pbdR: Harnessing HPC Research for Parallel Computing with R

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

ISM HPCCON and ISM HPC on R Workshop, October 9-12, 2015
Institute of Statistical Mathematics, Tokyo, Japan

The pbdR Core Team

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

[1]FDA
Washington, DC, 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

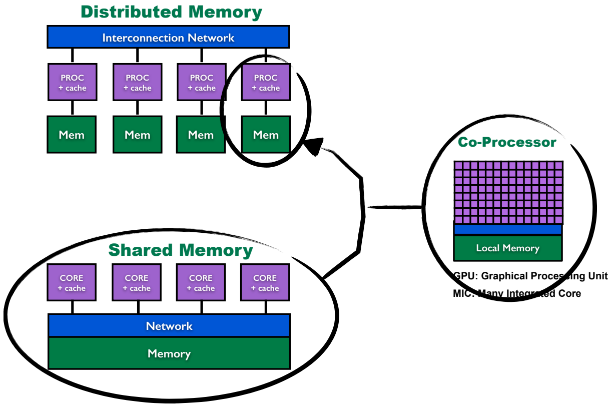
pbdR
Programming with Big Data in R

Support

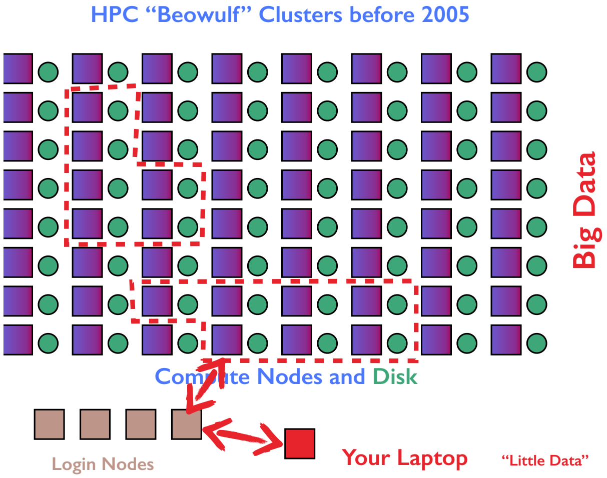
This material is based upon work supported by the National Science Foundation Division of Mathematical Sciences under Grant No. 1418195. This work used resources of the National Institute for Computational Sciences at the University of Tennessee, Knoxville, which is supported by the Office of Cyberinfrastructure of the U.S. National Science Foundation. 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.

Contents

Cores and Co-Processors to Nodes



Parallel Computing before Multicore



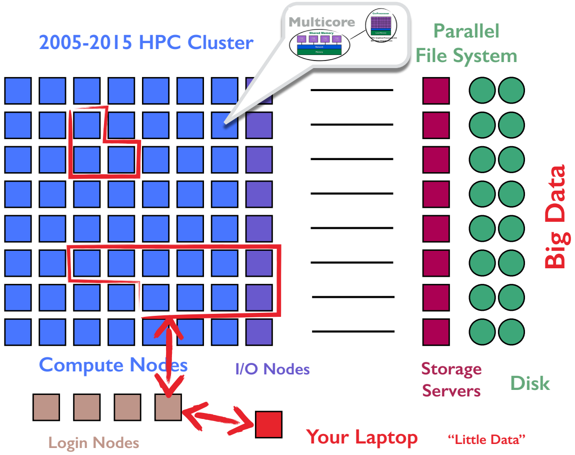
Software Developments:

MPI is mature, MapReduce emerges

Parallel Libraries: PBLAS, ScaLAPACK, PETSc, etc.

Resource Manager: PBS mature, HADOOP emerges

Multicore Emerges and Clusters become Diskless

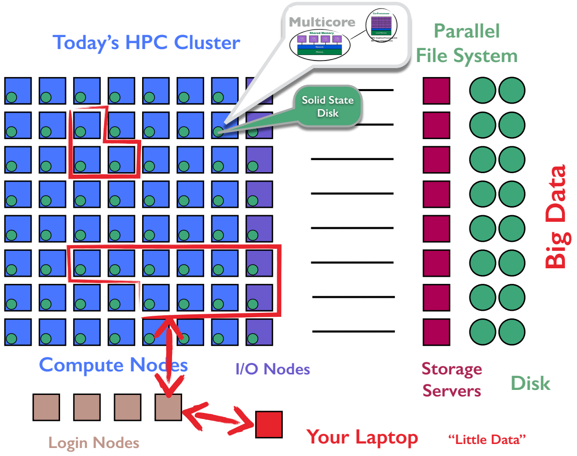


Software Developments

OpenMP, CUDA, OpenCL, OpenACC

Libraries: PLASMA, MAGMA, CuBLAS

Adding NVRAM to New HPC Systems

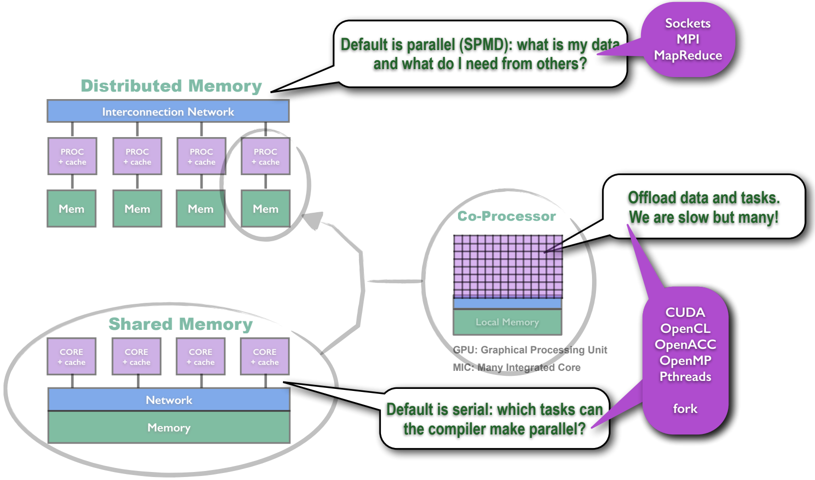


Software Developments

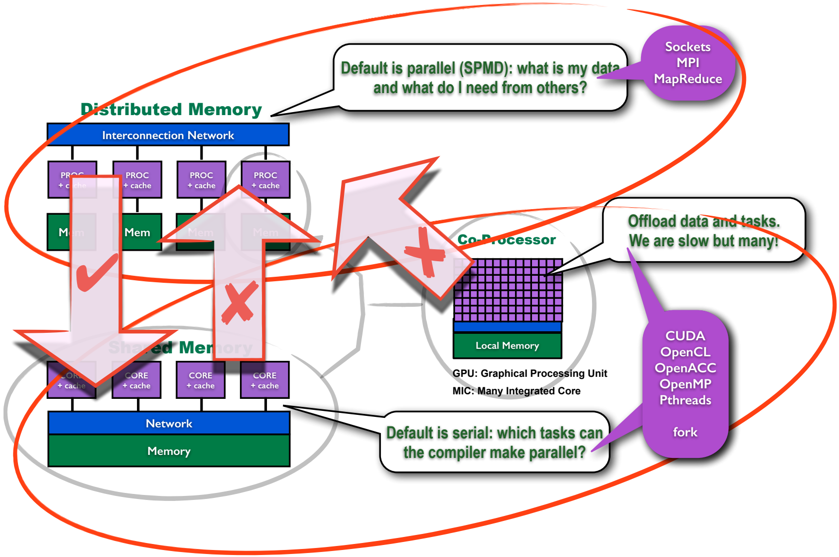
Libraries: DPLASMA, CombBLAS

HADOOP fades, Spark emerges

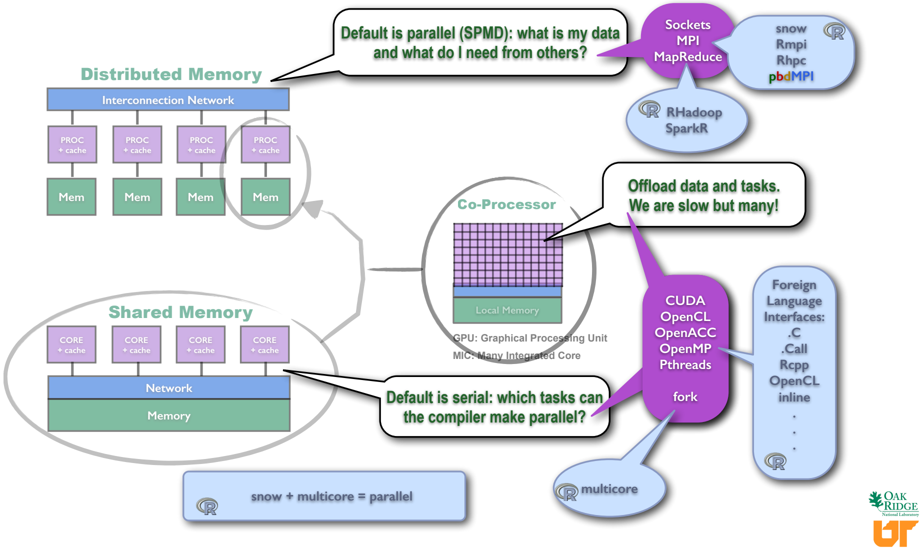
"Native" Programming Models and Tools



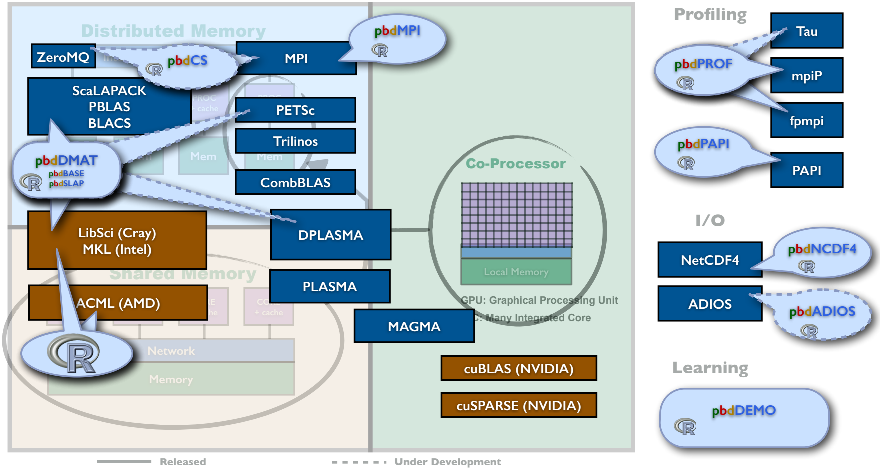
Distributed Programming Works in Shared Memory



R Interfaces to Low-Level Native Tools


Default is parallel (SPMD): what is my data and what do I need from others?
Sockets
MPI
MapReduce
snow
Rmpi
Rhpc
pbdMPI
RHadoop
SparkR
Distributed Memory
Interconnection Network
PROC + cache
Mem
Co-Processor
Local Memory
GPU: Graphical Processing Unit
MIC: Many Integrated Core
Offload data and tasks. We are slow but many!
CUDA
OpenCL
OpenACC
OpenMP
Pthreads
fork
Foreign Language Interfaces:
.C
.Call
Rcpp
OpenCL
inline
Shared Memory
CORE + cache
Network
Memory
Default is serial: which tasks can the compiler make parallel?
snow + multicore = parallel
multicore

R and pbdR Interfaces to HPC Libraries



Data analysis is interactive!

- Data reduction to knowledge
- Iterative process with same data
  - Exploration, model construction
  - Diagnostics of fit and quantification of uncertainty
  - Interpretation
- S (and R) interactive “answer” to batch data analysis
- Efficient use of expensive people

Big platform computing is batch!

- Libraries built for batch computing
- Traditionally data generation by simulation science
- Efficient use of expensive platforms

High-Level Language: Batch and Interactive Distinction Blurred.

- A function is a "batch" script
- R "An interactive environment to use batch scripts"

Ideal solution: Interactive Client with a Batch Server

- Parallel visualization systems (VisIt and ParaView) are client-server (batch on server)
- Current pbdR packages address server side (batch)
- pbdCS 0.1-0 released on GitHub
  - Interactive SPMD
  - Based on ZeroMQ distributed messaging (pbdZMQ 0.1-1 on CRAN)
  - Bridge resource manager (pbdSCHED 0.1-0 on GitHub)
  - Site configuration file
  - Manage relationship of big data (server side) to little data (client side)

Manager-Workers

- A serial program (Manager) divides up work and/or data
- Workers run in parallel without interaction
- Manager collects/combines results from workers
- Divide-Recombine fits this model

MapReduce

- A concept born of a search engine
- Decouples certain coupled problems with an intermediate communication - shuffle
- User writes two serial codes: Map and Reduce

MapReduce: a Parallel Search Engine Concept

Search MANY documents — **Serve MANY users**

$$\begin{array}{ccc} & & \text{Index Words (keys)} \\ \text{Web} & \text{p0} & \left[\begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \\ \hline B_1 & B_2 & B_3 & B_4 \\ \hline C_1 & C_2 & C_3 & C_4 \\ \hline D_1 & D_2 & D_3 & D_4 \end{array}\right] \\ \text{Pages} & \text{p1} & \\ \text{(records)} & \text{p2} & \\ & \text{p3} & \end{array} \quad \xrightarrow[\texttt{MPI\_Alltoallv}]{\text{Shuffle}} \quad \begin{array}{ccc} & & \text{Web Pages (records)} \\ \text{Index} & \text{p0} & \left[\begin{array}{cccc} A_1 & B_1 & C_1 & D_1 \\ \hline A_2 & B_2 & C_2 & D_2 \\ \hline A_3 & B_3 & C_3 & D_3 \\ \hline A_4 & B_4 & C_4 & D_4 \end{array}\right] \\ \text{Words} & \text{p1} & \\ \text{(keys)} & \text{p2} & \\ & \text{p3} & \end{array}$$

Matrix transpose in another language?

Can use different sets of processors

$$\text{Web Pages (records)} \quad \begin{array}{c|cccc|} & \multicolumn{4}{c}{\text{Index Words (keys)}} \\ p0 & & & & \\ \hline p1 & B_1 & B_2 & B_3 & B_4 \\ \hline p2 & & & & \\ \hline p3 & & & & \end{array} \quad \underset{\texttt{MPI\_Scatter}}{\overset{\text{Streaming Shuffle}}{\longrightarrow}} \quad \text{Index Words (keys)} \quad \begin{array}{c|c|} & \text{Web Pages (records)} \\ p4 & B_1 \\ \hline p5 & B_2 \\ \hline p6 & B_3 \\ \hline p7 & B_4 \end{array}$$

MPI and MapReduce

Both Concepts are about Communication

- One makes communication explicit, gives choices
- The other hides communication, gives one choice (shuffle)

SPMD: Single Program Multiple Data

- The prevalent way of distributed programming
- Can handle tightly coupled parallel computations
- It is designed for batch computing
- There is usually no manager - rather, all cooperate
- Prime driver behind MPI specification

Early SPMD Work in Statistics: Crossproduct (Row-Block)

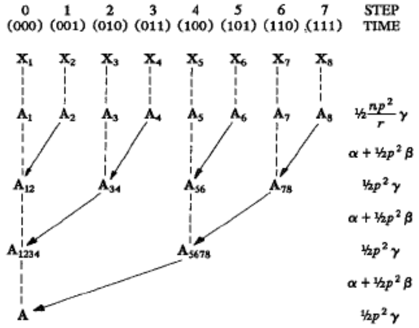

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)
STEP TIME
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
$\frac{1}{2}\frac{np^2}{r}\gamma$
$\alpha + \frac{1}{2}p^2\beta$
A_{12} A_{34} A_{56} A_{78}
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
A_{1234} A_{5678}
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
A
$\frac{1}{2}p^2\gamma$

FIG. 4. *Computation of* $\mathbf{A} = \mathbf{X'X}$ *on an 8-processor hypercube, with final result on processor 0.*

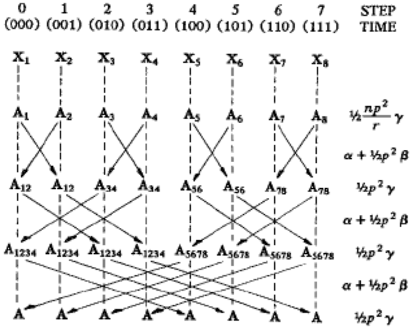

0 (000)
1 (001)
2 (010)
3 (011)
4 (100)
5 (101)
6 (110)
7 (111)
STEP TIME
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
$\frac{1}{2}\frac{np^2}{r}\gamma$
$\alpha + \frac{1}{2}p^2\beta$
A_{12} A_{12} A_{34} A_{34} A_{56} A_{56} A_{78} A_{78}
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
A_{1234} A_{1234} A_{1234} A_{1234} A_{5678} A_{5678} A_{5678} A_{5678}
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
A A A A A A A A
$\frac{1}{2}p^2\gamma$

FIG. 6. *Computation of* $\mathbf{A} = \mathbf{X'X}$ *on an 8-processor hypercube, with final result on all processors.*

Hypercube: Individual send() and recv() over each dimension

Ostrouchov (1987). Parallel Computing on a Hypercube: An overview of the architecture and some applications. *Proceedings of the 19th Symposium on the Interface of Computer Science and Statistics*, p.27-32.

Simplified with MPI (and further with pbdMPI)

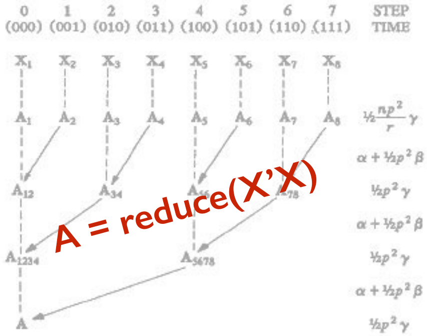

A = reduce(X'X)
0 (000) 1 (001) 2 (010) 3 (011) 4 (100) 5 (101) 6 (110) 7 (111)
STEP TIME
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
A_{12} A_{34} A_{56} A_{78}
A_{1234} A_{5678}
A
$\frac{1}{2}\frac{np^2}{r}\gamma$
$\alpha + \frac{1}{2}p^2\beta$
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
$\frac{1}{2}p^2\gamma$

FIG. 4. *Computation of* A = X'X *on an 8-processor hypercube, with final result on processor 0.*

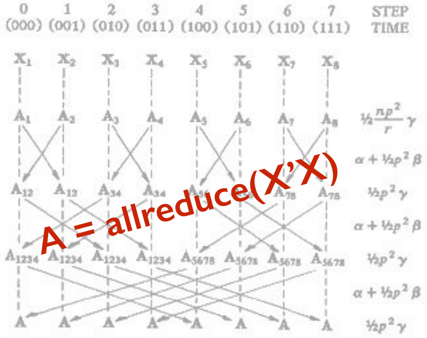

A = allreduce(X'X)
0 (000) 1 (001) 2 (010) 3 (011) 4 (100) 5 (101) 6 (110) 7 (111)
STEP TIME
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
A_{12} A_{12} A_{34} A_{34} A_{56} A_{56} A_{78} A_{78}
A_{1234} A_{1234} A_{1234} A_{1234} A_{5678} A_{5678} A_{5678} A_{5678}
A A A A A A A A
$\frac{1}{2}\frac{np^2}{r}\gamma$
$\alpha + \frac{1}{2}p^2\beta$
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
$\frac{1}{2}p^2\gamma$
$\alpha + \frac{1}{2}p^2\beta$
$\frac{1}{2}p^2\gamma$

FIG. 6. *Computation of* A = X'X *on an 8-processor hypercube, with final result on all processors.*

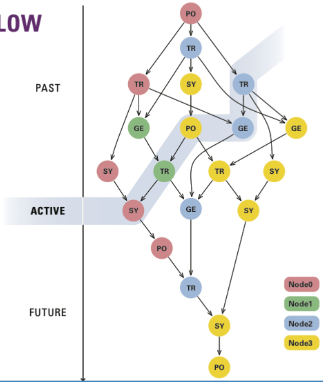
Architecture-specific vendor optimizations

- Cray MPT
- SGI MPT

Data-flow: Parallel Runtime Scheduling and Execution Controller (PaRSEC)

EFFICIENT DATA FLOW REPRESENTATION

FEATURES
Supports Distributed Heterogeneous Platforms
Sustained Performance
NUMA & Cache Aware Scheduling
State-of-the-art Algorithms
Capacity Level Scalability
Performance Portability
Implicit Communication
Communication Overlapping



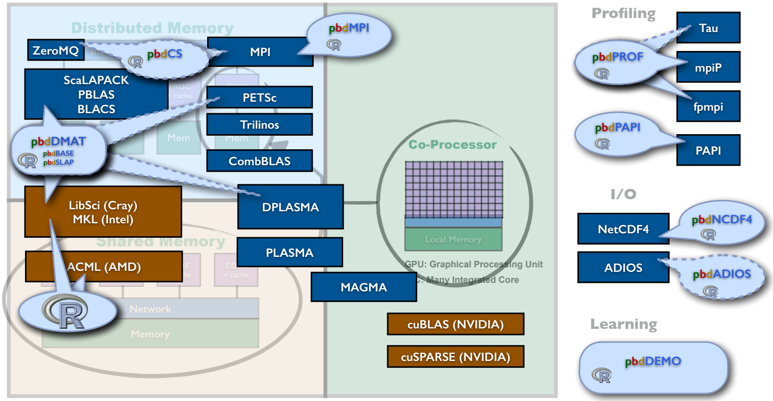
Graphic from icl.cs.utk.edu

Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J. "PaRSEC: Exploiting Heterogeneity to Enhance Scalability," IEEE Computing in Science and Engineering, Vol. 15, No. 6, 36-45, November, 2013.

- Master data-flow controller runs distributed on all cores.
- Dynamic generation of current level in flow graph
- Effectively removes collective synchronizations

Contents

pbdR Interfaces to Libraries: Sustainable Path



Why use HPC libraries?

- The libraries represent 30+ years of research by the HPC community
- *They're tested. They're fast. They're scalable.*
- Many science communities are invested in their API.
- HPC Simulation Science uses much of the same math as data analysis

pbdMPI: Simplified, Extensible, and Fast Communication Operations

- S4 methods for collective communication: extensible to other R objects.
- Default methods (like `Robj` in **Rmpi**) check for data type: safe for general users.
- API is simplified: defaults in control objects.
- Array and matrix methods without serialization: faster than **Rmpi**.

pbdMPI (S4)	**Rmpi**
`allgather`	`mpi.allgather`, `mpi.allgatherv`, `mpi.allgather.Robj`
`allreduce`	`mpi.allreduce`
`bcast`	`mpi.bcast`, `mpi.bcast.Robj`
`gather`	`mpi.gather`, `mpi.gatherv`, `mpi.gather.Robj`
`recv`	`mpi.recv`, `mpi.recv.Robj`
`reduce`	`mpi.reduce`
`scatter`	`mpi.scatter`, `mpi.scatterv`, `mpi.scatter.Robj`
`send`	`mpi.send`, `mpi.send.Robj`

Integer? Not always obvious in R.

```
> is.integer(1)
[1] FALSE
> is.integer(2)
[1] FALSE
> is.integer(1:2)
[1] TRUE
```

pbdMPI lets R figure it out

Rmpi

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

pbdMPI

```
allreduce(x)
```

Single Program (SPMD): Runs Asynchronous Parallel

Rank Query Example

1_rank.r

```r
library(pbdMPI, quiet = TRUE)
init()

my.rank <- comm.rank()
comm.print(my.rank, all.rank=TRUE)

finalize()
```

Execute this batch script via:

```
mpirun -np 2 Rscript 1_rank.r
```

Sample Output:

```
COMM.RANK = 0
[1] 0
COMM.RANK = 1
[1] 1
```

Mapping a Matrix to Processors

Processor Grid Shapes

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

(a) 1×6

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

(b) 2×3

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}$$

(c) 3×2

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

(d) 6×1

Table: Processor Grid Shapes with 6 Processors

2×3 block-cyclic grid on 6 processors: Global view “ddmatrix” class

$$x = \left[\begin{array}{cc|cc|cc|cc|c}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} & x_{27} & x_{28} & x_{29} \\
\hline
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & x_{37} & x_{38} & x_{39} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} & x_{47} & x_{48} & x_{49} \\
\hline
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & x_{57} & x_{58} & x_{59} \\
x_{61} & x_{62} & x_{63} & x_{64} & x_{65} & x_{66} & x_{67} & x_{68} & x_{69} \\
\hline
x_{71} & x_{72} & x_{73} & x_{74} & x_{75} & x_{76} & x_{77} & x_{78} & x_{79} \\
x_{81} & x_{82} & x_{83} & x_{84} & x_{85} & x_{86} & x_{87} & x_{88} & x_{89} \\
\hline
x_{91} & x_{92} & x_{93} & x_{94} & x_{95} & x_{96} & x_{97} & x_{98} & x_{99}
\end{array}\right]_{9\times 9}$$

$$\text{Processor grid} = \begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{vmatrix} = \begin{vmatrix} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \end{vmatrix}$$

2×3 block-cyclic grid on 6 processors: Local view "ddmatrix" class

$$\left[\begin{array}{cc|cc} x_{11} & x_{12} & x_{17} & x_{18} \\ x_{21} & x_{22} & x_{27} & x_{28} \\ \hline x_{51} & x_{52} & x_{57} & x_{58} \\ x_{61} & x_{62} & x_{67} & x_{68} \\ \hline x_{91} & x_{92} & x_{97} & x_{98} \end{array}\right]_{5\times4} \left[\begin{array}{cc|c} x_{13} & x_{14} & x_{19} \\ x_{23} & x_{24} & x_{29} \\ \hline x_{53} & x_{54} & x_{59} \\ x_{63} & x_{64} & x_{69} \\ \hline x_{93} & x_{94} & x_{99} \end{array}\right]_{5\times3} \left[\begin{array}{cc} x_{15} & x_{16} \\ x_{25} & x_{26} \\ \hline x_{55} & x_{56} \\ x_{65} & x_{66} \\ \hline x_{95} & x_{96} \end{array}\right]_{5\times2}$$

$$\left[\begin{array}{cc|cc} x_{31} & x_{32} & x_{37} & x_{38} \\ x_{41} & x_{42} & x_{47} & x_{48} \\ \hline x_{71} & x_{72} & x_{77} & x_{78} \\ x_{81} & x_{82} & x_{87} & x_{88} \end{array}\right]_{4\times4} \left[\begin{array}{cc|c} x_{33} & x_{34} & x_{39} \\ x_{43} & x_{44} & x_{49} \\ \hline x_{73} & x_{74} & x_{79} \\ x_{83} & x_{84} & x_{89} \end{array}\right]_{4\times3} \left[\begin{array}{cc} x_{35} & x_{36} \\ x_{45} & x_{46} \\ \hline x_{75} & x_{76} \\ x_{85} & x_{86} \end{array}\right]_{4\times2}$$

$$\text{Processor grid} = \left|\begin{array}{ccc} 0 & 1 & 2 \\ 3 & 4 & 5 \end{array}\right| = \left|\begin{array}{ccc} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \end{array}\right|$$

pbdR Example Syntax

```
x <- x[-1, 2:5]
x <- log(abs(x) + 1)
x.pca <- prcomp(x)
xtx <- t(x) %*% x
ans <- svd(solve(xtx))
```

The above (and over 100 other functions) runs on 1 core with R or 10,000 cores with pbdR ddmatrix class

```
> showClass("ddmatrix")
Class "ddmatrix" [package "pbdDMAT"]
Slots:
Name:      Data      dim     ldim    bldim    ICTXT
Class:   matrix  numeric  numeric  numeric  numeric
```

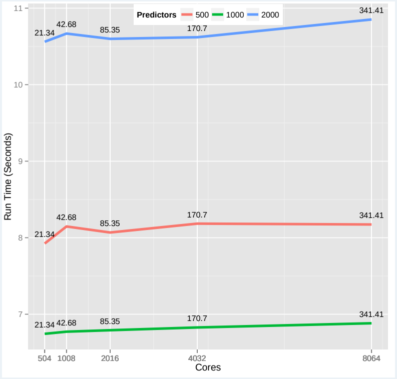
```
> x <- as.rowblock(x)
> x <- as.colblock(x)
> x <- redistribute(x, bldim=c(8, 8), ICTXT = 0)
```

pbdDMAT Scalability Benchmarks

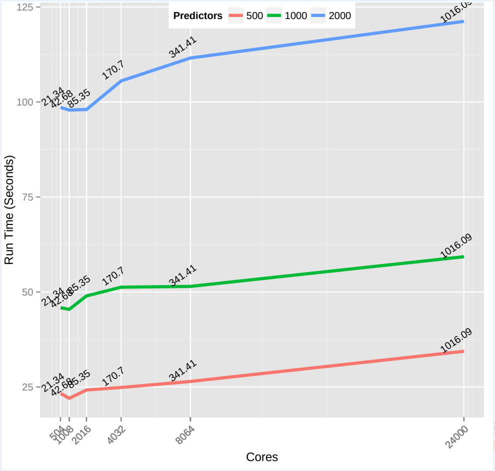
- Default choices throughout (no MKL, ACML, etc.)
- 1 core = 1 MPI process (Kraken: 6-core Opterons)
- Generate random matrix
  - Global Columns: 500, 1000, and 2000
  - Global Rows: fixed per core to make 43.4*MiB*
- Measure wall clock time
- "weak scaling" = global problem grows with core count

pbdDMAT Scalability Benchmarks

```
x <- ddmatrix("rnorm",
    nrow=n, ncol=p)
cov.x <- cov(x)
```

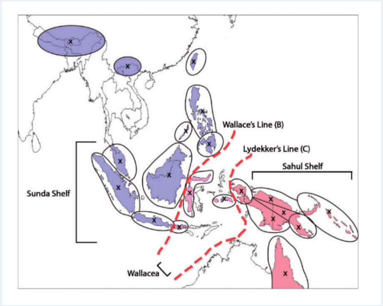


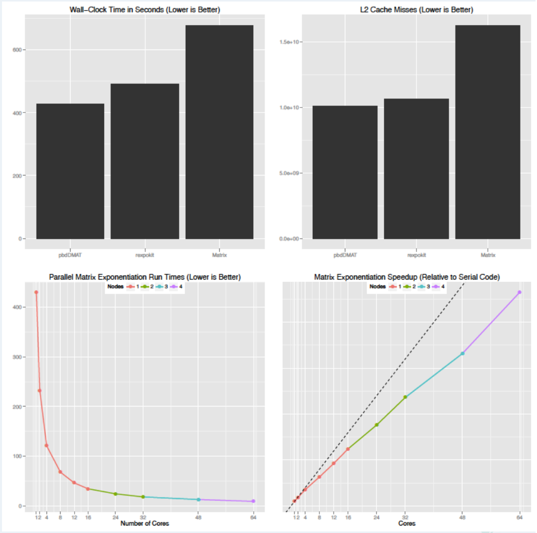
```
b <- ddmatrix("runif", nrow=p,
    ncol=1)
y <- x %*% b
b.hat <- lm.fit(x, y)$coefficients
```



Matrix Exponentiation (pbdDMAT)

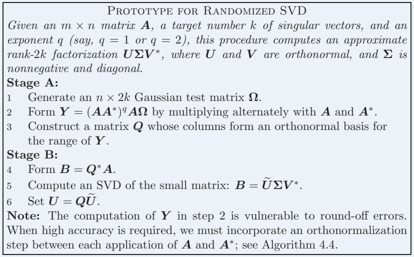
- Fitting biogeography models requires many matrix exponentiations
- Benchmark: Matrix exponential of 5000×5000 matrix.
- R 3.1.0, Matrix 1.1-2, rexpokit 0.25, pbdDMAT 0.3-0
- Libs: Cray LibSci, NETLIB ScaLAPACK, Compilers: gnu 4.8.2
- Configuration: 1 thread == 1 MPI rank == 1 physical core





Schmidt and Matzke (2014) Distributed matrix exponentiation, The R User Conference (UseR! 2014), Los Angeles, CA, August 2014

Randomized truncated SVD[1]

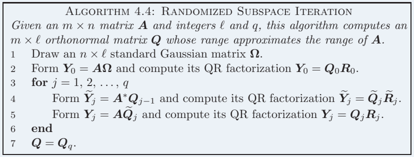


Prototype for Randomized SVD

Given an $m \times n$ matrix $\boldsymbol{A}$, a target number k of singular vectors, and an exponent q (say, $q = 1$ or $q = 2$), this procedure computes an approximate rank-$2k$ factorization $\boldsymbol{U\Sigma V}^*$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal, and $\boldsymbol{\Sigma}$ is nonnegative and diagonal.

```
Stage A:
1   Generate an $n \times 2k$ Gaussian test matrix $\boldsymbol{\Omega}$.
2   Form $\boldsymbol{Y} = (\boldsymbol{AA}^*)^q\boldsymbol{A\Omega}$ by multiplying alternately with $\boldsymbol{A}$ and $\boldsymbol{A}^*$.
3   Construct a matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for
    the range of $\boldsymbol{Y}$.
Stage B:
4   Form $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$.
5   Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}}\boldsymbol{\Sigma V}^*$.
6   Set $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.
```

Note: The computation of $\boldsymbol{Y}$ in step 2 is vulnerable to round-off errors. When high accuracy is required, we must incorporate an orthonormalization step between each application of $\boldsymbol{A}$ and $\boldsymbol{A}^*$; see Algorithm 4.4.



Algorithm 4.4: Randomized Subspace Iteration

Given an $m \times n$ matrix $\boldsymbol{A}$ and integers ℓ and q, this algorithm computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

```
1   Draw an $n \times \ell$ standard Gaussian matrix $\boldsymbol{\Omega}$.
2   Form $\boldsymbol{Y}_0 = \boldsymbol{A\Omega}$ and compute its QR factorization $\boldsymbol{Y}_0 = \boldsymbol{Q}_0\boldsymbol{R}_0$.
3   for $j = 1, 2, \dots, q$
4       Form $\widetilde{\boldsymbol{Y}}_j = \boldsymbol{A}^*\boldsymbol{Q}_{j-1}$ and compute its QR factorization $\widetilde{\boldsymbol{Y}}_j = \widetilde{\boldsymbol{Q}}_j\widetilde{\boldsymbol{R}}_j$.
5       Form $\boldsymbol{Y}_j = \boldsymbol{A}\widetilde{\boldsymbol{Q}}_j$ and compute its QR factorization $\boldsymbol{Y}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$.
6   end
7   $\boldsymbol{Q} = \boldsymbol{Q}_q$.
```

Serial R

```r
randSVD <- function(A, k, q=3)
  {
    ## Stage A
    Omega <- matrix(rnorm(n*2*k),
      nrow=n, ncol=2*k)
    Y <- A %*% Omega
    Q <- qr.Q(qr(Y))
    At <- t(A)
    for(i in 1:q)
      {
        Y <- At %*% Q
        Q <- qr.Q(qr(Y))
        Y <- A %*% Q
        Q <- qr.Q(qr(Y))
      }

    ## Stage B
    B <- t(Q) %*% A
    U <- La.svd(B)$u
    U <- Q %*% U
    U[, 1:k]
  }
```

[1]Halko, Martinsson, and Tropp. 2011. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions *SIAM Review* **53** 217–288

Randomized truncated SVD

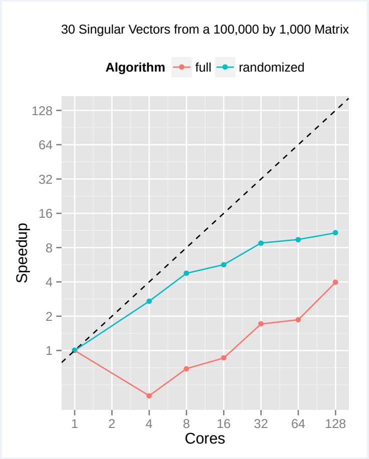
Serial R

```
randSVD <- function(A, k, q=3)
  {
    ## Stage A
    Omega <- matrix(rnorm(n*2*k),
           nrow=n, ncol=2*k)
    Y <- A %*% Omega
    Q <- qr.Q(qr(Y))
    At <- t(A)
    for(i in 1:q)
      {
        Y <- At %*% Q
        Q <- qr.Q(qr(Y))
        Y <- A %*% Q
        Q <- qr.Q(qr(Y))
      }

    ## Stage B
    B <- t(Q) %*% A
    U <- La.svd(B)$u
    U <- Q %*% U
    U[, 1:k]
  }
```

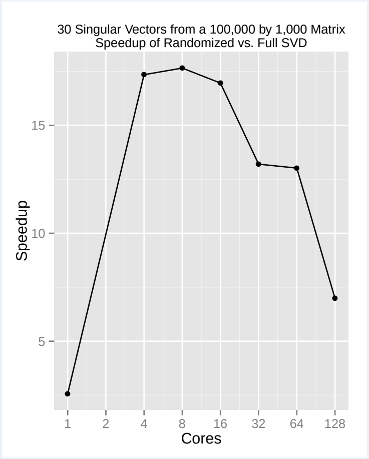
Parallel pbdR

```
randSVD <- function(A, k, q=3)
  {
    ## Stage A
    Omega <- ddmatrix("rnorm",
      nrow=n, ncol=2*k)
    Y <- A %*% Omega
    Q <- qr.Q(qr(Y))
    At <- t(A)
    for(i in 1:q)
      {
        Y <- At %*% Q
        Q <- qr.Q(qr(Y))
        Y <- A %*% Q
        Q <- qr.Q(qr(Y))
      }

    ## Stage B
    B <- t(Q) %*% A
    U <- La.svd(B)$u
    U <- Q %*% U
    U[, 1:k]
  }
```

From journal to scalable code and scaling data in one day.



Speedup relative to 1 core



RandSVD speedup relative to full SVD

Letter Recognition Data

Example 1: Letter Recognition data from package **mlbench** (20,000 $\times$ 17)

```
[,1] lettr capital letter
[,2] x.box horizontal position of box
[,3] y.box vertical position of box
[,4] width width of box
[,5] high height of box
[,6] onpix total number of on pixels
[,7] x.bar mean x of on pixels in box
[,8] y.bar mean y of on pixels in box
[,9] x2bar mean x variance
[,10] y2bar mean y variance
[,11] xybar mean x y correlation
[,12] x2ybr mean of x^2 y
[,13] xy2br mean of x y^2
[,14] x.ege mean edge count left to right
[,15] xegvy correlation of x.ege with y
[,16] y.ege mean edge count bottom to top
[,17] yegvx correlation of y.ege with x
```

P. W. Frey and D. J. Slate (Machine Learning Vol 6/2 March 91): "Letter Recognition Using Holland-style Adaptive Classifiers".

Example 1: Random Forest Algorithm

1. Build simple regression trees from random subsets of columns
2. Use model averaging for prediction
3. Package **randomForest** has a `combine()` function that enables the following parallel approach:
    1. Everyone gets the same training data
    2. Split regression tree building among processors (**randomForest**)
    3. Use `allgather` to bring built predictors to all
    4. Everyone `combine` predictors
    5. Split prediction work by blocks of rows
    6. Use `allreduce` to assess prediction
4. Steps (3) and (4) can be improved with a custom reduce/combine to take advantage of MPI vendor optimizations

Example 1: Random Forest Code (Split learning by blocks of trees. Split prediction by blocks of rows.)

Serial Code 4_rf_s.r

```r
library(randomForest)
library(mlbench)
data(LetterRecognition) # 26 Capital Letters Data 20,000 x 17
set.seed(seed=123)
n <- nrow(LetterRecognition)
n_test <- floor(0.2*n)
i_test <- sample.int(n, n_test) # Use 1/5 of the data to test
train <- LetterRecognition[-i_test, ]
test <- LetterRecognition[i_test, ]

## train random forest
rf.all <- randomForest(lettr ~ ., train, ntree=500,
    norm.votes=FALSE)

## predict test data
pred <- predict(rf.all, test)
correct <- sum(pred == test$lettr)
cat("Proportion Correct:", correct/(n_test), "\n")
```

Example 1: Random Forest Code (Split learning by blocks of trees. Split prediction by blocks of rows.)

Parallel Code 4_rf_p.r

```r
library(randomForest)
library(mlbench)
data(LetterRecognition)
comm.set.seed(seed=123, diff=FALSE) # same training data
n <- nrow(LetterRecognition)
n_test <- floor(0.2*n)
i_test <- sample.int(n, n_test) # Use 1/5 of the data to test
train <- LetterRecognition[-i_test, ]
test <- LetterRecognition[i_test, ][get.jid(n_test), ]

comm.set.seed(seed=1e6*runif(1), diff=TRUE)
my.rf <- randomForest(lettr ~ ., train, ntree=500%/%comm.size(),
     norm.votes=FALSE)
rf.all <- do.call(combine, allgather(my.rf))

pred <- predict(rf.all, test)
correct <- allreduce(sum(pred == test$lettr))
comm.cat("Proportion Correct:", correct/(n_test), "\n")
```

Runs serial or on any number of cores

```
[beacon-login2 stats]$ time Rscript 4_rf_s.r
Proportion Correct: 0.96725
real 0m49.028s   user 0m48.626s    sys 0m0.335s
[beacon-login2 stats]$ time Rscript 4_rf_p.r
Proportion Correct: 0.96425
real 0m52.634s   user 0m51.914s    sys 0m0.598s
[beacon-login2 stats]$ time mpirun -np 2 Rscript 4_rf_p.r
Proportion Correct: 0.96425
real 0m28.349s   user 0m54.570s    sys 0m1.070s
[beacon-login2 stats]$ time mpirun -np 4 Rscript 4_rf_p.r
Proportion Correct: 0.963
real 0m16.380s   user 1m1.559s     sys 0m1.664s
[beacon-login2 stats]$ time mpirun -np 8 Rscript 4_rf_p.r
Proportion Correct: 0.963
real 0m11.010s   user 1m19.301s    sys 0m3.421s
[beacon-login2 stats]$ time mpirun -np 16 Rscript 4_rf_p.r
Proportion Correct: 0.9635
real 0m10.655s   user 2m32.508s    sys 0m6.624s
[beacon-login2 stats]$ time mpirun -np 32 Rscript 4_rf_p.r
Proportion Correct: 0.96325
real 0m21.692s   user 4m44.114s    sys 0m20.179s
```

fda.usc Package

Profiling min.basis()

```
> summaryRprof()
$by.total
                      total.time  total.pct  self.time  self.pct
"min.basis"                12.32     100.00       0.00      0.00
"type.CV"                   6.54      53.08       0.02      0.16
"S.basis"                   5.76      46.75       0.00      0.00
"drop"                      4.20      34.09       0.00      0.00
"norm.fdata"                4.20      34.09       0.00      0.00
"metric"                    4.18      33.93       1.04      8.44
"%*%"                       3.98      32.31       3.98     32.31
"getbasispenalty"           2.72      22.08       0.02      0.16
"bsplinepen"                2.68      21.75       0.36      2.92
"int.simpson2"              2.54      20.62       1.96     15.91
"t"                         2.10      17.05       0.10      0.81
"ppBspline"                 1.60      12.99       0.82      6.66
. . .
```

Example: min.basis() 110 lines SPMD: Add 5, change 3

```r
min.basis <- function(fdataobj, type.CV = GCV.S, . . ., ...)
{
    . . . 13 lines
    library(pbdMPI)
    init()
    my.k <- get.jid(lenlambda)
    my.gcv <- array(Inf, dim = c(lenbasis, length(my.k)))
    . . . 36 lines
    for (i in 1:lenbasis) {
        . . . 3 lines
        for (k in my.k) {
            S2 <- S.basis(tt, base, lambda[k])
            my.gcv[i, k - my.k[1] + 1] <-
                type.CV(fdataobj, S = S2, W = W, trim =
                    par.CV$trim, draw = par.CV$draw, ...)
        }
    }
    gcv <- do.call(cbind, allgather(my.gcv))
    finalize()
    . . . 48 lines
```

Contents

3 pbdCS
- Client-Server Demo

0MQ

Some explanation goes here The demo goes here

4 Future Work

Future Work

- Second year of a 3 year NSF grant to
  - Bring back interactivity via client/server (pbdCS 0.1-0)
  - Simplify parallel data input
  - Begin DPLASMA integration
  - Outreach to the statistics community
- DOE funding: In-situ or staging use with simulations
  - Machine learning from fusion simulation data
- Collaboration wishlist
  - RDD, HDFS, etc., file readers
  - Communicator integration with SparkR or Spark
  - Communicator integration with VisIt and ParaView
  - pbdCS integration with RStudio IDE
  - Instrumentation of various R packages with pbdR

Where to learn more?

- http://r-pbd.org/
- **pbdDEMO** vignette
- Googlegroup:RBigDataProgramming