pR: Introduction to Parallel R for Statistical Computing for people with little knowledge but infinite intelligence

CScADS Scientific Data and Analytics for Petascale Computing Workshop

Nagiza F. Samatova

samatovan@ornl.gov
North Carolina State University
Oak Ridge National Laboratory

8/03/2009
Why R?

- Technical computing
- Matrix and vector formulations

Statistical computing and graphics
http://www.r-project.org
  - Developed by R. Gentleman & R. Ihaka
  - Expanded by community as open source
  - Statistically rich

- Data Visualization and analysis platform
- Image processing, vector computing
Statistical Computing with R – http://www.r-project.org

Open source, most widely used for statistical analysis and graphics
Extensible via dynamically loadable add-on packages
>1,800 packages on CRAN

> library (stats)
> pca = prcomp (data)
> summary (pca)
> ...
> dyn.load( “foo.so”)
> .C( “foobar”)
> dyn.unload( “foo.so”)

Getting Started:
- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To download R, please choose your preferred CRAN mirror.
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News:
- R version 2.9.1 has been released on 2009-06-26. The source code will first become available in the directory, and eventually via all of CRAN. Binaries will arrive in due course (see download instructions above).
- The first issue of The R Journal is now available.
- The R Foundation has awarded four slots for R projects in the Google Summer of Code 2009.
- DISC 2009, the 6th workshop on Directions in Statistical Computing, has been held at the Center for Health and Society, University of Copenhagen, Denmark, July 13-14, 2009.
- useR! 2009, the R user conference, has been held at Agrocampus Rennes, France, July 8-10, 2009.
- useR! 2010, the R user conference, will be held at the NIST, Gaithersburg, Maryland, USA, July 20-23, 2010.
How to Get Started with R?

• Step 1: **Download R** (under Linux):
  – mkdir for RHOME; cd $RHOME
  – wget http://cran.cnr.berkeley.edu/src/base/R-2/R-2.9.1.tar.gz

• Step 2: **Install R** (under Linux):
  – tar –zxvf R-2.9.1.tar.gz
  – ./configure --prefix=<RHOME> --enable-R-shlib
  – make
  – make install

• Step 3: **Run R** (under Linux):
  – Update env. variables in $HOME/.bash_profile:
    • export PATH=<RHOME>/bin:$PATH
    • export R_HOME=<RHOME>
  – R
Session 1: Vectors and vector operations

To create a vector:

# c() command to create vector x
x=c(12,32,54,33,21,65)
# c() to add elements to vector x
x=c(x,55,32)

# seq() command to create sequence of number
years=seq(1990,2003)
# to contain in steps of .5
a=seq(3,5,.5)

# rep() command to create data that follow a regular pattern
b=rep(1,5)
c=rep(1:2,4)

To access vector elements:

# 2nd element of x
x[2]
# first five elements of x
x[1:5]
# all but the 3rd element of x
x[-3]
# values of x that are < 40
x[x<40]
# values of y such that x is < 40
y[x<40]

To perform operations:

# mathematical operations on vectors
y=c(3,2,4,3,7,6,1,1)
x+y; 2*y; x*y; x/y; y^2
Session 2: Matrices & matrix operations

To create a matrix:

```r
# matrix() command to create matrix A with rows and cols
A = matrix(c(54,49,49,41,26,43,49,50,58,71), nrow=5, ncol=2)
```

To access matrix element:

```r
# matrix_name[row_no, col_no]
A[2,1]  # 2nd row, 1st column element
A[3,]   # 3rd row
A[,2]   # 2nd column of the matrix
A["KC",] # access row by name, "KC"
```

To perform element by element ops:

```r
2*A+3; A+B; A*B; A/B;
```

Statistical operations:

```r
rowSums(A)
colSums(A)
rowMeans(A)
colMeans(A)
apply(A,2,max)
apply(A,1,min)
```
Help in R

• **At the package level:**
  – help (package=pkg_name) (e.g. help (package=stats))
  – pkg_name::<TAB><TAB> (stats::)
  – CRAN Search:
  – CRAN Task Views: [http://cran.cnr.berkeley.edu/web/views/](http://cran.cnr.berkeley.edu/web/views/)
  – From R GUI: Help → Search help...

• **At the function level:**
  – help (function_name) (e.g. help(prcomp))
  – ?function_name (e.g.: ?prcomp)
  – ??function_name
How to Add a Package?

• From R prompt:
  – install.packages("pkg_name") (install.packages("ctv"))

• From R GUI:
  – Packages \(\rightarrow\) Install package(s)...

• From Command Line:
  – R CMD INSTALL pkg_name.version.tar.gz
Useful R Links

• R Home: http://www.r-project.org/
• R’s CRAN package distribution: http://cran.cnr.berkeley.edu/
• Writing R extensions: http://cran.cnr.berkeley.edu/doc/manuals/R-exts.pdf
• Other R documentation: http://cran.cnr.berkeley.edu/manuals.html
The Programmer’s Dilemma

- **Assembly**
- **Functional languages** (C, Fortran)
- **Object Oriented** (C++, Java)
- **Scripting** (R, MATLAB, IDL)

The diagram illustrates the trade-off between productivity and performance across different programming language levels.
Towards High-Performance High-Level Languages

How do we get there? — Parallelization
One Hat Does NOT Fit All
Parallel R for Data Intensive Statistical Computing

- Technical computing
- Matrix and vector formulations
- Data Visualization and analysis platform
- Image processing, vector computing

Data Intensive Statistical Computing

- Developed by R. Gentleman & R. Ihaka
- Expanded by community as open source
- Extensible via dynamically loadable libs

Statistical computing and graphics

http://www.r-project.org
Towards Enabling Parallel Computing in R
http://cran.cnr.berkeley.edu/web/views/HighPerformanceComputing.html

• **snow** (Luke Tierney): general API on top of message passing routines to provide high-level (*parallel apply*) commands; mostly demonstrated for *embarrassingly parallel* applications.

<table>
<thead>
<tr>
<th>snow API</th>
<th>High Level Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>parLapply</td>
<td>parallel lapply</td>
</tr>
<tr>
<td>parSapply</td>
<td>parallel sapply</td>
</tr>
<tr>
<td>parApply</td>
<td>parallel apply</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Routines</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>clusterExport</td>
<td>export variables to nodes</td>
</tr>
<tr>
<td>clusterCall</td>
<td>call function on each node</td>
</tr>
<tr>
<td>clusterApply</td>
<td>apply function to arguments on nodes</td>
</tr>
<tr>
<td>clusterApplyLB</td>
<td>load balanced clusterApply</td>
</tr>
<tr>
<td>clusterEvalQ</td>
<td>evaluate explicit expression on nodes</td>
</tr>
<tr>
<td>clusterSplit</td>
<td>split vector into pieces for nodes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Administrative Routines</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>makeCluster</td>
<td>create a new cluster of nodes</td>
</tr>
</tbody>
</table>

• **Rmpi** (Hao Yu): *R* interface to *MPI*.

• **rpvm** (Na Li and Tony Rossini): *R* interface to *PVM*; requires knowledge of parallel programming.

> library (*rpvm*)
> .PVM.start.pvmd ()
> .PVM.addhosts (...) 
> .PVM.config ()
Lessons Learned from R/Matlab Parallelization
Interactivity and High-Level: Curse & Blessing

- **Automatic parallelization**
  - task parallelism
  - task-pR (Samatova et al, 2004)

- **Embarrassing parallelism**
  - data parallelism
  - snow (Tierney, Rossini, Li, Sevcikova, 2006)

- **Manual parallelization**
  - message passing
  - Rmpi (Hao Yu, 2006)
  - rpvm (Na Li & Tony Rossini, 2006)

- **Compiled approach**
  - Matlab → C → automatic parallelization

Packages: http://cran.r-project.org/
Types of Users

• **R end-user:**
  – Use R scripting language for statistical analysis tasks

• **R contributor:**
  – Contribute R packages
  – Use R (sometimes serial C/C++, Fortran)

• **HPC developer:**
  – MPI C/C++/Fortran codes
  – Linear algebra routines that underlie data analysis
  – Parallel machine learning and data mining algorithms:
    • Unsupervised learning: clustering, association rule mining,...
    • Supervised learning: SVM, NN, Decision Trees, Bayesian networks
Our Philosophy

• From R end user’s perspective:
  – Require \textbf{NO} (very trivial) changes to serial R code
  – Yet deliver HPC performance

• From HPC developer’s perspective:
  – Provide \textit{native} to HPC developer interface to R internals
  – With NO (constantly small) overhead
Task and Data Parallelism in pR

**Goal:** Parallel R (pR) aims:
1. to automatically detect and execute *task-parallel* analyses;
2. to easily plug-in *data-parallel* MPI-based C/C++/Fortran codes;
3. to retain high-level of *interactivity, productivity and abstraction*.

**Embarrassingly-parallel:**
- Likelihood Maximization
- Sampling: Bootstrap, Jackknife
- Markov Chain Monte Carlo
- Animations

**Data-parallel:**
- k-means clustering
- Principal Component Analysis
- Hierarchical clustering
- Distance matrix, histogram
Interactive R Client

Loosely Coupled

R Servers

Tightly Coupled

A \leftarrow \text{matrix (1:10000, 100,100)}

library (pR)

S \leftarrow \text{sla.svd(A)}

b \leftarrow \text{list ()}

for (k in 1:dim (A) [ 1 ] ) {
  b [ k ] \leftarrow \text{sum ( A [ k, ] )}
}

m \leftarrow \text{mean ( A )}

d \leftarrow \text{sum ( A )}

Tightly Coupled

Data parallel jobs

Loosely Coupled

Task/Embarrassingly parallel jobs

Data Bank Server(s)

Memory & I/O Management

setValue() putValue()
Parallel Paradigm Hierarchy

Parallel Paradigms

Explicit Parallelism
- Rmpi
- rpvm

Implicit Parallelism

Task-Parallel
- taskPR

Data-Parallel

Hybrid: Task + Data Parallel
- pR
- taskPR

Intensive Inter-Process Communication
- pR
- RScaLAPACK

No or Limited Inter-Process Communication
- pRapply
- multicore
- snow
Rmpi May Not Be Ideal for All End-Users

- R-wrapper around MPI
- R is required at each compute node
- Executed as interpreted code, which introduces noticeable overhead
- Supports ~40 of >200 MPI-2 functions
- Users must be familiar with MPI details
- Can be especially useful for prototyping
Rmpi Matrix Multiplication Requires Parallel Programming Knowledge and is Rmpi Specific

```r
mm_Rmpi <- function(A, B, n_cpu = 1) {
  da <- dim(A)  ## dimensions of matrix A
  db <- dim(B)  ## dimensions of matrix B

  ## Input validation
  matrix_mult_validate( A, B, da, db )
  if( n_cpu == 1 )
    return(A %*% B)

  ## spawn R workers
  mpi.spawn.Rslaves( nslaves = n_cpu )

  ## broadcast data and functions
  mpi.bcast.Robj2slave( A )
  mpi.bcast.Robj2slave( B )
  mpi.bcast.Robj2slave( n_cpu )

  ## how many rows on workers?
  nrows_workers <- ceiling( da[1] / n_cpu )
  nrows_last <- da[1] - ( n_cpu - 1 ) *
    nrows_workers

  ## broadcast number of rows and foo to apply
  mpi.bcast.Robj2slave( nrows_workers )
  mpi.bcast.Robj2slave( nrows_last )
  mpi.bcast.Robj2slave( mm_Rmpi_worker )

  mm_Rmpi_worker <- function(){
    commrank <- mpi.comm.rank()
    if(commrank == ( n_cpu - 1 ))
    else

    mpi.gather.Robj(local_results, root = 0, comm = 1)
  }

  ## start partial matrix multiplication
  mpi.bcast.cmd( mm_Rmpi_worker() )

  ## gather partial results from workers
  local_results <- NULL
  results <- mpi.gather.Robj(local_results)
  C <- NULL

  ## Rmpi returns a list
  for(i in 1:n_cpu)
    C <- rbind(C, results[, i + 1 ])

  mpi.close.Rslaves()
  C
}
```

pR Matrix Multiplication

pR example:

library (RScaLAPACK)
A = matrix (c(1:256),16,16)
B = matrix (c(1:256),16,16)
C = sla.multiply (A, B)

Using R:

A = matrix (c(1:256),16,16)
B = matrix (c(1:256),16,16)
C = A % * % B
pR Parallel Plugin: Goals & Software Stack

- Couple general parallel computation capabilities within R
- Shield end-users from the complexities of parallel computing
- Enable execution of compiled and scripting codes together
Example: Do it Yourself

R End-User

A = matrix(c(1:256),16,16)
B = matrix(c(1:256),16,16)
C = mm(A, B)

pR glue code

mm(SEXP args) {
  double *a, *b;
  a = args.getPointer();
  b = args.getPointer();
  c = matrix_multiply(a, b);
  return c.getRObject();
}

3rd Party Code: HPC MPI code

matrix_multiply(a, b) {
  ...
  return c;
}
pR Overview

Parallel Programmer
R

End-User
R

C++/Fortran

In-memory Transfer

3rd Party

Data Distribution
Communication
Computation

3rd Party

MPI

3rd Party

MPI

3rd Party

MPI

3rd Party

MPI

3rd Party

MPI
C/C++/Fortran Plug-in to pR

```c
pR SEXP median(SEXP args)
{
  pR::pRParameters prpArgs (args);
  pR::pRVector <double> vec(prpArgs (0));
  vector<double>* myVec = vec.getNativePointer();
  // ... calculate myMedian for myVec ...
  pR::pRVector<double> ret (1);
  ret[0] = myMedian;
  return ret.getRObject();
}
```

```
dyn.load("SharedLibrary.so")
nums = as.numeric(1:1000000);
Result = .External("median", nums);
```
## Serial pR Performance over Python and R

<table>
<thead>
<tr>
<th>Method</th>
<th>Python</th>
<th>R</th>
<th>pR</th>
<th>pR Improv. over Python</th>
<th>pR Improv. over R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>0.300</td>
<td>0.108</td>
<td>0.030</td>
<td>10.0</td>
<td>3.6</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td>0.323</td>
<td>N/A</td>
<td>0.128</td>
<td>2.9</td>
<td>N/A</td>
</tr>
<tr>
<td>Average</td>
<td>0.071</td>
<td>0.049</td>
<td>0.012</td>
<td>5.9</td>
<td>4.1</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.420</td>
<td>0.020</td>
<td>0.019</td>
<td>22.1</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>18.1</strong></td>
<td><strong>2.8</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>C++</th>
<th>R</th>
<th>pR</th>
<th>R Overhead</th>
<th>pR Overhead</th>
<th>Times Improv. over R</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddot</td>
<td>0.003</td>
<td>0.131</td>
<td>0.003</td>
<td>0.128</td>
<td>0.001</td>
<td>43.7</td>
</tr>
<tr>
<td>dnrm2</td>
<td>0.024</td>
<td>0.044</td>
<td>0.026</td>
<td>0.014</td>
<td>0.000</td>
<td>1.7</td>
</tr>
<tr>
<td>dchdc</td>
<td>N/A</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.000</td>
<td>1.0</td>
</tr>
<tr>
<td>dsvdc</td>
<td>N/A</td>
<td>0.661</td>
<td>3.864</td>
<td>0.026</td>
<td>0.000</td>
<td>0.2</td>
</tr>
<tr>
<td>dgeev</td>
<td>0.002</td>
<td>0.557</td>
<td>0.004</td>
<td>0.027</td>
<td>0.002</td>
<td>139.3</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>0.002</strong></td>
<td><strong>0.002</strong></td>
<td><strong>37.2</strong></td>
</tr>
</tbody>
</table>
**pR Often Lowers Overhead Compared to R**

<table>
<thead>
<tr>
<th>Method</th>
<th>C++</th>
<th>R</th>
<th>pR</th>
<th>R Overhead</th>
<th>pR Overhead</th>
<th>Times Improv. over R</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddot</td>
<td>0.003</td>
<td>0.131</td>
<td>0.003</td>
<td>0.128</td>
<td>0.001</td>
<td>43.7</td>
</tr>
<tr>
<td>dnorm2</td>
<td>0.024</td>
<td>0.044</td>
<td>0.026</td>
<td>0.014</td>
<td>0.000</td>
<td>1.7</td>
</tr>
<tr>
<td>dchdc</td>
<td>N/A</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.000</td>
<td>1.0</td>
</tr>
<tr>
<td>dsvdc</td>
<td>N/A</td>
<td>0.661</td>
<td>0.516</td>
<td>0.026</td>
<td>0.000</td>
<td>1.3</td>
</tr>
<tr>
<td>dgeev</td>
<td>0.002</td>
<td>0.557</td>
<td>0.004</td>
<td>0.027</td>
<td>0.002</td>
<td>139.3</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.057</strong></td>
<td><strong>0.594</strong></td>
<td><strong>0.048</strong></td>
<td><strong>0.038</strong></td>
<td><strong>0.002</strong></td>
<td><strong>37.2</strong></td>
</tr>
</tbody>
</table>

Comparing C++, R and BridgeR External Method Calls in Seconds
pR to R Performance Comparison

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>R (sec)</th>
<th>pR (sec)</th>
<th>pR Factor Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>&lt; 1 \cdot 10^{-3}</td>
<td>1 \cdot 10^{-3}</td>
<td>0.0</td>
</tr>
<tr>
<td>64</td>
<td>1 \cdot 10^{-3}</td>
<td>8 \cdot 10^{-3}</td>
<td>0.01</td>
</tr>
<tr>
<td>128</td>
<td>1 \cdot 10^{-3}</td>
<td>8 \cdot 10^{-2}</td>
<td>0.07</td>
</tr>
<tr>
<td>256</td>
<td>5 \cdot 10^{-2}</td>
<td>1 \cdot 10^{-1}</td>
<td>0.41</td>
</tr>
<tr>
<td>512</td>
<td>8 \cdot 10^{-1}</td>
<td>2 \cdot 10^{-1}</td>
<td>5.63</td>
</tr>
<tr>
<td>1024</td>
<td>1 \cdot 10^{1}</td>
<td>6 \cdot 10^{-1}</td>
<td>17.18</td>
</tr>
<tr>
<td>2048</td>
<td>9 \cdot 10^{1}</td>
<td>4 \cdot 10^{0}</td>
<td>19.78</td>
</tr>
<tr>
<td>4096</td>
<td>7 \cdot 10^{2}</td>
<td>2 \cdot 10^{1}</td>
<td>28.32</td>
</tr>
</tbody>
</table>

Single processor matrix-multiplication testcase.

System: 32 node Intel-based Infiniband cluster running Linux. Each node contains two 3.4 GHz Pentium IV processors and 6GB of memory.
pR Achieves Superlinear Speedup vs. Serial R

Using a matrix-multiplication testcase on 4096x4096 matrices.
Matrix-multiplication testcase on 4096x4096 matrices
**pR Offers Parallel Scripting Computing with the Same Performance as Parallel Compiled Codes**

\[ S(p) = \frac{T_{\text{serial}}}{T_{\text{parallel}}(p)} \]

Using a matrix-multiplication testcase on 4096 x 4096 matrices, and comparing against a serial R implementation.

- **pR introduces minimal overhead and closely mirrors the performance of C**
- **Rmpi speedup is lower than the ideal**
pR Provides Simple, Efficient Third-Party Access to R

- Tightly-Coupled R Interface Between R and Third-Party Code
- Bidirectional Translation of Data Objects
- Memory Management: Direct Memory Access to R objects
- Compared to R: Average Speedup of 37x (with large variability)
- Compared to C: Negligible Overhead Induced
End-to-End Data Analytics

Domain Application Layer
- Biology
- Climate
- Fusion

Interface Layer
- Dashboard
- Web Service
- Workflow

Middleware Layer
- Automatic Parallelization
- Scheduling
- Plug-in

Analytics Core Library Layer
- Parallel
- Distributed
- Streamline

Data Movement, Storage, Access Layer
- Data Mover Light
- Parallel I/O
- Indexing
pR

- Lightweight middleware to bridge 3rd party codes
- pR-based packages:
  - RScaLAPACK
  - pRapply
  - pRstats
  - ....
- You can add your own libraries using pR
Parallel Paradigm Hierarchy

Parallel Paradigms

Explicit Parallelism
- Rmpi
- rpvm

Implicit Parallelism
- Task-Parallel
  - taskPR
- Data-Parallel
- Hybrid: Task + Data Parallel
  - pR
  - taskPR

Intensive Inter-Process Communication
- pR
- RScaLAPACK

No or Limited Inter-Process Communication
- pRapply
- multicore
- snow
What is RScaLAPACK?

- Motivation:
  - Many data analysis routines call linear algebra functions
  - In R, they are built on top of **serial** LAPACK library: [http://www.netlib.org/lapack](http://www.netlib.org/lapack)

- ScaLAPACK:
  - **parallel** LAPACK: [http://www.netlib.org/scalapack](http://www.netlib.org/scalapack)

- RScaLAPACK is a wrapper library to ScaLAPACK:
  - Also allows to link with ATLAS: [http://www.netlib.org/atlas](http://www.netlib.org/atlas)
Ex: RScaLAPACK Examples

Using RScaLAPACK:

- library (RScaLAPACK)
- sla.solve (A,b)
- sla.svd (A)
- sla.prcomp (A)

Using R:

- solve (A,b)
- La.svd (A)
- prcomp (A)
RScaLAPACK Functions

• library (RScaLAPACK)
• help (package=RScaLAPACK)
• help (sla.solve) or ?sla.solve
• example (sla.solve)
• demo (RScaLAPACK)
## Currently Supported Functions

<table>
<thead>
<tr>
<th>Serial R Functions</th>
<th>Parallel RScaLAPACK</th>
<th>RScaLAPACK Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svd</code></td>
<td><code>sla.svd</code></td>
<td>Compute a singular value decomposition of a rectangular matrix</td>
</tr>
<tr>
<td><code>eigen</code></td>
<td><code>sla.eigen</code></td>
<td>Computes the Eigen values and Eigen vectors of symmetric square matrix</td>
</tr>
<tr>
<td><code>chol</code></td>
<td><code>sla.chol</code></td>
<td>Computes the Choleski factorization of a real symmetric positive definite square matrix</td>
</tr>
<tr>
<td><code>chol2inv</code></td>
<td><code>sla.chol2inv</code></td>
<td>Invert a symmetric, positive definite, square matrix from its Choleski decomposition</td>
</tr>
<tr>
<td><code>solve</code></td>
<td><code>sla.solve</code></td>
<td>This generic function solves the equation $a^*x=b$ for $x$</td>
</tr>
<tr>
<td><code>qr</code></td>
<td><code>sla.qr</code></td>
<td>computes the QR decomposition of a matrix</td>
</tr>
<tr>
<td><code>factanal</code></td>
<td><code>sla.factanal</code></td>
<td>Perform maximum-likelihood factor analysis on a covariance matrix or data matrix using RScaLAPACK functions</td>
</tr>
<tr>
<td><code>factanal.fit.mle</code></td>
<td><code>sla.factanal.fit.mle</code></td>
<td>Perform maximum-likelihood factor analysis on a covariance matrix or data matrix using RScaLAPACK functions</td>
</tr>
<tr>
<td><code>prcomp</code></td>
<td><code>sla.prcomp</code></td>
<td>performs a principal components analysis on the given data matrix using RScaLAPACK functions</td>
</tr>
<tr>
<td><code>princomp</code></td>
<td><code>sla.princomp</code></td>
<td>performs a principal components analysis on the given data matrix using RScaLAPACK functions</td>
</tr>
<tr>
<td><code>varimax</code></td>
<td><code>sla.varimix</code></td>
<td>These functions rotate loading matrices in factor analysis using RScaLAPACK functions</td>
</tr>
<tr>
<td><code>promax</code></td>
<td><code>sla.promax</code></td>
<td>These functions rotate loading matrices in factor analysis using RScaLAPACK</td>
</tr>
</tbody>
</table>
Scalability of pR: RScaLAPACK

\[ R > \text{solve}(A,B) \quad \text{pR} > \text{sla.solve}(A, B, NPROWS, NPCOLS, MB) \]

A, B are input matrices; NPROWS and NPCOLS are process grid specs; MB is block size

Architecture: SGI Altix at CCS of ORNL with 256 Intel Itanium2 processors at 1.5 GHz; 8 GB of memory per processor (2 TB system memory); 64-bit Linux OS; 1.5 TeraFLOPs/s theoretical total peak performance.
Changing the Processor Grid in RScaLAPACK

library (RScaLAPACK)
A = matrix(rnorm(128*128),128,128)
?sla.gridInit

Changing processor grid:

sla.gridInit(NPROCS=8)
x = sla.solve (A, NPROWS=4)
sla.gridExit()
RedHat and CRAN Distribution

CRAN R-Project

Available for download from R’s CRAN web site (www.R-Project.org) with 37 mirror sites in 20 countries.

http://cran.r-project.org/web/packages/RSCaLAPACK/index.html

http://rpmfind.net/linux/RPM/RByName.html
RScaLAPACK Installation

- **Download** RScaLAPACK from R’s CRAN web-site
- **Install dependency packages:**
  - Install R
  - MPI (Open MPI, MPICH, LAM MPI)
  - ScaLAPACK (with the proper MPI distribution)
  - Setup environment variables
    
    ```
    export LD_LIBRARY_PATH=<path2deps>/lib:$LD_LIBRARY_PATH
    ```

- **Install RScaLAPACK:**
  - R CMD INSTALL --configure-args="--with-f77
    --with-mpi=<MPI install home directory>
    --with-blacs=<blacs build>/lib
    --with-blas=<blas build>/lib
    --with-lapack=<lapack build>/lib
    --with-scalapack=<scalapack build>/lib"

    RScaLAPACK_0.6.1.tar.gz
Parallel Paradigm Hierarchy

Explicit Parallelism
- Rmpi
- rpvm

Task-Parallel
- taskPR

Implicit Parallelism

Data-Parallel

Hybrid: Task + Data Parallel
- pR
- taskPR

Intensive Inter-Process Communication
- pR
- RScaLAPACK

No or Limited Inter-Process Communication
- pRapply
- multicore
- snow
R’s `lapply` Method is a Natural Candidate for Automatic Parallelization

- Examples: Bootstrapping, Monte Carlo, etc.
Existing R Packages with Parallel \texttt{lapply}

- **multicore**
  - Limited to single-node execution
  - \texttt{mclapply}

- **snow**
  - Built on \texttt{Rmpi} – uses MPI for communication
  - Requires users to explicitly manage R dependencies (libraries, variables, functions)
  - \texttt{clusterApply}
snow Example

```r
library(snow);
library(abind);
x = as.list(1:16);
axis=0;
y = matrix(1:12,3,4);
fn <- function(){
  z = y+100;
  b = dim(abind(y,z,along=axis))
}
cl = makeCluster(numProcs, type = "MPI")
clusterApply(cl, x, fn);
stopCluster(c1);
clusterExport(cl, list(axis, y));
clusterEvalQ(cl, library(abind));
```

Explicitly send libraries, functions, and variables
pRapply Example

pRlapply (list, fn, procs=2, cores=2)

Using pRapply:

```
1 library(pRapply);
2 library(abind);
3 x = as.list(1:16);
4 axis=0;
5 y = matrix(1:12,3,4);
6 fn <- function(){
7    z = y+100;
8    b = dim(abind(y,z,along=axis))
9  }
10 pRlapply(x, fn)
```

Using R:

```
1
2 library(abind);
3 x = as.list(1:16);
4 axis=0;
5 y = matrix(1:12,3,4);
6 fn <- function(){
7    z = y+100;
8    b = dim(abind(y,z,along=axis))
9  }
10 lapply(x, fn)
```
pR Automatic Parallelization Uses a 2-Tier Execution Strategy

R End-User System

\texttt{lapply(list, function)}

R Worker

\(C_i = i^{\text{th}} \text{ core}\)
pRapply: ~Ideal Speedup vs. snow

Multi-node speedup for pR and snow compared to the serial R implementation using a testcase that sums 4096 vectors, each of size $2^{24}$. 

---

*SDM Center*  
*NC State University*  
*Oak Ridge National Laboratory*  
*Office of Science*  
*U.S. Department of Energy*
**pRapply: Ideal or Better Weak Scaling**

\[
\text{Work}(p) = W^*p
\]

Weak scaling comparison of snow and pR in a 64-node hybrid multi-node multi-core environment; each process sums a \(2^{24}\) element vector.
**pRapply:** Like *multicore*, but supports hybrid multi-node, multi-core execution.

The pR overhead and total execution time compared to the multicore package over powers of two processes (1-64) using a testcase that *sleeps* 120 times for 4 seconds.
pR Achieves Load-Balanced Performance

pR compute node execution heterogeneity in a hybrid multi-node multi-core Environment using 16 nodes to each sum a $2^{24}$ element vector.
pR Major Contributions

• Enables integrating efficient serial codes into R using in-memory data transfers
• Provides parallel computing capabilities within R
• Enables automatic parallelization of data-parallel codes in hybrid multi-core and multi-node environments
Acknowldgements

• Guruprasad Kora, ORNL
• Paul Breimyer, Lincoln Lab
• Srikanth Yoginath, ORNL
• David Bauer, GA Tech
Publications


2. **Parallel R for High Performance Analytics: Applications to Biology;** Scientific Data Management (Book In Press), 2008; Nagiza F. Samatova, Paul Breimyer, Guruprasad Kora, Chongle Pan, Srikanth Yiginath; Editors: Arie Shoshani, Doron Rotem, Chandrika Kamath

3. **pR: Automatic Parallelization of Data-Parallel Statistical Computing Codes for R in Hybrid Multi-Node and Multi-Core Environments;** International Association for Development of Information Society (IADIS) 2009; Paul Breimyer, Guruprasad Kora, William Hendrix, Nagiza F. Samatova

4. **RScaLAPACK on CRAN:** http://cran.r-project.org/mirrors.html


Session 2: Input and Output

To input data:

- \texttt{# scan()} command to scan data
  \texttt{x=scan()}
- # enter numbers, one or more on a line, and type a blank line to end
- \texttt{# read.table()} command to read data from file
  \texttt{y=read.table("myfile")}

To access vector elements:

- \texttt{# 2nd element of x}
  \texttt{x[2]}
- # first five elements of x
  \texttt{x[1:5]}
- # all but the 3rd element of x
  \texttt{x[-3]}
- # values of x that are < 40
  \texttt{x[x<40]}
- # values of y such that x is < 40
  \texttt{y[x<40]}

To perform operations:

- \texttt{# mathematical operations on vectors}
  \texttt{y=c(3,2,4,3,7,6,1,1)}
  \texttt{x+y; 2*y; x*y; x/y; y^2}
Session 4: Plotting in R

• To create a vector:
  – \( x=c(12,32,54,33,21,65) \) # help (c)