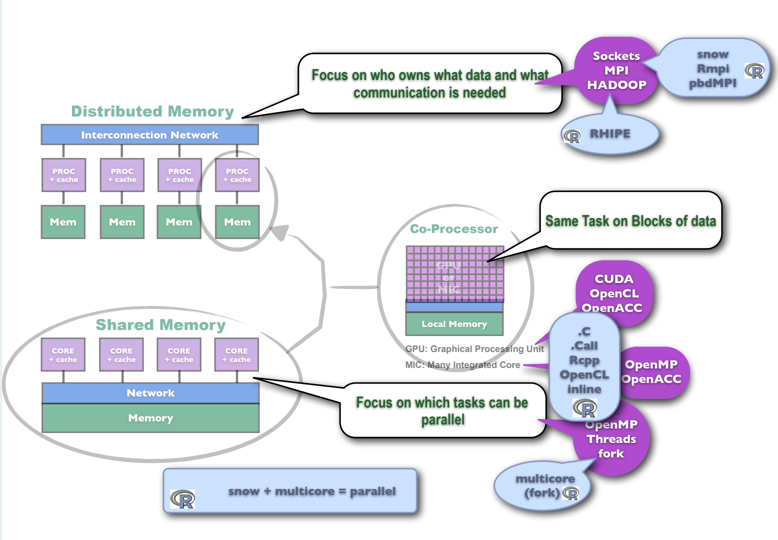
Same Task on Blocks of data

GPU: Graphical Processing Unit

MIC: Many Integrated Core

Focus on which tasks can be parallel

R Interfaces to Native Tools



Focus on who owns what data and what communication is needed

Sockets
MPI
HADOOP

snow
Rmpi
pbdMPI

RHIPE

Distributed Memory

Interconnection Network

PROC + cache

PROC + cache

PROC + cache

PROC + cache

Mem

Mem

Mem

Mem

Same Task on Blocks of data

Co-Processor

GPU or MIC

Local Memory

GPU: Graphical Processing Unit

MIC: Many Integrated Core

CUDA
OpenCL
OpenACC

.C
.Call
Rcpp
OpenCL
inline

OpenMP
OpenACC

Shared Memory

CORE + cache

CORE + cache

CORE + cache

CORE + cache

Network

Memory

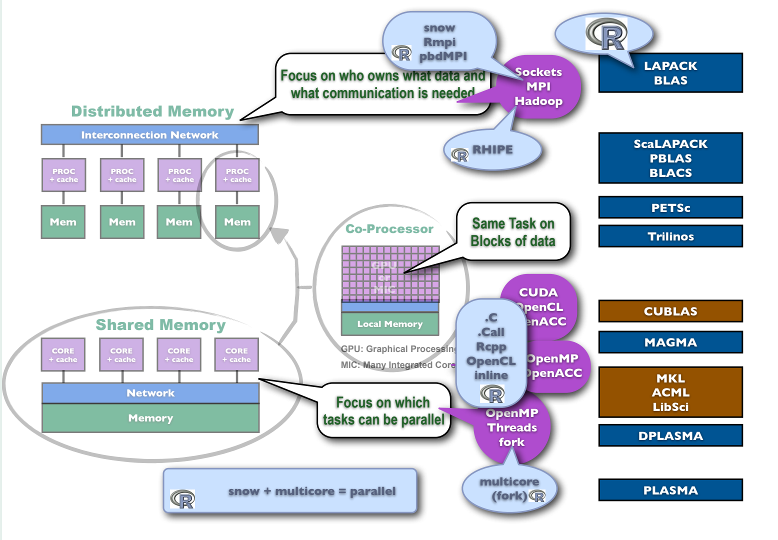
Focus on which tasks can be parallel

OpenMP
Threads
fork

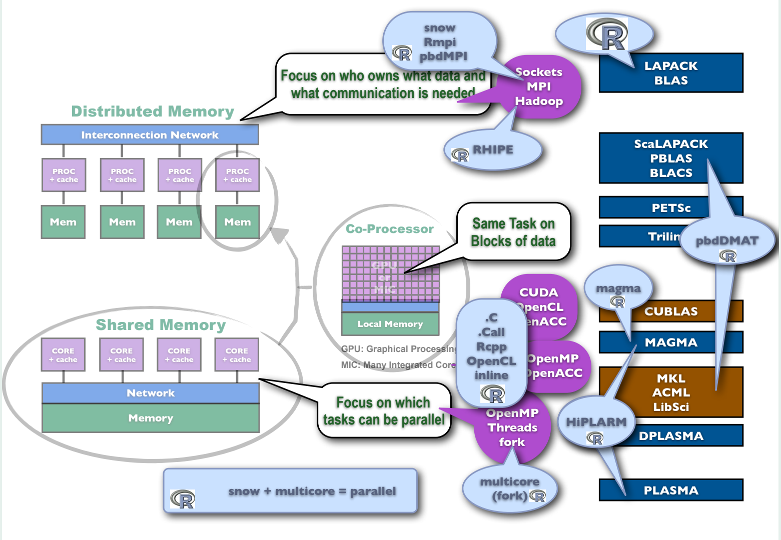
multicore
(fork)

snow + multicore = parallel

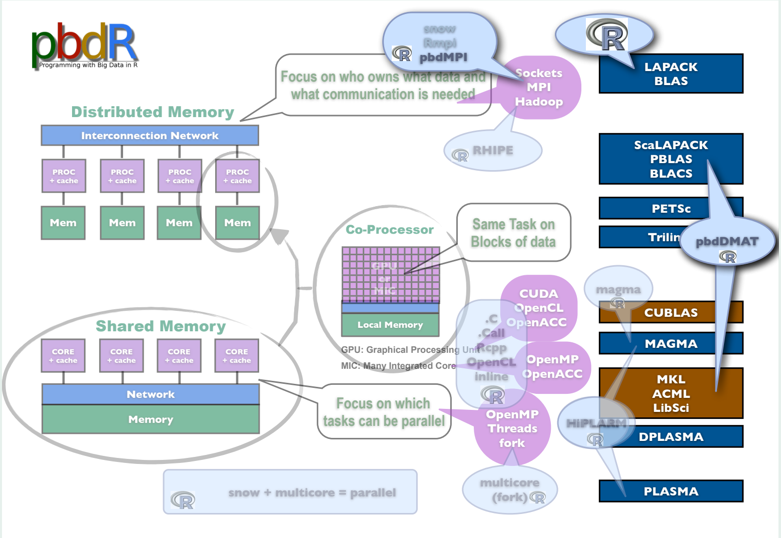
HPC Libraries



R Interfaces to Scalable HPC Libraries



pbdR Interfaces to Scalable HPC Libraries



pbdR
Programming with Big Data in R

Focus on who owns what data and what communication is needed

Distributed Memory

Interconnection Network

PROC + cache PROC + cache PROC + cache PROC + cache

Mem Mem Mem Mem

snow
Rmpi
pbdMPI

Sockets
MPI
Hadoop

RHIPE

LAPACK
BLAS

ScaLAPACK
PBLAS
BLACS

PETSc

Trilinos

pbdDMAT

Co-Processor

GPU or MIC

Local Memory

GPU: Graphical Processing Unit
MIC: Many Integrated Core

Same Task on Blocks of data

CUDA
OpenCL
OpenACC

.C
.Call
Rcpp
OpenCL
inline

OpenMP
OpenACC

magma

CUBLAS

MAGMA

MKL
ACML
LibSci

HiPLARM

DPLASMA

PLASMA

Shared Memory

CORE + cache CORE + cache CORE + cache CORE + cache

Network

Memory

Focus on which tasks can be parallel

OpenMP
Threads
fork

multicore (fork)

snow + multicore = parallel

Separate manual: http://r-project.org/

- scan()
- read.table()
- read.csv()
- . . .
- readBin()
- ncvar_get()
- readSocket()

No parallel file system: Read Serial then Distribute

read.csv()

```r
library(pbdDMAT)
if(comm.rank() == 0) { # only read on process 0
  x <- read.csv("myfile.csv")
} else {
  x <- NULL
}

dx <- as.ddmatrix(x)
```

New Issues

- How to read in parallel?
- CSV, SQL, NetCDF4, HDF, ADIOS, custom binary
- How to partition data across nodes?
- How to structure for scalable libraries?
- Read directly into form needed or restructure?
- . . .
- A lot of work needed here!

CSV Data

Serial Code

```
d <- read.csv(‘‘x.csv’’)
```

Parallel Code 0_readcsv.r

```
library(pbdDEMO, quiet = TRUE)
init.grid()
dx <- read.csv.ddmatrix("x.csv", header=TRUE,
            sep=’’,’’, nrows=10, ncols=10,
            num.rdrs=2, ICTXT=0)
comm.print(dx)
finalize()
```

NetCDF4 Data

Parallel Read

```
### Must determine who will read what portion(s) and how to assemble them before reading

### parallel read after determining st and co
nc <- nc_open_par(file.name)

nc_var_par_access(nc, "TREFHT")
new.X.gbdc <- ncvar_get(nc, "TREFHT", start = st, count = co)
nc_close(nc)

finalize()
```

Parallel Data Input

Binary Data

Read subcube

```
library(pbdDMAT, quiet = TRUE)
init.grid()

data.dim <- c(2048, 2048, 2048) # full data dimension
g.start <- c(1, 1, 513)         # global subcube corner
g.dim <- c(64, 64, 1024)        # global subcube extent

my.start <- g.start + c(0, 0, comm.rank()*my.dim[3])
my.dim <- g.dim / c(1, 1, comm.size())

size <- 4 # file is single precision floats

vx <- block3d.read(‘‘filename’’, data.dim, my.start, my.dim, size)

## local reshape dimensions
my.nrow <- prod(my.dim[1:2])
my.ncol <- my.dim[3]
ldim <- c(my.nrow, my.ncol)

## global reshape dimensions
g.nrow <- prod(g.dim[1:2])
g.ncol <- g.dim[3]
gdim <- c(g.nrow, g.ncol)

## reshape local
X <- matrix(vx, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)

## glue local pieces into a ddmatrix
X <- new("ddmatrix", Data=X, dim=gdim, ldim=ldim, bldim=ldim, ICTXT=1)

## transform to 2d block cyclic
X <- redistribute(X, bldim=c(8,8), ICTXT=0)
```

Binary Data

3d Block Binary Reader

```
block3d.read <- function(file, data.dim, my.start,
    my.dim, size=4) {
  con.x <- file(file, "rb", blocking=TRUE)

  start <- sum((my.start - 1) * c(1,
      cumprod(data.dim)[-length(data.dim)]))

  x <- rep(NA, prod(my.dim))

  block <- 1:my.dim[1]

  for(j in 1:my.dim[3]) {
    sofar <- 0
    for(i in 1:my.dim[2]) {
      seek(con.x, where=start, rw="read", origin="start")
      x[block] <- readBin(con=con.x, what="numeric",
          n=my.dim[1], size=size)
      block <- block + my.dim[1]

      start <- start + data.dim[1]*size
      sofar <- sofar + data.dim[1]*size
    }
    start <- start - sofar + data.dim[1]*data.dim[2]*size
  }

  close(con.x)
  x
}
```

Contents

3 Principal Components Analysis For Spatio-Temporal Data

- Principal Components Analysis for Spatio-Temporal Data
- Principal Components Analysis

Example 1: Empirical Orthogonal Functions

- $m \times n$ matrix X at n spatial locations and m times
- X_c is the column centered X
- $X_c = VDU^T$
- $VD = X_c U$ columns are n time series
- $UD = X_c^T V$ columns are m images
- VD and UD have same units as X

Empirical Orthogonal Functions in Climate Analysis

- Computation and volume rendering of large-scale EOF coherent modes in rotating turbulent flow data, AGU Fall Meeting, December 2013

Coherent Modes in Turbulent Flow

Get and Redistribute the Data

```r
library(pbdDMAT, quiet = TRUE)
init.grid()

## load local data (file assumes 4 processors!)
g.dim <- c(64, 64, 1024)
my.dim <- g.dim / c(1, 1, comm.size())
save.file <- paste("xyz.RData", comm.rank(), sep="") # assumes 4 processors!
load(save.file)

## reshape 3d array into a matrix for PCA (EOF) computation
## first two dimensions become rows and third becomes columns
## local reshape dimensions
my.nrow <- prod(my.dim[1:2])
my.ncol <- my.dim[3]
ldim <- c(my.nrow, my.ncol)

## global reshape dimensions
g.nrow <- prod(g.dim[1:2])
g.ncol <- g.dim[3]
gdim <- c(g.nrow, g.ncol)

## now reshape local
X <- matrix(vx, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)
Y <- matrix(vy, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)
Z <- matrix(vz, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)

## glue local pieces into a ddmatrix
X <- new("ddmatrix", Data=X, dim=gdim, ldim=ldim, bldim=ldim, ICTXT=1)
Y <- new("ddmatrix", Data=Y, dim=gdim, ldim=ldim, bldim=ldim, ICTXT=1)
Z <- new("ddmatrix", Data=Z, dim=gdim, ldim=ldim, bldim=ldim, ICTXT=1)

## transform to 2d block cyclic
X <- redistribute(X, bldim=c(8,8), ICTXT=0)
Y <- redistribute(Y, bldim=c(8,8), ICTXT=0)
Z <- redistribute(Z, bldim=c(8,8), ICTXT=0)
```

Coherent Modes in Turbulent Flow

Compute PCA and do Scree Plot (0_pca.r)

```r
E <- sqrt(X^2 + Y^2 + Z^2) # energy from velocity
E.pca <- prcomp(x=E, retx=TRUE, scale=FALSE)

# plot using one process
if(comm.rank() == 0)
  {
    ## scree plot for first 50 components
    variance <- E.pca$sdev^2  # note: all own sdev
    proportion <- variance[1:50]/sum(variance)
    cumulative <- cumsum(proportion)
    component <- 1:length(proportion)
    png("scree.png")
    plot(component, cumulative, ylim=c(0,1))
    points(component, proportion, type="h", col="blue")
    dev.off()
  }

finalize()
```

Contents

4 Plot Ensembles in Parallel
- Parallel Plot Ensembles

How can we plot in parallel?

- Several plots, one or more on each processor (can do now)
- One plot by several processors (need to rewrite graphics)

Parallel Plot Ensembles

Plots in parallel

png.slice

```
png.slice <- function(x, g.dim, lab="slice", title=lab,
    work.dir="", zero.center=TRUE, most.positive=TRUE)
{
  X <- array(as.vector(x), dim=g.dim)

  ## prepare zero centered topo.colors
  if(zero.center)
    {
       . . .
    }
  else
    zlim <- range(X)

  ## set most positive (for unique up to sign)
  if(most.positive)
    {
       . . .
    }

  ## make png file
  file <- paste(work.dir, lab, "-r", comm.rank(),
      ".png", sep="")
  png(file)
  image(x=1:g.dim[1], y=1:g.dim[2], z=X,
      col=topo.colors(40), useRaster=TRUE, asp=1,
      xlim=c(1, g.dim[1] + 1), ylim=c(1, g.dim[2] + 1),
      zlim=zlim)
  title(title)
  ret <- dev.off()
  invisible(ret)
}
```

Parallel Plot Ensembles

Plots in parallel

Get and Redistribute the Data

```r
library(pbdDMAT, quiet = TRUE)
init.grid()

## set global and local dimensions
g.dim <- c(64, 64, 1024)
my.dim <- g.dim / c(1, 1, comm.size())

save.file <- paste("xyz.RData", comm.rank(), sep="")
load(save.file)     # gets vx vector

## reshape 3d array into a matrix
## first two dimensions become rows and third becomes columns

## local reshape dimensions
my.nrow <- prod(my.dim[1:2])
my.ncol <- my.dim[3]
ldim <- c(my.nrow, my.ncol)

## global reshape dimensions
g.nrow <- prod(g.dim[1:2])
g.ncol <- g.dim[3]
gdim <- c(g.nrow, g.ncol)

## now reshape local
X <- matrix(vx, nrow=my.nrow, ncol=my.ncol, byrow=FALSE)

## glue local pieces into a ddmatrix
X <- new("ddmatrix", Data=X, dim=gdim, ldim=ldim, bldim=ldim, ICTXT=1)

## transform to 2d block cyclic
X <- redistribute(X, bldim=c(8,8), ICTXT=0)

## Plot few columns in parallel
   . . .
finalize()
```

Plots in parallel

Make comm.size() plots in parallel

```
step <- 5
max.plots <- min(20, ncol(X) %/% step)
last.plot <- 1 - step
time <- comm.timer(
for(i in 1:max.plots)
    {
        now.plots <- last.plot + step*(1:comm.size())
        my.col <- gather.col(X[, now.plots])
        lab <- paste("col", lead0(now.plots[comm.rank()
            + 1]), sep="")
        png.slice(my.col, g.dim[1:2], lab)
        last.plot <- now.plots[length(now.plots)]
    }
)
```

Plots in parallel

gather.col First Attempt (1_plot.r)

```r
gather.col <- function(x, num=min(ncol(x), comm.size()))
{
    ## gather complete columns of a global array to different ranks
    my.local <- as.vector(x[, comm.rank() + 1], proc.dest=comm.rank())
    my.local
}
```

Plots in parallel

gather.col Second Attempt (2_plot.r)

```
gather.col <- function(x, num=min(ncol(x), comm.size()))
{
  ## gather complete columns of a global array to different ranks
  my.local <- NULL
  for(i in 1:num)
    {
      ## serial collection of unique data to each rank
      local <- as.vector(x[, i])
      if(comm.rank() + 1 == i) my.local <- local
    }
  my.local
}
```

Plots in parallel

gather.col The Right Way (3_plot.r)

```
gather.col <- function(x, num=min(ncol(x), comm.size()))
{
  ## gather complete columns of a global array to different ranks
  x.num <- x[, 1:num]
  x.num <- as.colblock(x.num)

  ## ScaLAPACK fix (a future release will automate)
  if(ownany(x.num))
    ret <- as.vector(submatrix(x.num))
  else
    ret <- NULL
  ret
}
```

Plots in parallel

Now Plot the PCA Components (4_plot.r)

```
E <- sqrt(X^2 + Y^2 + Z^2)

E.pca <- prcomp(x=E, retx=TRUE, scale=FALSE)

## Use ranks 1 to n.pca to plot individual components in
    parallel
n.pca <- min(comm.size(), g.nrow)
my.col <- gather.col(E.pca$x, num=n.pca)

if(!is.null(my.col))
  {
    ## component plots on rank 1 to n.pca
    lab <- paste("pc", comm.rank(), sep="")
    title <- paste(lab, "sigma^2 =",
        variance[comm.rank() + 1])
    png.slice(my.col, g.dim[1:2], lab, title=title,
        work.dir=work.dir)
  }
```

Parallel Plot Ensembles

Exercise: scripts/pbdDMAT/dmat_app

- Experiment with scripts 0_pca.r, 1_plot.r, 2_plot.r, 3_plot.r, 4_plot.r

Contents

Rearranging Data

Simple Redistributions

- as.block(dx, square.bldim = TRUE)
- as.rowblock(dx)
- as.colblock(dx)
- as.rowcyclic(dx, bldim = .BLDIM)
- as.colcyclic(dx, bldim = .BLDIM)
- as.blockcyclic(dx, bldim = .BLDIM)

BLACS context (Processor Grid)

- init.grid(P,Q)
- .ICTXT = 0 gives P x Q
- .ICTXT = 1 gives PQ x 1
- .ICTXT = 2 gives 1 x PQ

Exercise: scripts/pbdDMAT/dmat_app

- Experiment with scripts 5ictxt.r, 6_ictxt.r, and 7_ictxt.r
- Experiment with other redistributions

The pbdR Project

- Our website: http://r-pbd.org/
- Email us at: RBigData@gmail.com
- Our google group: http://group.r-pbd.org/

Where to begin?

- The **pbdDEMO** package
  http://cran.r-project.org/web/packages/pbdDEMO/
- The **pbdDEMO** Vignette: http://goo.gl/HZkRt

Thanks for coming!

Questions?

pbdR

Programming with Big Data in R

http://r-pbd.org/