Data Science with R and pbdR at ORNL: From the CADES Cloud to the OLCF

Part 2: pbdR and the OLCF

Drew Schmidt and George Ostrouchov
6/18/2018

Parallel Hardware

Three Flavors of Hardware

Three Flavors of Hardware

Three Flavors of Hardware

Three Flavors of Hardware

Three Flavors of Hardware

A Bit of Cluster History...

Commodity Cluster Before 2003

HPC Introduces Diskless Compute Nodes ~2003

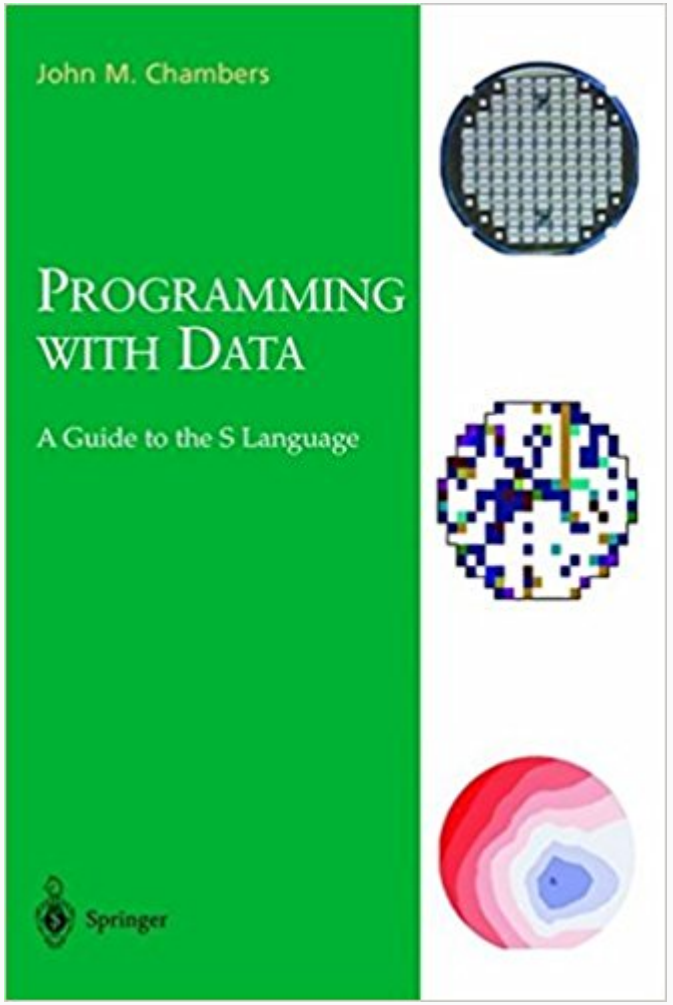
Disk Comes Back as SSD ~2010

Working with Today's HPC Systems

pbdR

What is it?

- An acronym
  - *Programming with Big Data in R*
  - *Parallel Big Data R*
  - *Pretty Bad for Dyslexics*
- A set of R packages
- Core Team: Wei-Chen Chen, George Ostrouchov, Drew Schmidt



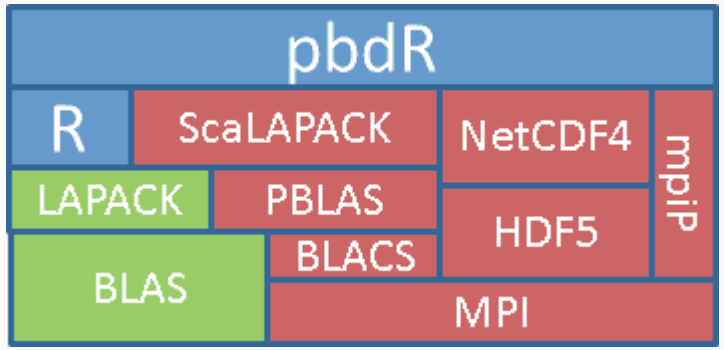
pbdR

Strive for *Productivity, Portability, Performance*

- Bridge high-performance computing with high-productivity of R language
- Keep syntax *identical* to R, when possible.
- Software reuse philosophy:
  - Don't reinvent the wheel when possible
  - Introduce HPC standards with R flavor
  - Use scalable HPC libraries with R convenience
- Simplify and use R intelligence where possible

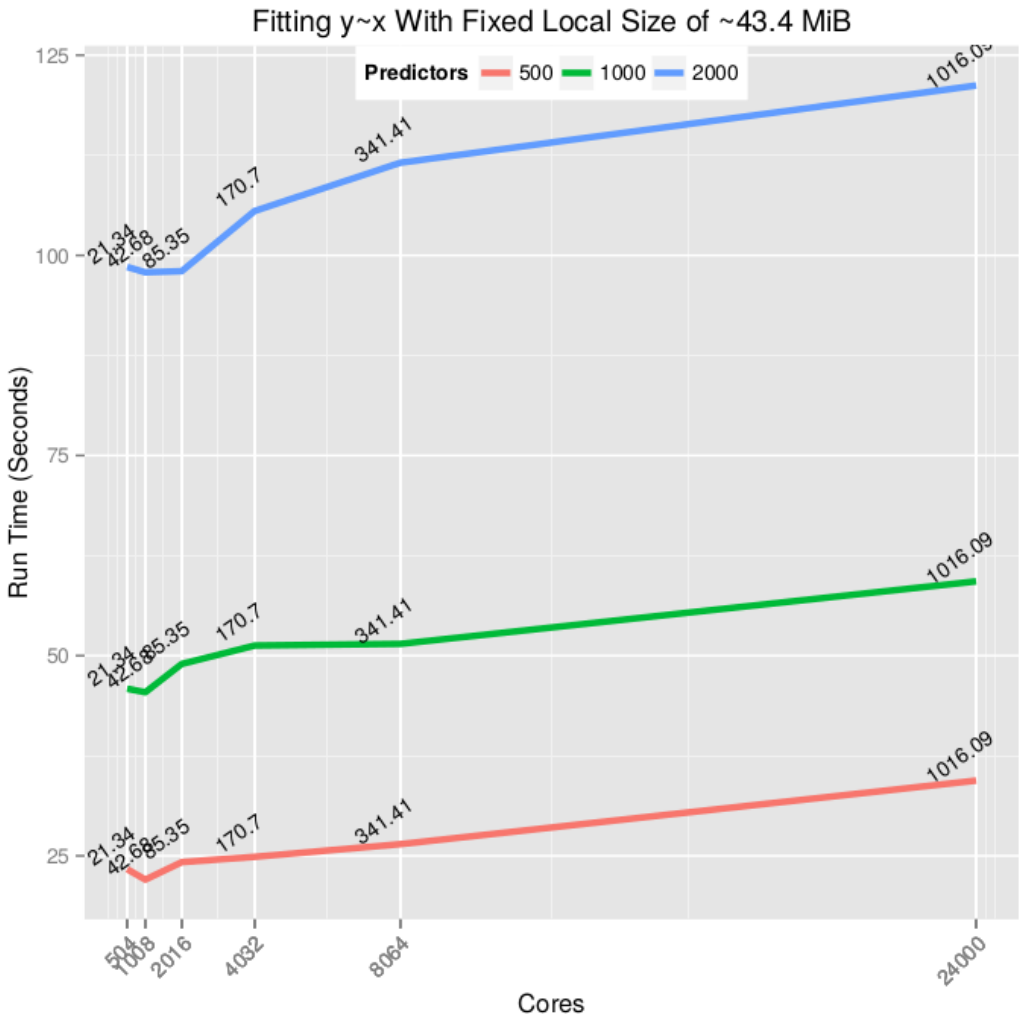
pbdR

- MPI packages
  - pbdMPI
  - pbdSLAP, pbdBASE, pbdDMAT, pbdML, pmclust
  - kazaam
  - tasktools
- Communication tools
  - pbdZMQ
  - remoter
  - pbdCS
- Profilers
  - pbdPROF
  - pbdPAPI
  - hpcvis
- I/O packages
  - pbdIO
  - pbdNCDF4
  - pbdADIOS
  - hdfio (soon)



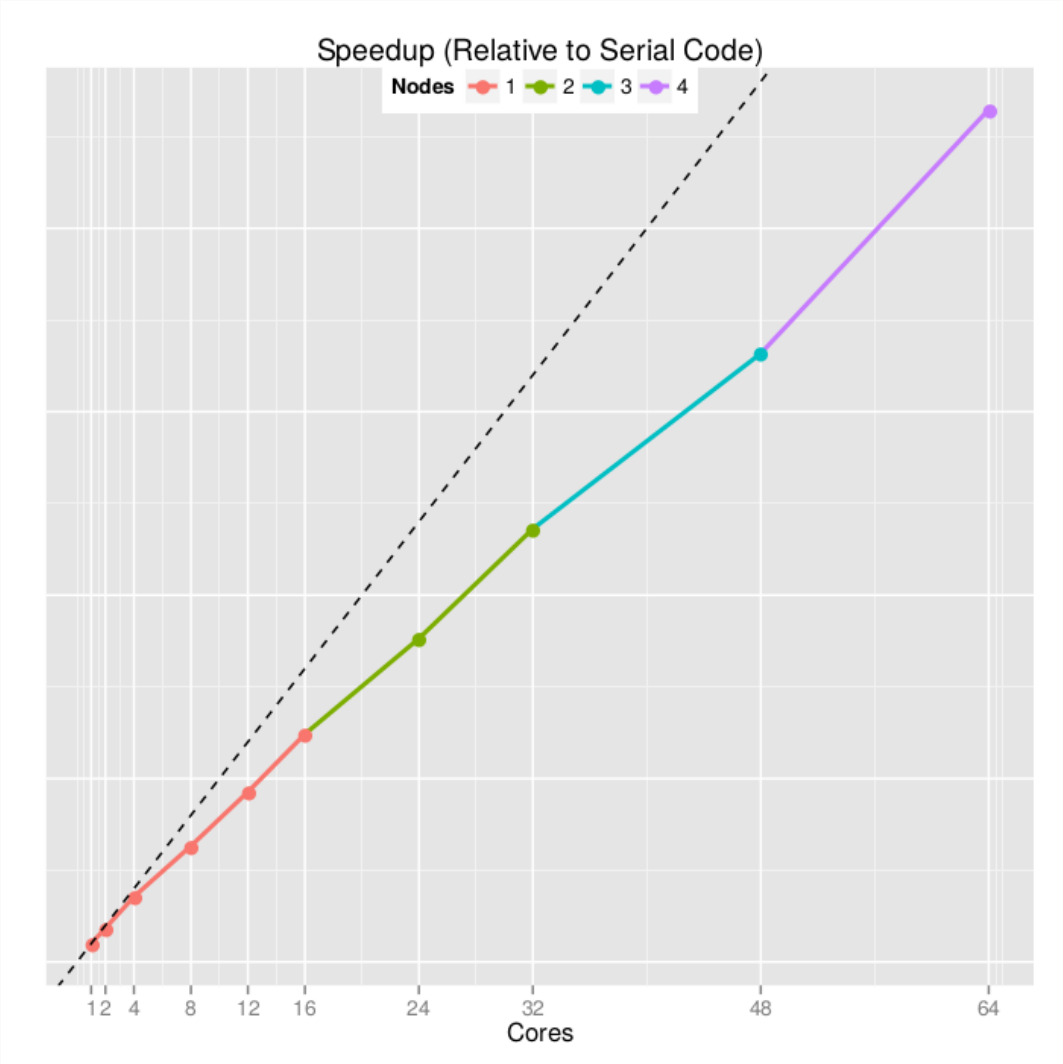
HPC Libraries and Their R/pbdR Connections

Least Squares Benchmark




```
library(pbdDMAT)
init.grid()

x = ddmatrix("rnorm", nrow=m, ncol=n)
y = ddmatrix("rnorm", nrow=m, ncol=1)
mdl = lm.fit(x=x, y=y)

finalize()
```

Matrix Exponentiation Benchmark




```
library(pbdDMAT)
init.grid()

dx = ddmatrix("rnorm", 5000, 5000)
expm(dx)

finalize()
```

Other (distributed) HPC Packages for R

- Rmpi
- A handful of hadoop/spark packages
- That's about it...

Rmpi vs pbdMPI

- Rmpi can be used interactively. pbdMPI is batch (without the client/server)
- pbdMPI often easier to install
- pbdMPI has simpler syntax

Rmpi

```
# int
mpi.allreduce(x, type=1)
# double
mpi.allreduce(x, type=2)
```

pbdMPI

```
allreduce(x)
```

Types in R

```
typeof(1)
```

```
## [1] "double"
```

```
typeof(2)
```

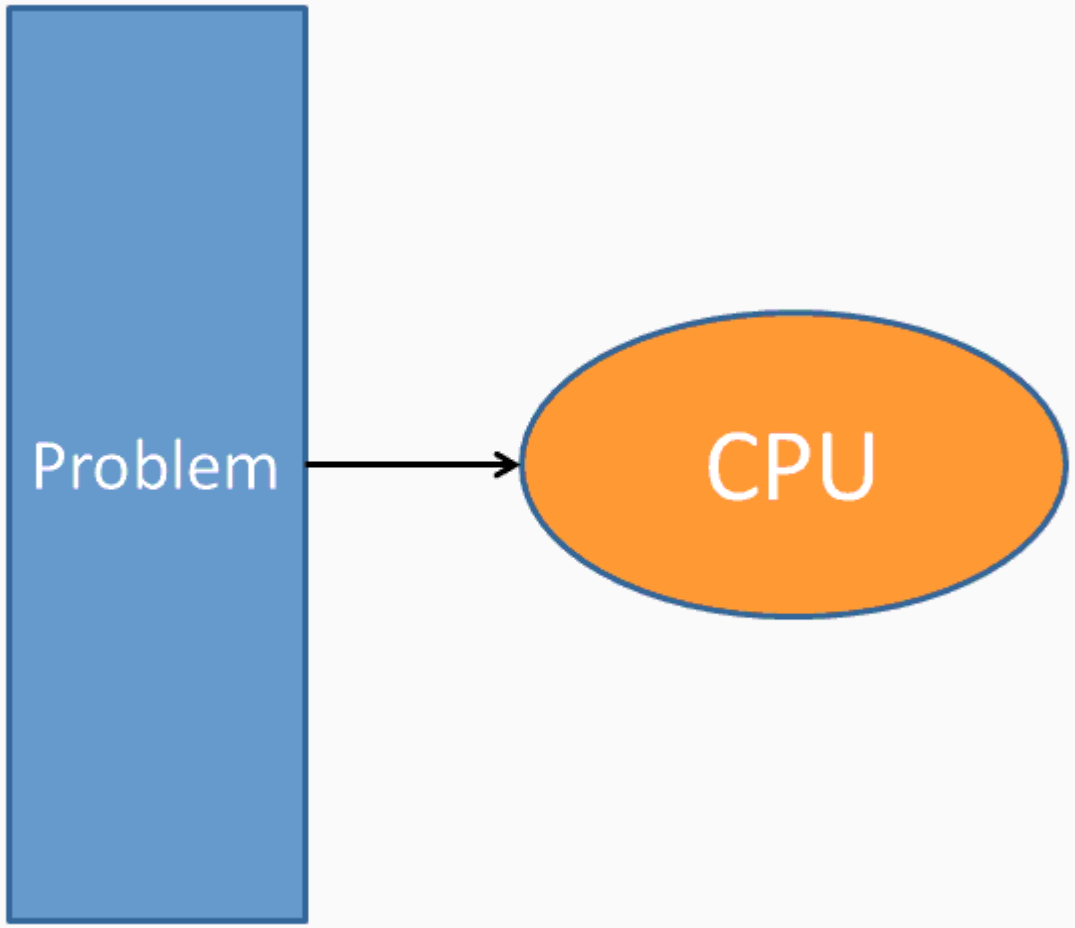
```
## [1] "double"
```

```
typeof(1:2)
```

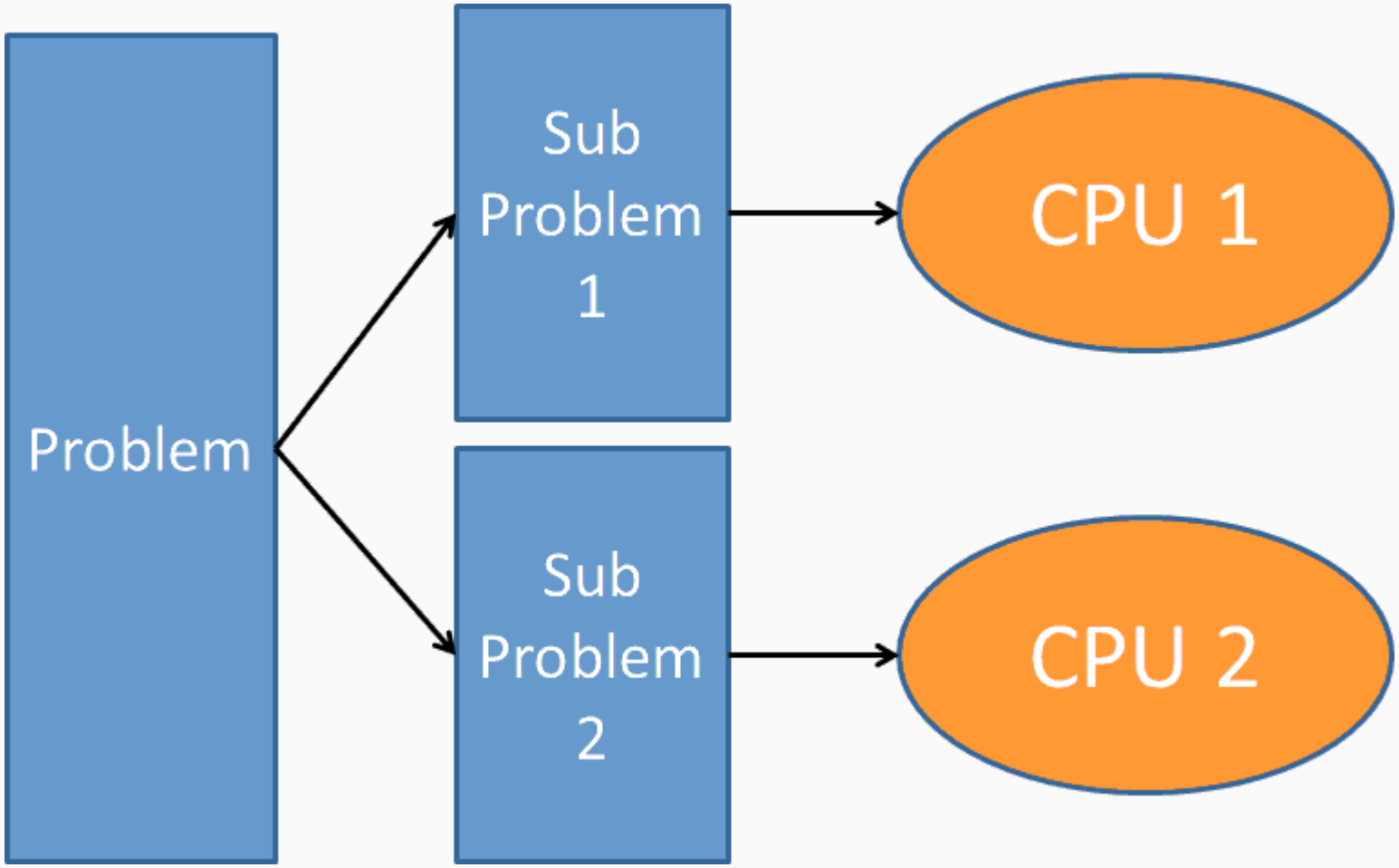
```
## [1] "integer"
```

MPI with pbdMPI

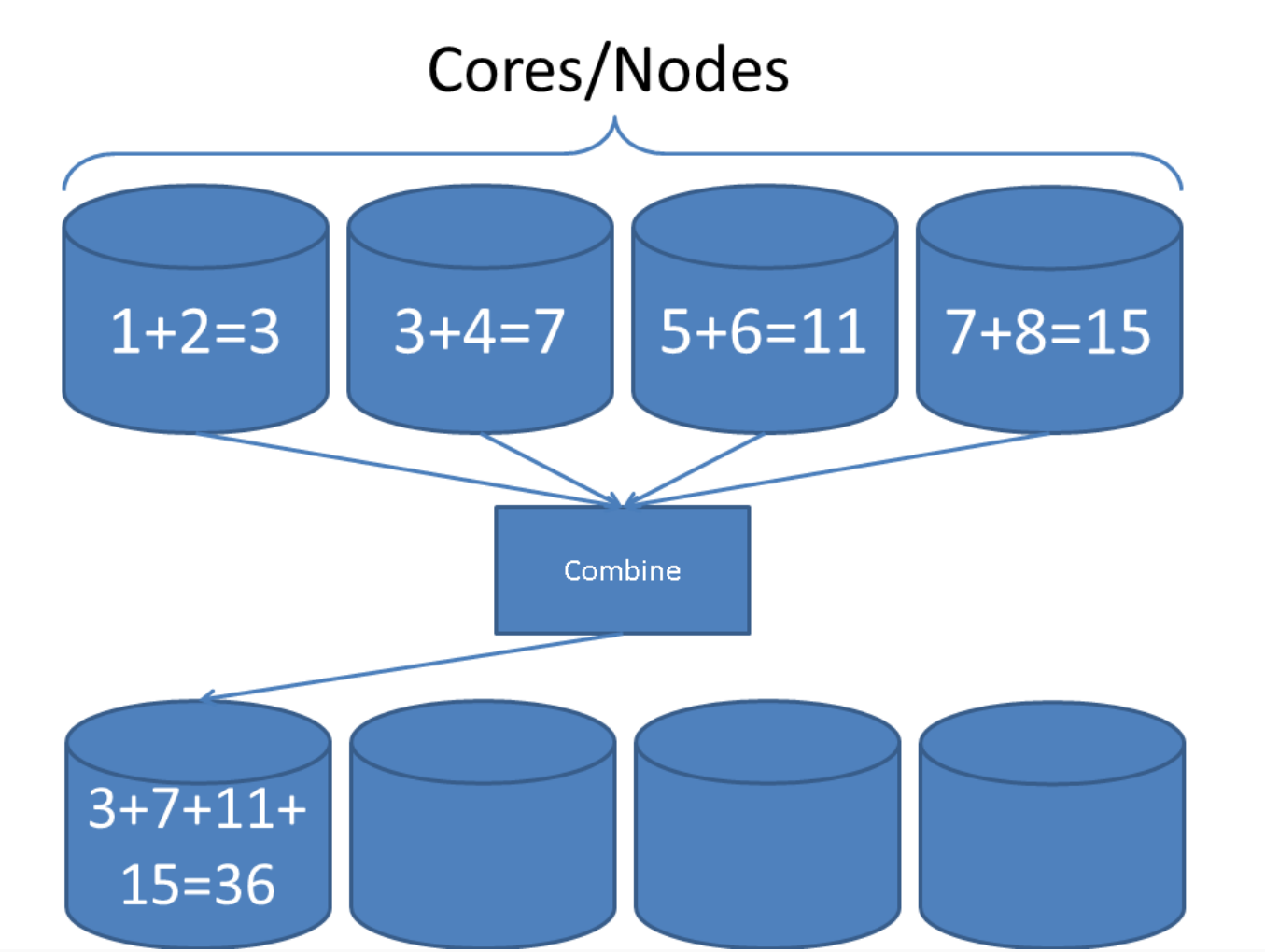
Parallelism



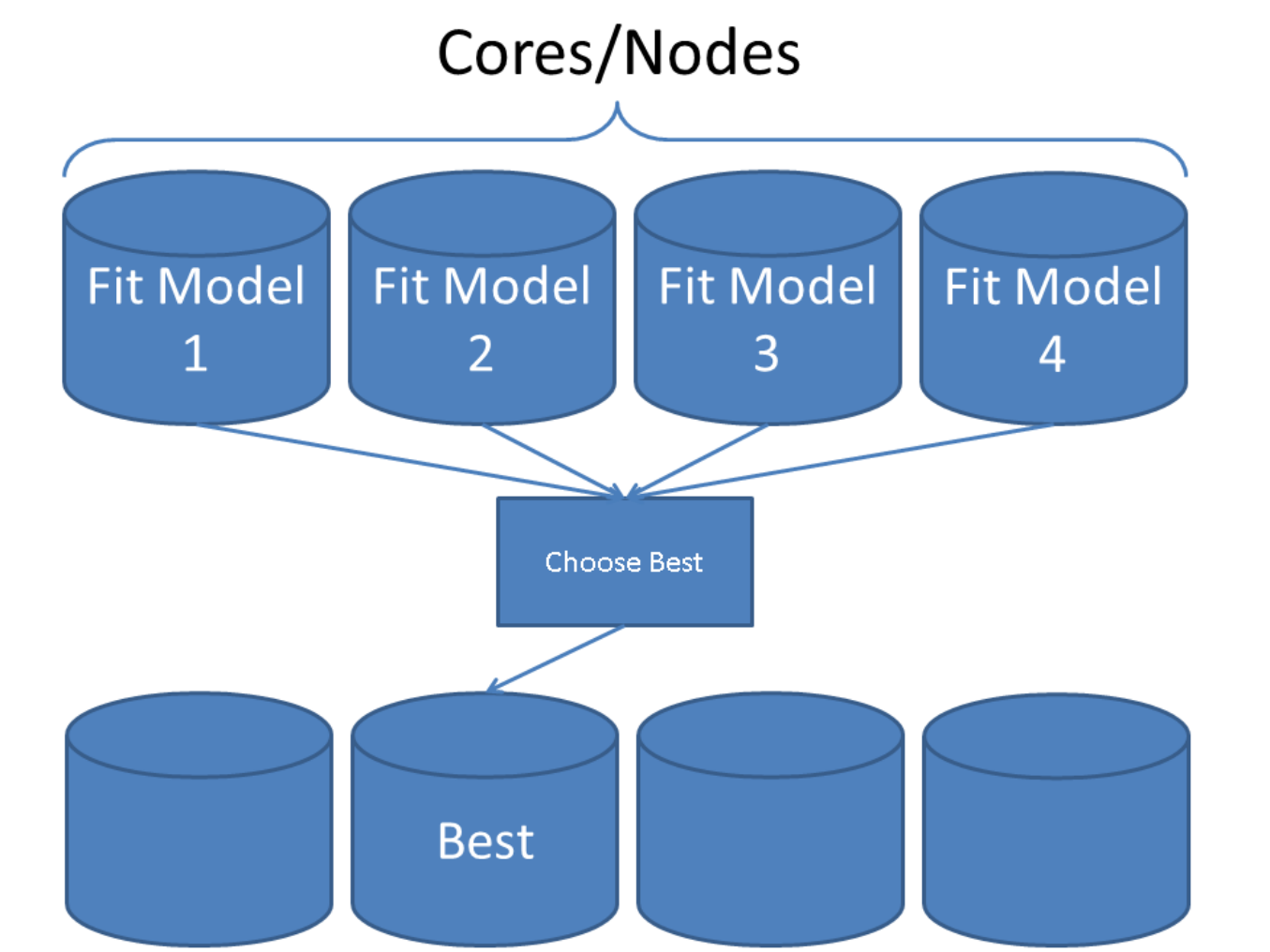
Parallelism



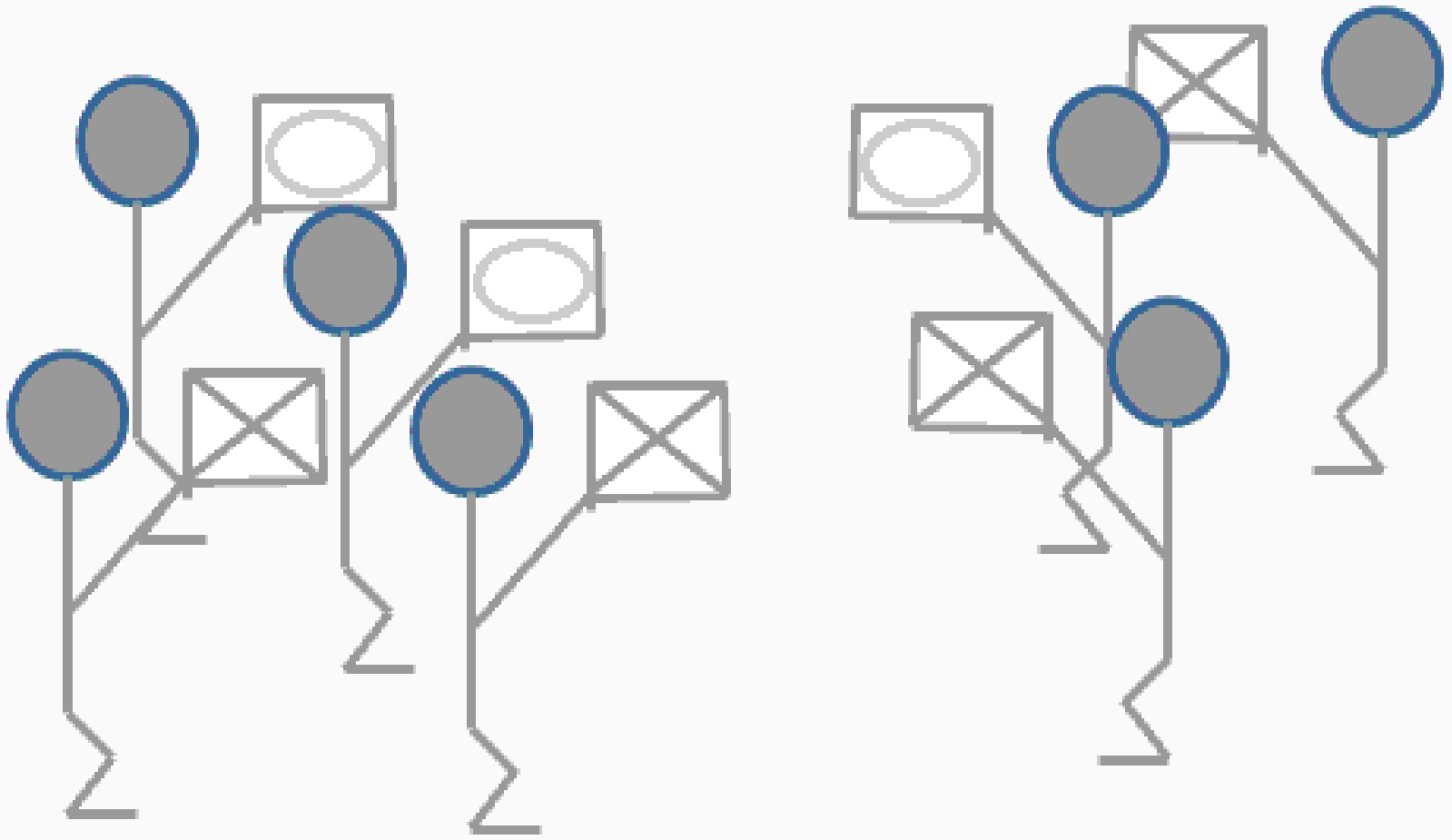
Parallelism



Parallelism



MPI Operations: Reduce



MPI Operations: Reduce

```
library(pbdMPI)

nranks = comm.size()
ret = allreduce(1)
comm.print(nranks)
comm.print(ret)

finalize()
```

MPI Operations: Broadcast

MPI Operations: Broadcast

```
library(pbdMPI)

if (comm.rank() == 0){
  important_value = 1+1
} else {
  important_value = NULL
}

ret = bcast(important_value)
comm.print(ret, all.rank=TRUE)

finalize()
```

MPI Operations: Gather

MPI Operations: Gather

```
library(pbdMPI)

val.local = comm.rank()
vals = gather(val.local)
comm.print(vals)

finalize()
```

MPI Operations: Barrier

MPI Operations: Barrier

```
library(pbdMPI)

comm.print("starting huge computation...")

if (comm.rank() == 0){
  Sys.sleep(5)
}

barrier()
comm.print("ok!")

finalize()
```

Task Parallelism

tasktools

- Tools for task-based parallelism.
- Has an `lapply()`-like interface.
- Automatically handles input-checkpointing:
  - Have thousands of "jobs"
  - Run as many as you can in 2 hour run window
  - Keep running job until all tasks eventually complete.
- Can be used as a workflow tool for external programs.

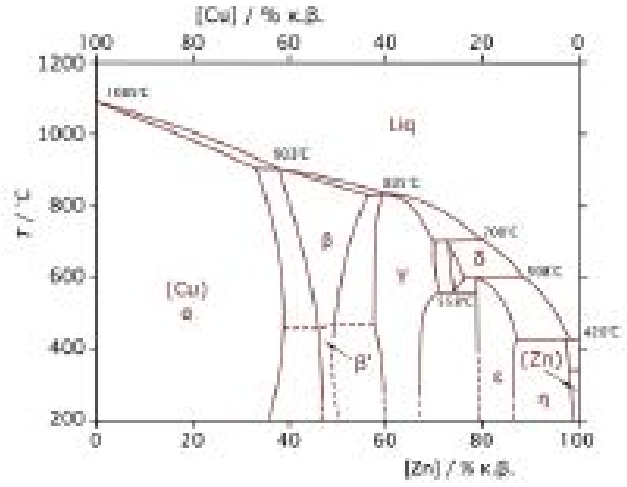
tasktools

```
costly = function(x, waittime)
{
  Sys.sleep(waittime)
  cat(paste("iter", i, "executed on rank", comm.r

  sqrt(x)
}

ret = mpi_napply(10, costly, checkpoint_path="/tr
comm.print(unlist(ret))
```

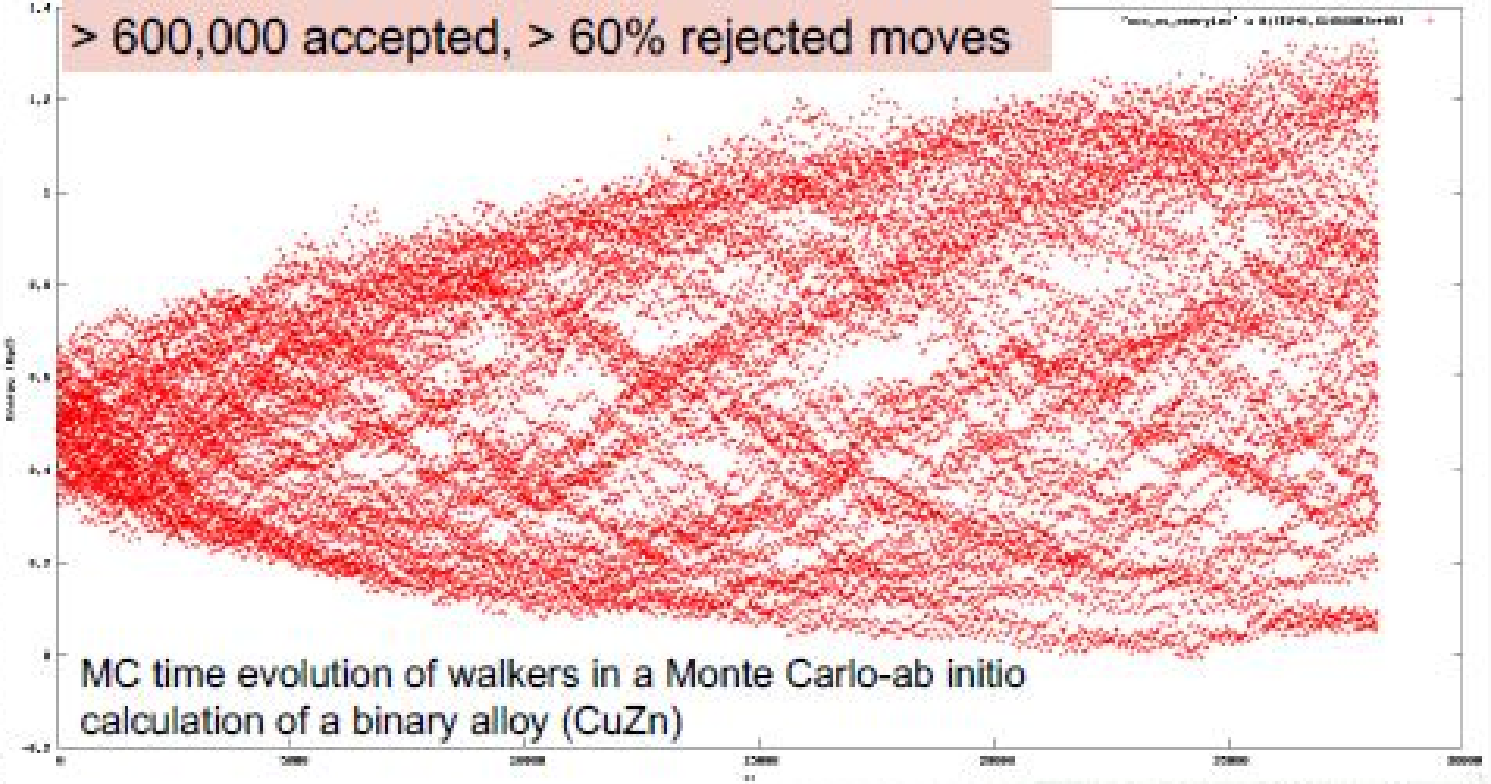
```
$ mpirun -np 3 r mpi_napply.r
iter 4 executed on rank 1
iter 7 executed on rank 2
iter 1 executed on rank 0
^Citer 2 executed on rank 0
iter 8 executed on rank 2
iter 5 executed on rank 1

$ mpirun -np 3 r mpi_napply.r
iter 9 executed on rank 2
iter 3 executed on rank 0
iter 6 executed on rank 1
iter 10 executed on rank 2

 [1] 1.000000 1.414214 1.732051 2.000000 2.23606
 [5] 2.449490 2.645751 2.828427 3.000000 3.16227
```

tasktools

Material Science



Each dot is a 250-atom DFT calculation !!
> 600,000 accepted, > 60% rejected moves



MC time evolution of walkers in a Monte Carlo-ab initio calculation of a binary alloy (CuZn)

3 Accelerating Stochastic Sampling Methods by On-Line Data Analy

Titan

- Cray XK7
- 18,688 nodes
- 299,008 cores
- 693.5 TiB of RAM

tasktools

Parameters

```
eta_set = c(0.01, 0.05, 0.1, 0.5, 1)
gamma_set = 0:3
max_depth_set = c(6, 10, 15)
min_child_weight_set = c(1, 3, 5)

combos = expand.grid(eta=eta_set, gamma=gamma_se
NROW(combos)
```

```
## [1] 180
```

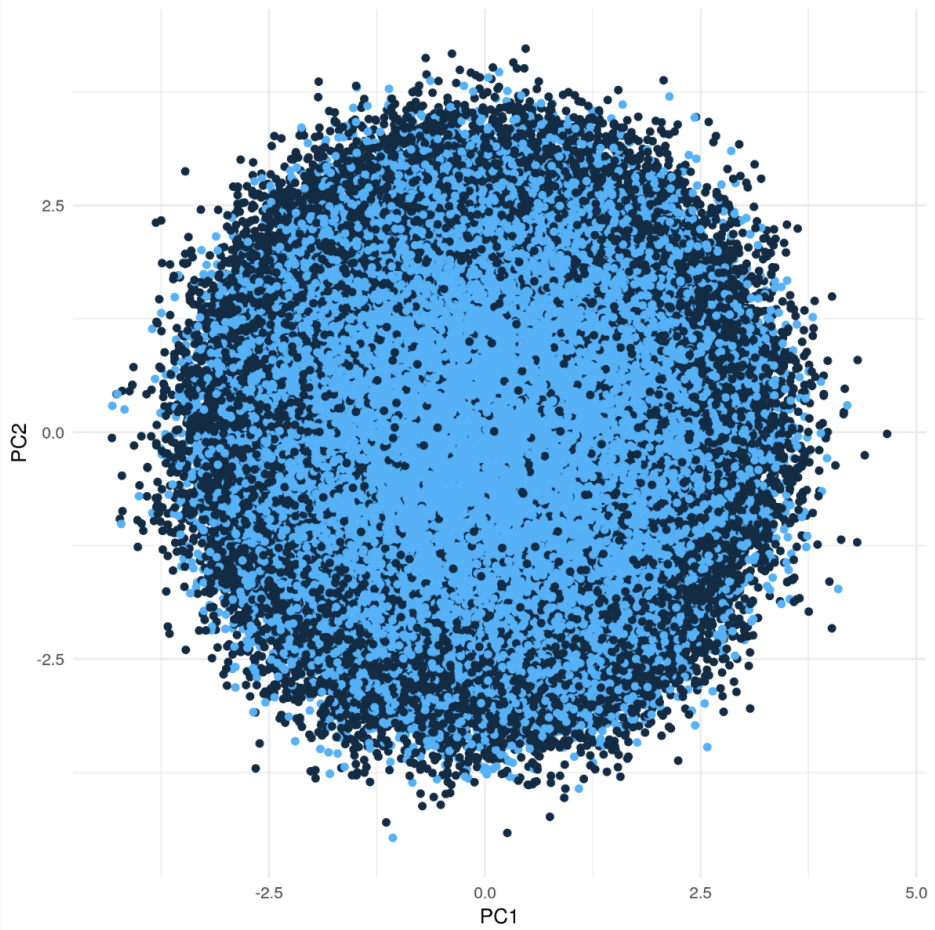
Launch

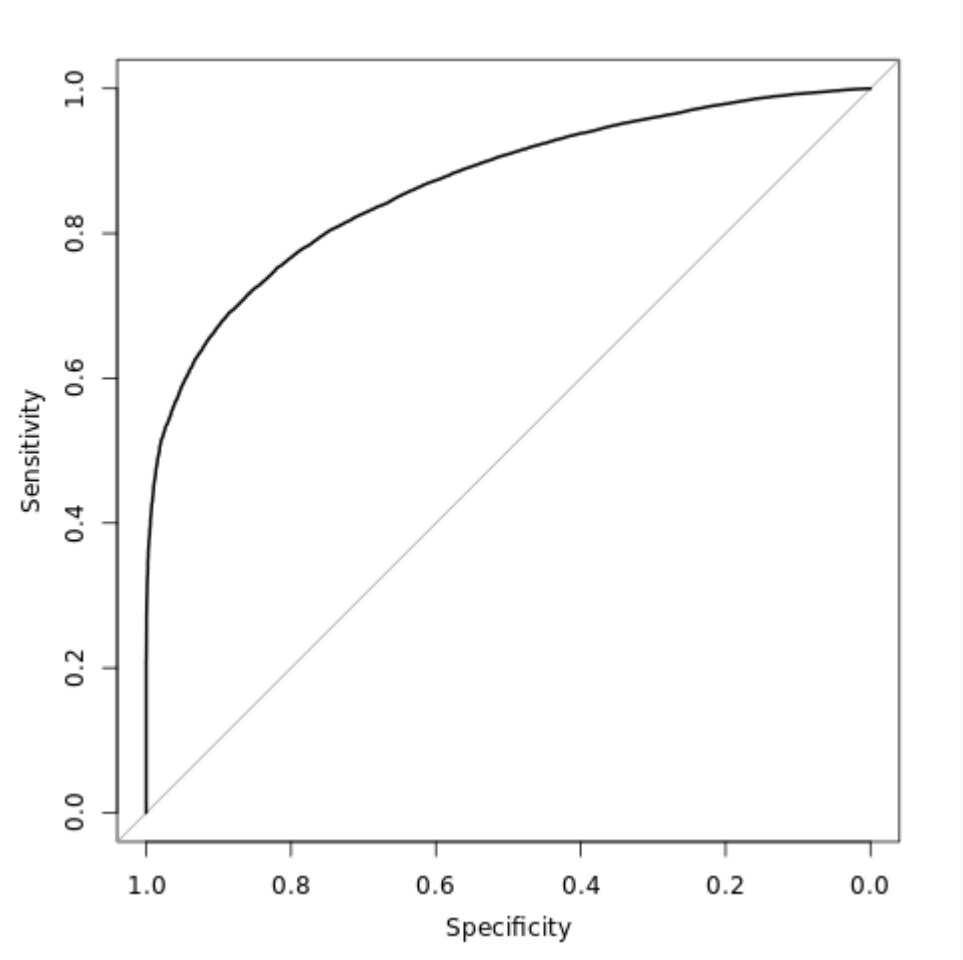
```
aprun -n 30 -d 16 xgb.r
```

Script

```
run_one_cv = function(i)
{
  eta = combos[i, 1]
  gamma = combos[i, 2]
  max_depth = combos[i, 3]
  min_child_weight = combos[i, 4]

  params = list(...)
  cv = xgb.cv(params=params, ...)
  it = which.max(cv$evaluation_log$test_auc_mean
  best = cv$evaluation_log[it]

  list(params=combos[i, , drop=FALSE], rating=be
}

results = mpi_napply(n, run_one_cv, checkpoint_p
```

tasktools


2.5
0.0
-2.5
PC2
-2.5
0.0
2.5
5.0
PC1


1.0
0.8
0.6
0.4
0.2
0.0
Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
Specificity

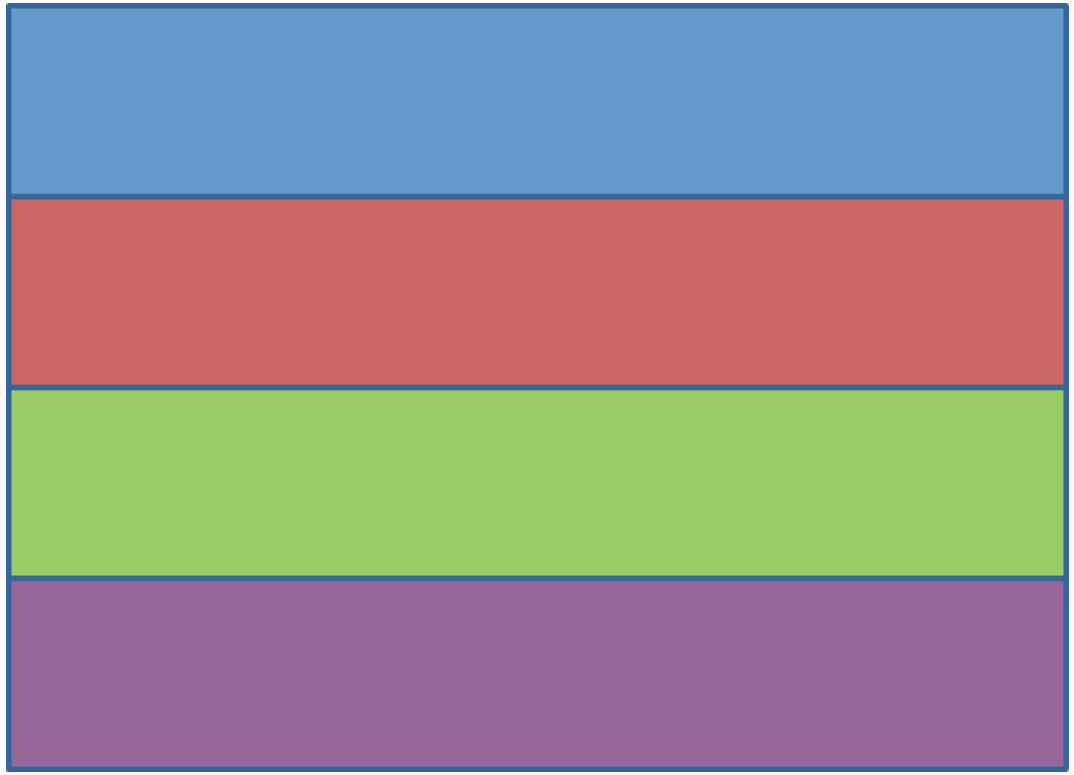
Distributed Matrices

pbdDMAT

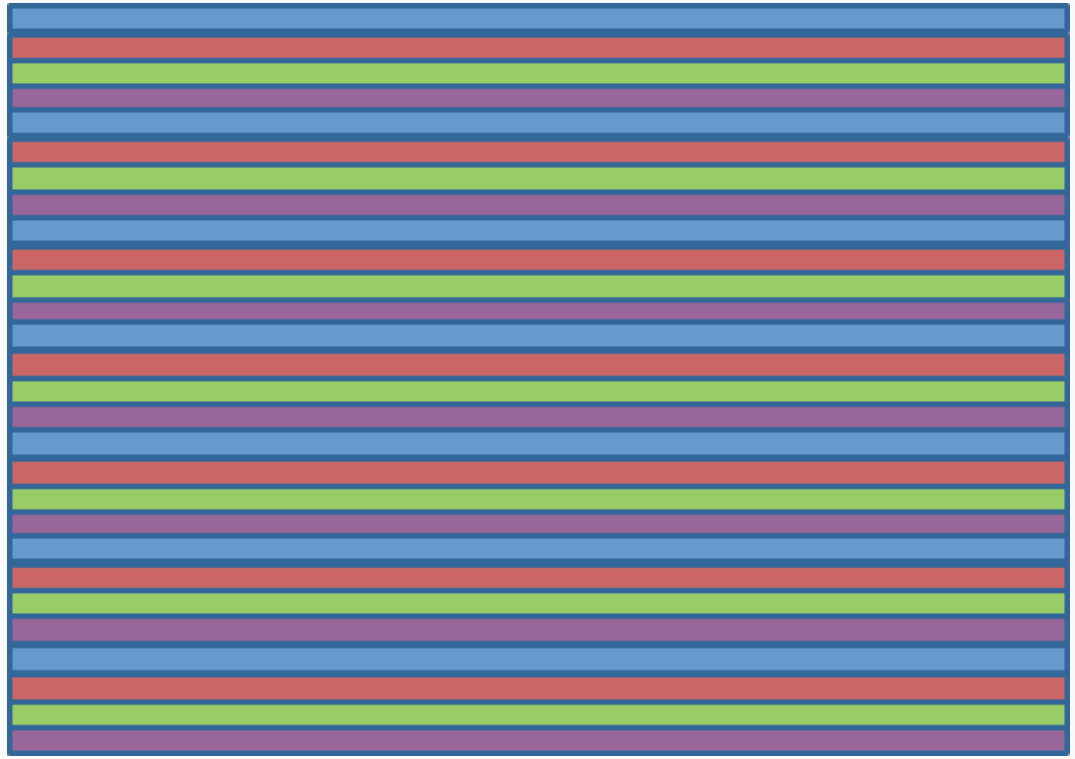
- High-level framework for distributed linear algebra and statistics
- Uses block-cyclic data decomposition (ScaLAPACK)
- Makes computing easy, but reading data still hard
- Syntax often identical to base R
  - Helpers: `[`, `rbind()`, `apply()`, ...
  - Linear algebra: `%*%`, `svd()`, `qr()`, ...
  - Basic statistics: `median()`, `mean()`, `rowSums()`, ...
  - Multivariate statistics: `lm.fit()`, `prcomp()`, `cov()`, ...

Block-cyclic

1-d Block

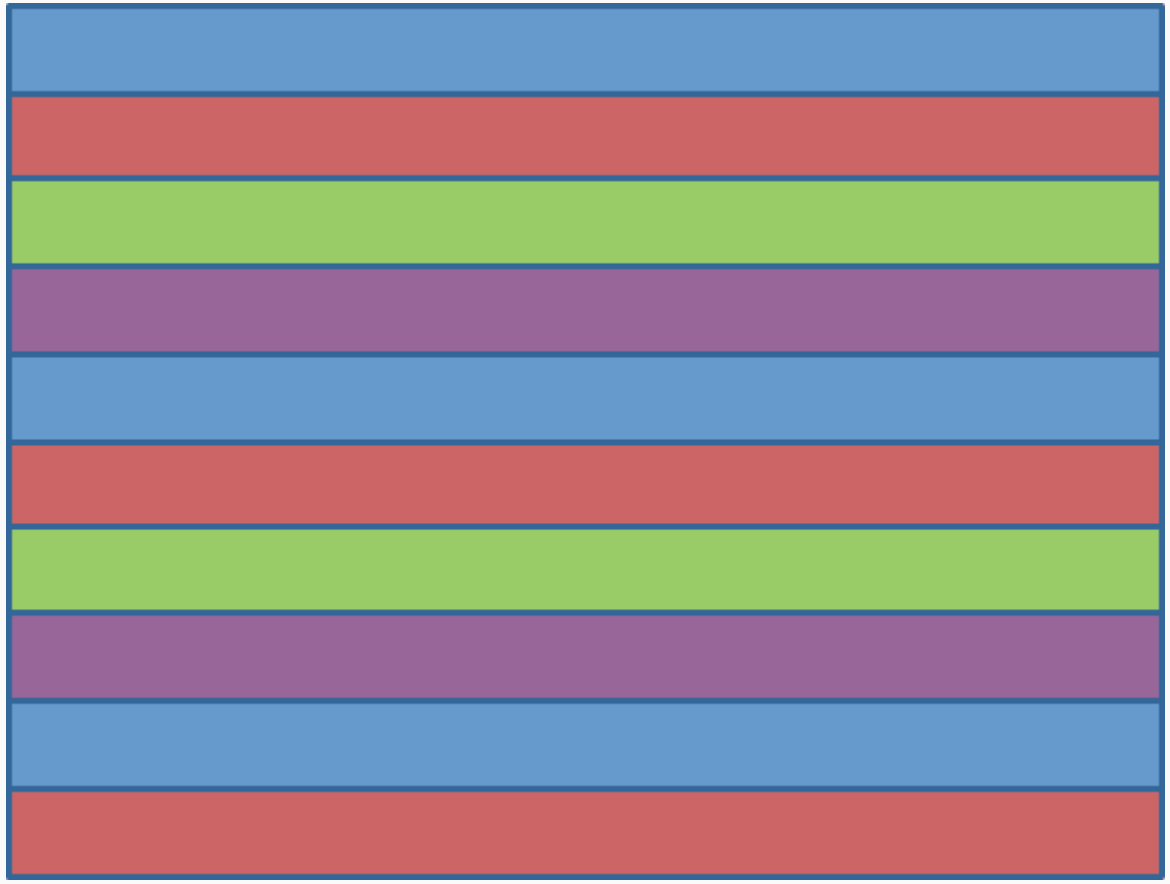


1-d cyclic



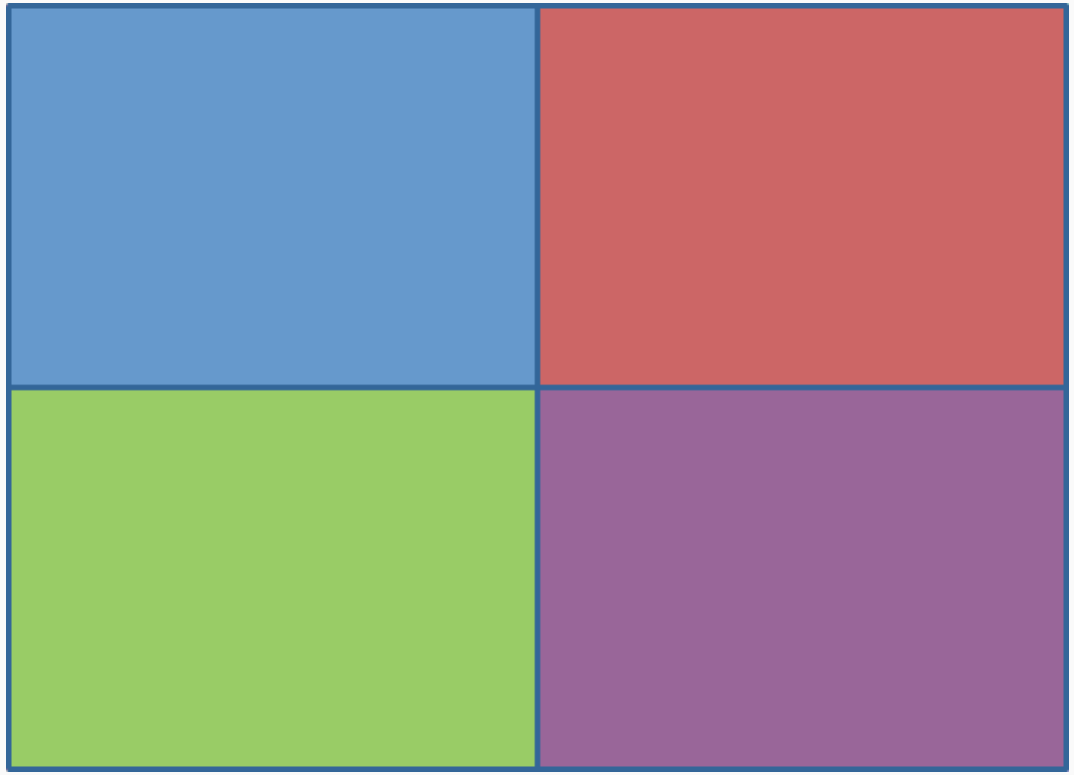
Block-cyclic

1-d block-cyclic

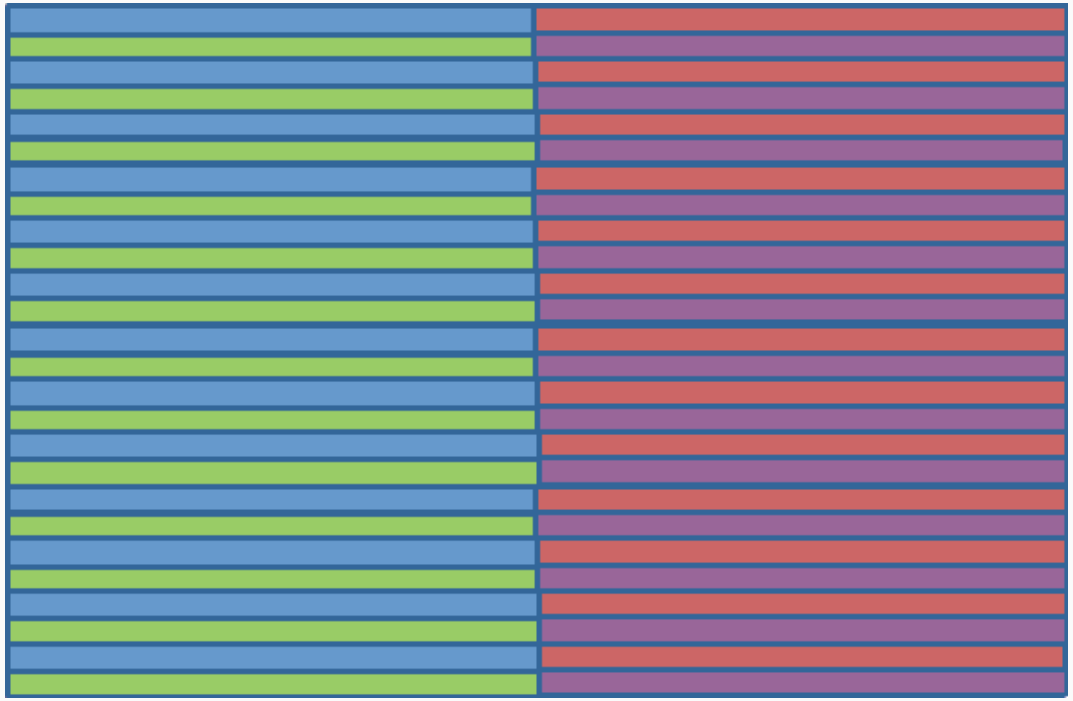


Block-cyclic

2-d Block

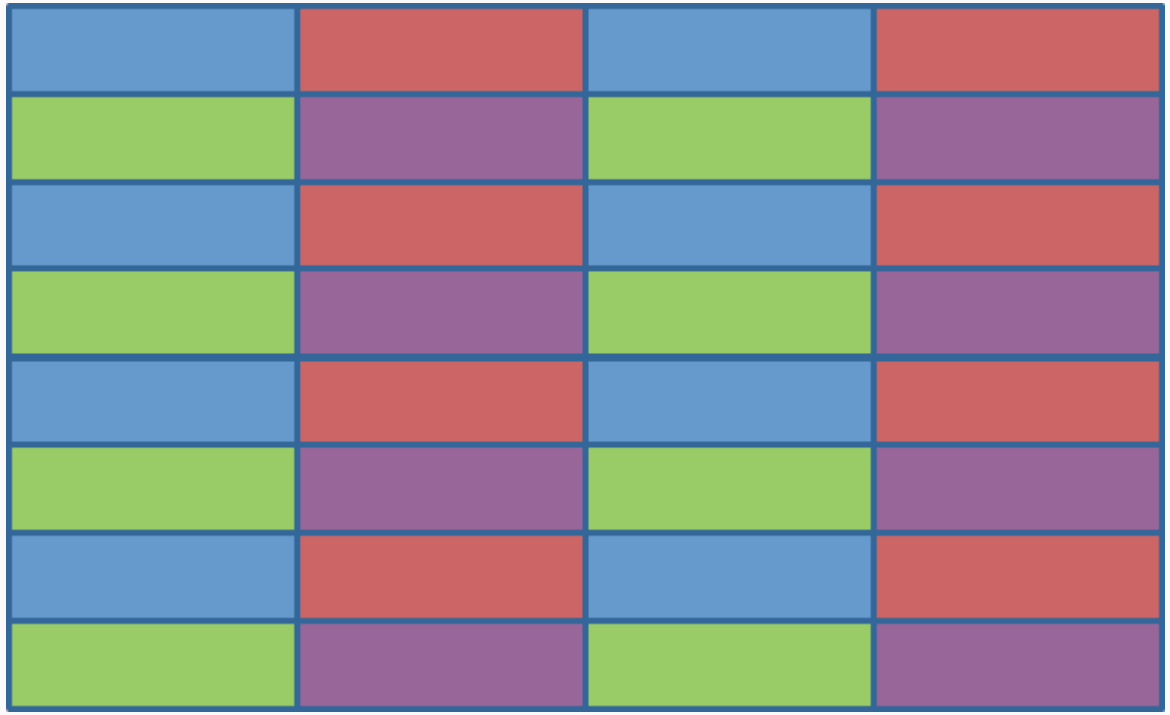


2-d cyclic



Block-cyclic

2-d block-cyclic



kazaam

- High-level framework for distributed linear algebra and statistics
- Optimized for very tall matrices with comparatively few columns ("shaqs")
- Many linear algebra and machine learning methods
- Data distributed by rows (however you want)

pbdDMAT vs kazaam

- pbdDMAT much more thorough (sort of has to be...)
- Both have similar analytics capabilities (clustering, classifiers, dimension reduction, ...)
- kazaam presently works better on GPU's
  - ECP slate may change this
  - both DIY right now
- Can redistribute from one layout to the other fairly easily

Benchmarks

Percival

- Cray XC40
- 168 nodes
- 10,752 cores
- 21 TiB of RAM

pbdDMAT

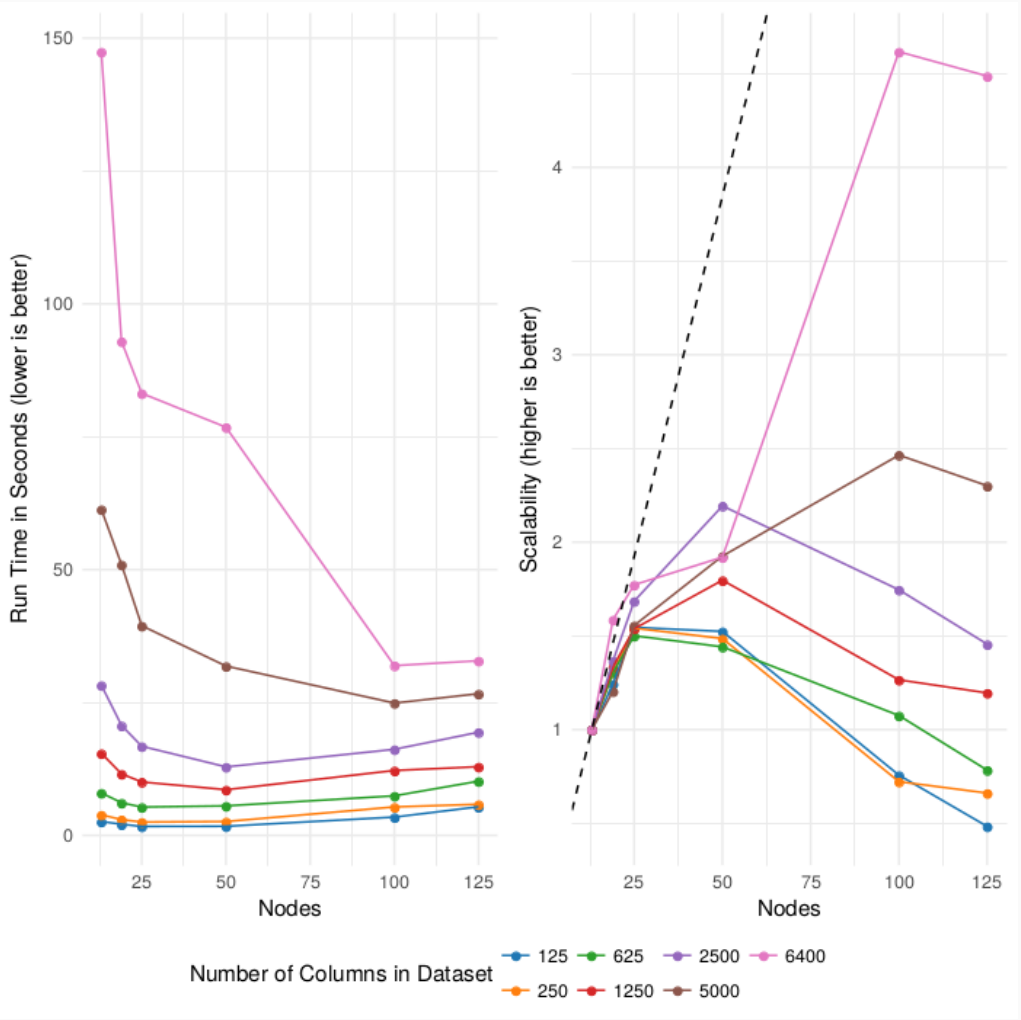
```
x = ddmatrix("rnorm", m, n, ICTXT=2)

time = comm.timer({
  cp = crossprod(x)
  eigen(cp, symmetric=TRUE, only.values=TRUE)
})
```

kazaam

```
x = ranshaq(rnorm, m.local, n, local=TRUE)

time = comm.timer(svd(x, nu=0, nv=0))
```

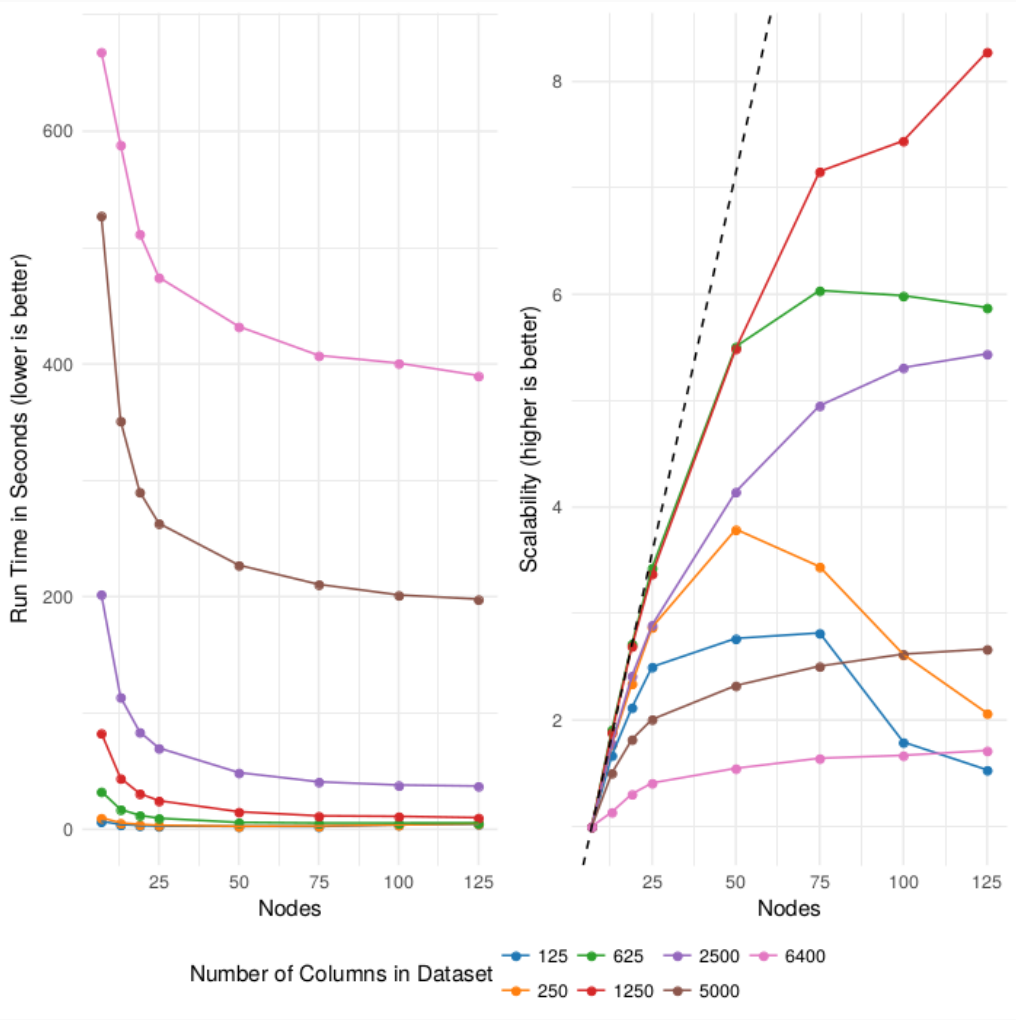
Benchmarks

pbdDMAT



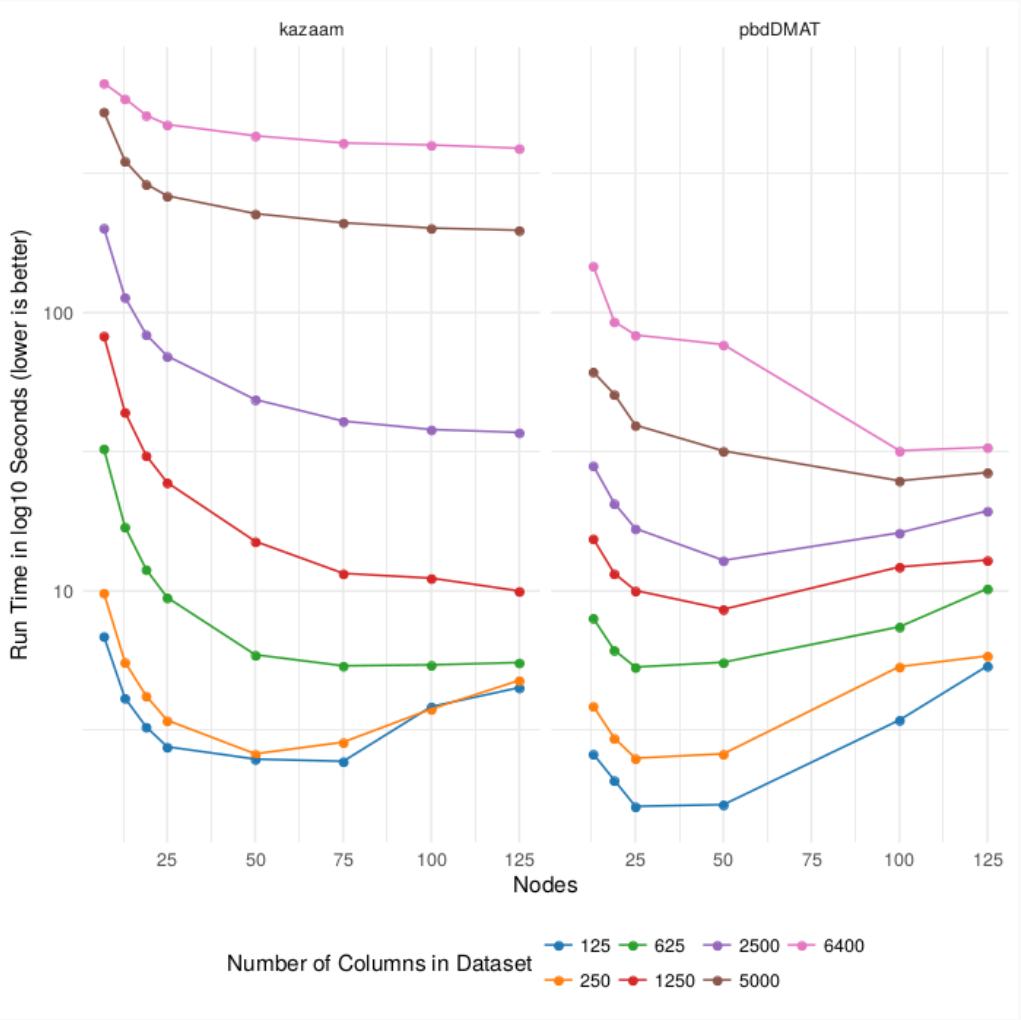
Benchmarks

kazaam



Benchmarks

pbdDMAT vs kazaam



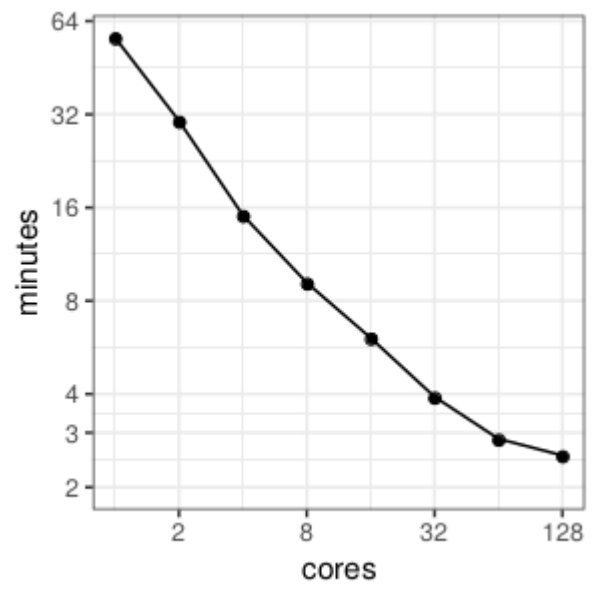
An Application

Outline

- Why you need performance for data analysis
- HOSVD
  - The math
  - The Algorithm
  - Removing covariance structure
- Workflow and scaling results
- Code notes

Why you need performance for data analysis

- Data analysis is a discovery process
- Iterate many times with different parameters, transformations, or models
- Context is lost if an iteration takes more than few minutes to compute
- Recovering context can take hours of researcher time



HOSVD: The Math

Primary source (including figures)

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl*, 21:1253-1278, 2000.

The SVD (2d tensor HOSVD)

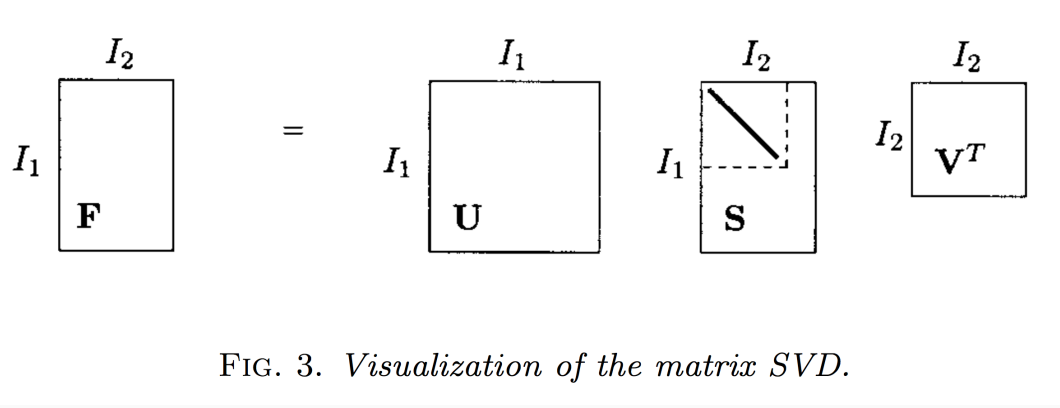


FIG. 3. *Visualization of the matrix SVD.*

- $F = USV^T$
- $F = S \times_1 U \times_2 V$
- U and $$\bV$ are orthogonal ($\bU^T\bU = \bI$, $V^TV = I$)
- S positive, diagonal, ordered
- U and V are unique up to sign
- Consequently $F^TF = US^2U^T$

HOSVD: The Math

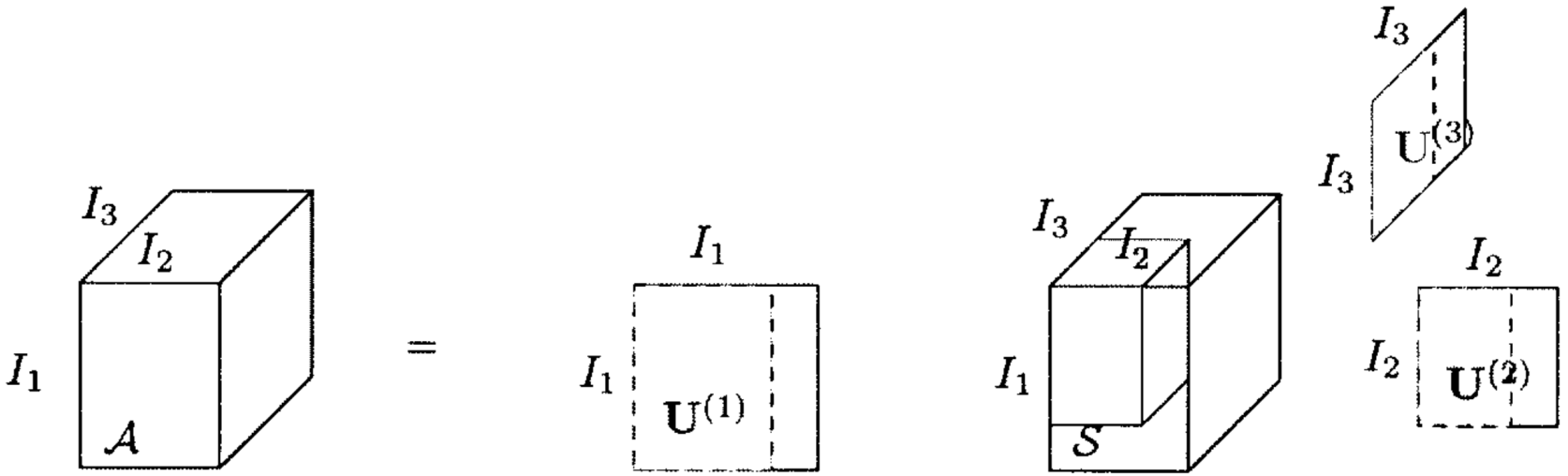


FIG. 4. *Visualization of the HOSVD for a third-order tensor.*

- $\mathcal{A} = \mathcal{S} \times_1 \mathrm{U}^{(1)} \times_2 \mathrm{U}^{(2)} \times_3 \mathrm{U}^{(3)}$
- $\mathrm{U}^{(i)}$ orthogonal (${\bU^{(i)}}^T\bU^{(i)} = I$ for $i = 1, 2, 3$)
- $\mathcal{S}$ tensor is all-orthogonal (all its slice matrices are orthogonal)

HOSVD: The Algorithm

3d HOSVD computes 3 SVDs, one for each unfolding

- $\mathrm{A}_{(1)} = \mathrm{U}^{(1)} \Sigma^{(1)} {\mathrm{V}^{(1)}}^T$
- $\mathrm{A}_{(2)} = \mathrm{U}^{(2)} \Sigma^{(2)} {\mathrm{V}^{(2)}}^T$
- $\mathrm{A}_{(3)} = \mathrm{U}^{(3)} \Sigma^{(3)} {\mathrm{V}^{(3)}}^T$

Keep $\mathrm{U}^{(1)}$, $\mathrm{U}^{(2)}$, $\mathrm{U}^{(3)}$ and compute the $\mathcal{S}$

- $\mathcal{S} = \mathcal{A} \times_1 {\mathrm{U}^{(1)}}^T \times_1 {\mathrm{U}^{(2)}}^T \times_1 {\mathrm{U}^{(3)}}^T$
- Note that $(\mathcal{A} \times_i \mathrm{U}^{(i)})_{(i)} = {\mathrm{U}^{(i)}}^T \mathrm{A}_{(i)}$
- So $\mathrm{S}_{(3)} = {\mathrm{U}^{(3)}}^T ({\mathrm{U}^{(2)}}^T ({\mathrm{U}^{(1)}}^T \mathrm{A}_{(1)})_{(2)})_{(3)}$

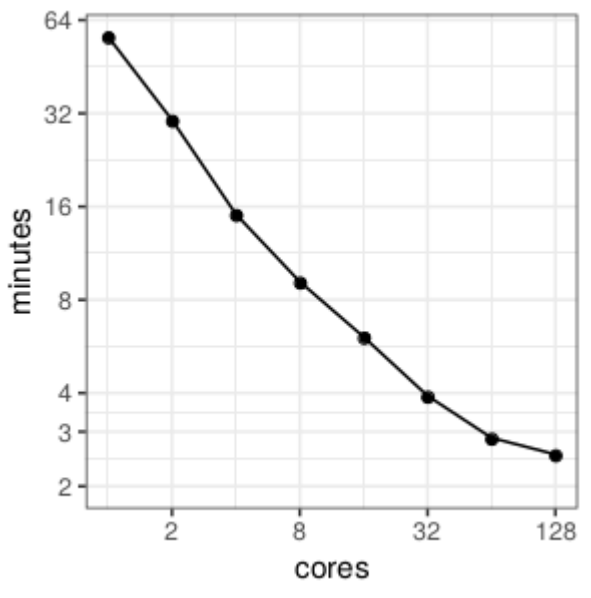
Removing Covariance Structure: Whitening}

Displaying

- $\mathcal{A}$ is toroidal angle * time * mesh
- $\mathrm{A}_{(3)} = \mathrm{U}^{(3)}\Sigma^{(3)}{\mathrm{V}^{(3)}}^{T}$
  - Each row of $\mathrm{V}^{(3)}$ maps to mesh
  - Want to visualize this adjusted for angle and time covariance structure
- $({\mathrm{U}^{(2)}}^{T}({\mathrm{U}^{(1)}}^{T}\mathrm{A}_{(1)})_{(2)})_{(3)} = \tilde{\mathrm{U}}^{(3)}\tilde{\Sigma}^{(3)}{\tilde{\mathrm{V}}^{(3)}}^{T}$

HOSVD Workflow Script

- Tensor dimensions 41 *32* 232,011 (~ 2.3 GB)
- Read 41 HDF5 files (30 MB each)
- 5 SVD computations in series
- 6 unfoldings
- 80 pdf plots (~3 MB each)



Multiple Plots in Parallel

```
library(pbdIO)

#. . .

nplots = min(maxplots, length(d))
myPCs = comm.chunk(nplots, form="vector", type="balance")
my.d = d[myPCs]

#. . .

for (i in seq_along(myPCs)){
    spacePlot(mesh, Vc[, i]*my.d[i], sprintf(ftag, myPCs[i]))
}
```

Creating Directories

```
## Crate output directories
screedir = paste0(ref_dir, "scree/")
ref = sprintf("%0.5d", w_center)
plotdir = paste0(ref_dir, "plots", ref, "/")
if(myrank == 0) { # only one rank should be creating a directory
    dir.create(ref_dir, showWarnings=FALSE)
    dir.create(screedir, showWarnings=FALSE)
    dir.create(plotdir, showWarnings=FALSE)
}

barrier() # must be reachable by all ranks
```

Reading HDF5 in Parallel

```
library(rhdf5)

# . . .

iopair_n = comm.chunk(mesh$n_n, form = "iopair")

## rhdf5 needs to add 1 to C/Python written data!
buffer = h5read(file, "coordinates/values",
                start=c(0 + 1, iopair_n[1] + 1),
                count=c(2, iopair_n[2]))

rz = do.call(c, allgather(unlist(buffer)))
```

HOSVD: The Algorithm

Unfolding a 3d tensor: Data wrangling!! Skinny matrices!!

Unfolding a 3d tensor and shaq/tshaq SVD

```
tens = read_xgc_window(file_var, var, w_center, window)$Data
tdim = dim(tens) # tensor dimensions (1, 2, 3d) = (toro, time, mesh)

## u1 toro: want dimensions (1, 3d*2) - need tshaq
u1tens = as.vector(tens) # (1, 2, 3d)
dim(u1tens) = c(tdim[1], tdim[2]*tdim[3]) # (1, 2*3d)
u1tens.s = tshaq(u1tens) # (1, 2*3d) tshaq
u1svd = svd(u1tens.s)

## u3 mesh: want dimensions (3d, 1*2) - need shaq of transpose
u3tens = as.vector(tens) # (1, 2, 3d)
dim(u3tens) = c(tdim[1]*tdim[2], tdim[3]) # dim (1*2, 3d)
u3tens.s = shaq(t(u3tens)) # transposed so dim (3d, 1*2) shaq
u3svd = svd(u3tens.s)
```

Core tensor data wrangling

```
## Core tensor computation
u1core1 = crossprod(u1svd$u, u1tens) # dim (1, 2*3d), all local op

u3core1 = as.vector(u1core1) # (1, 2, 3d)
dim(u3core1) = c(tdim[1]*tdim[2], tdim[3]) # dim (1*2, 3d)
u2core1 = u3core1[rindex, ] # reordered to (2, 1, 3d) dim (2*1, 3d)
dim(u2core1) = c(tdim[2], tdim[1]*tdim[3]) # dim (2, 1*3d)
u2core21 = crossprod(u2svd$u, u2core1) # dim(2, 1*3d), all local

u3core21 = as.vector(u2core21) # (2, 1, 3d)
dim(u3core21) = c(tdim[2]*tdim[1], tdim[3]) # dim (2*1, 3d)
u3core321 = crossprod(u3svd$u, shaq(t(u3core21))) # dim (3td, 2*1)
## Note that 3td = 2*1 as 3 > 2*1 leads to 3 - 2*1 zero eigenvalues
##      so that leading dimension of u3core321 is 2*1 instead of 3.
##      u3core ends up a local matrix that is replicated. The shaq
##      crossprod collapses the long dimension.
```

Thanks!