Adventures in Supercomputing with R

Lecture 8: Distributed Memory Methods

George Ostrouchov

Oak Ridge National Laboratory and University of Tennessee

2022/02/20 (updated: 2022-04-06)

From Last Lecture

- Login issue to karolina resolved with providing user-name!

```
## login without user-name assumes same user-name as laptop
ssh karolina.it4i.cz

## providing karolina user-name resolves the issue
ssh my-karolina-user-name@karolina.it4i.cz
```

- Intermittent access denial to HDF5 file
  - Example of locking shared resources while modified

```
## mnist_read.R now opens MNIST with read-only access.
## Eliminates locking the file. Others can use it at the same time.
h5tr = H5Fopen(paste0(dir, "train.hdf5"), flags="H5F_ACC_RDONLY")
```

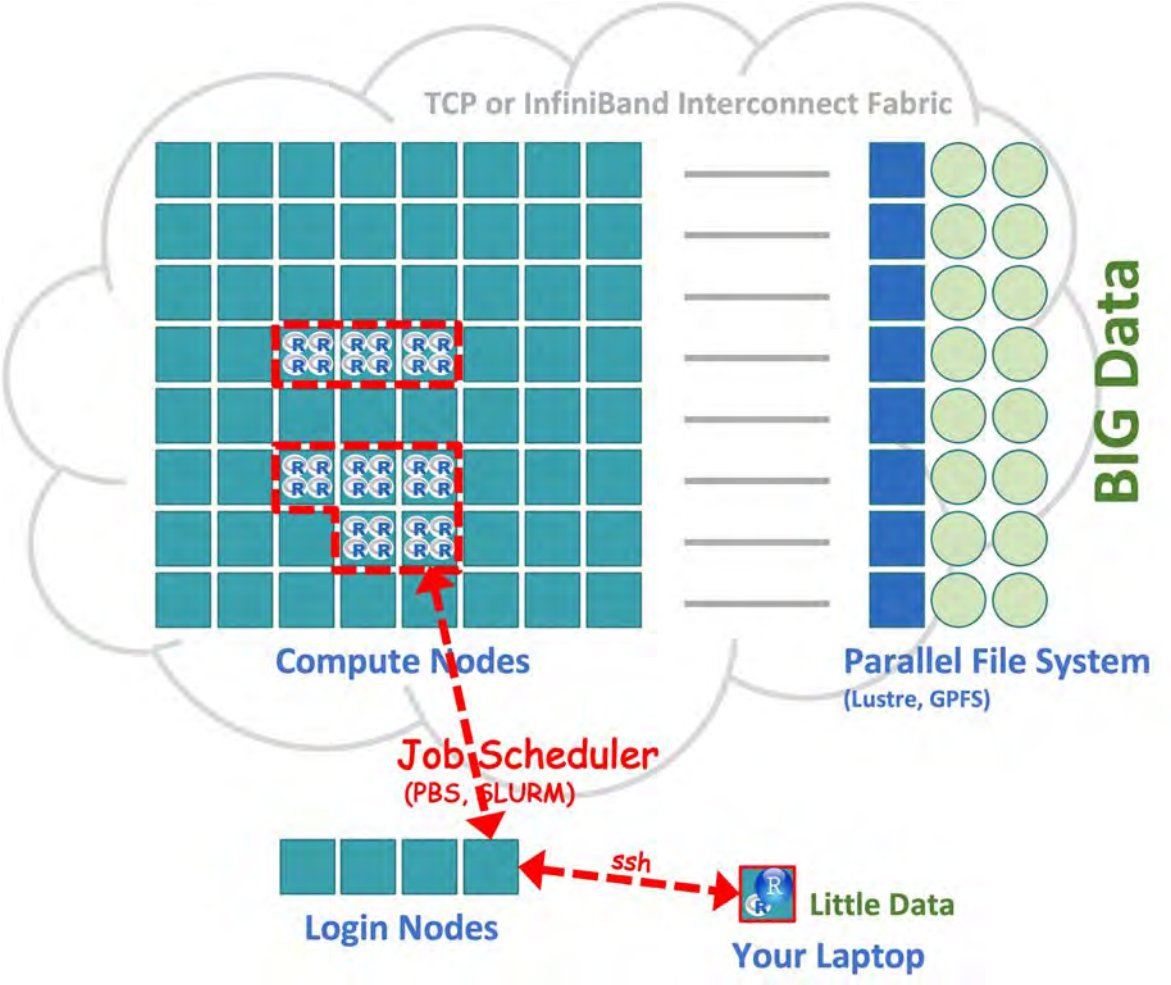
Go over Exercise 7 ...

Introduce Exercise 8 ...

Profiling to find speedup needs

```
Rprof()
##
## your code to profile
##
Rprof(NULL)
summaryRprof()
```

Running Distributed on a Cluster



Parallel Computing Models

Shared memory parallel computing

- Processors have access to all memory
  - Locking mechanism
- Kinds: unix fork, pthreads, OpenMP, OpenACC
- Libraries: OpenBLAS, MKL, FlexiBLAS, PLASMA, MAGMA, etc.

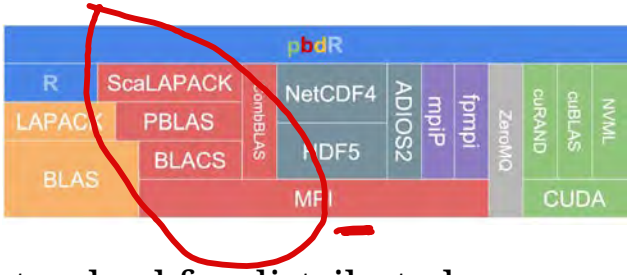
Distributed memory parallel computing

- Processors have only local memory
  - Communication mechanism
- Kinds: MPI, MapReduce, DataFlow
- Libraries: OpenMPI, ScaLAPACK, PETSci, Trillinos, etc.

Distributed Programming Models

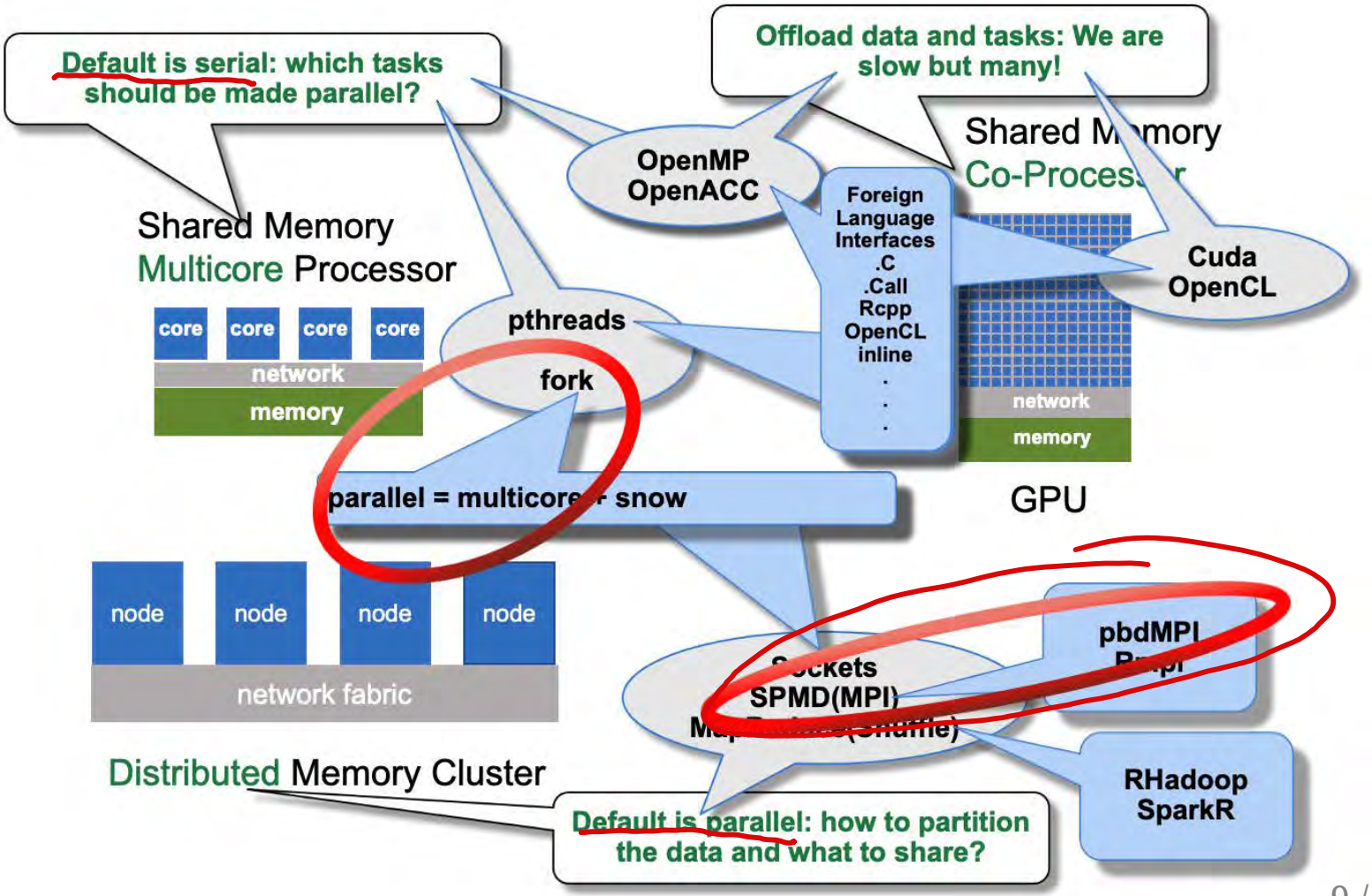
- Manager-workers
  - More common in shared memory (e.g. `mclapply()`)
- Single program, multiple data (SPMD)
  - Most common on supercomputer clusters
  - Most scalable because based on collaboration rather than control
  - Concept behind MPI design
- MapReduce
  - Common in data processing but falling out of favor
  - Slow because includes all-to-all shuffle communication
- Dataflow
  - Dependency graph directed, still evolving
  - Sometimes combined with MPI for distributed control

pbdR-ScaLAPACK-PBLAS-BLACS-MPI



- MPI: Message Passing Interface - *de facto* standard for distributed communication in supercomputing
    - Used for data mostly via collective communication - high level
    - `pbdMPI`, `kazaam`, and `cop` R packages
- ScaLAPACK: Scalable LAPACK - Distributed version of LAPACK (uses PBLAS/BLAS but not LAPACK)
    - 2d Block-Cyclic data layout - mostly automated in `pbdDMAT` package
    - BLACS: Communication collectives for distributed matrix computation
    - PBLAS: BLAS - distributed BLAS (uses shared memory BLAS within blocks)
    - `pbdDMAT` and `pbdML` R packages - most matrix operations identical to serial through overloading operators and `ddmatrix` class

R Interfaces to Low-Level Native Tools



Single Program Multiple Data (SPMD)

Hello world!

```
suppressMessages(library(pbdMPI))

my_rank = comm.rank()
ranks = comm.size()
msg = paste0("Hello World! My name is Empi", my_rank,
             ". We are ", ranks, " identical siblings.")
cat(msg, "\n")

finalize()
```

One code and a parallel mindset

A generalization of a serial code

Many rank-aware operations are automated

No manager, it is all cooperation

Explicit point-to-point communications are an advanced topic

hello_world.R

```
suppressMessages(library(pbdMPI))

my_rank = comm.rank()
ranks = comm.size()
msg = paste0("Hello World! My name is Empi", my_rank,
             ". We are ", ranks, " identical siblings.")
cat(msg, "\n")

finalize()
```

```
[gostrouc@cn054.karolina mpi]$ mpirun -np 4 Rscript hello_world.R
Hello World! My name is Empi0. We are 4 identical siblings.
Hello World! My name is Empi2. We are 4 identical siblings.
Hello World! My name is Empi3. We are 4 identical siblings.
Hello World! My name is Empi1. We are 4 identical siblings.
```

comm.cat( )

=

Order!

Can even mix words from different ranks.

comm.cat( )
comm.print( )

use barrier( ) to prevent mixing

comm.chunk() function

You may need to install this package from GitHub:

```
remotes::install_github("RBigData/pbdIO")
```

`comm.chunk()` divides a number of items into chunks and returns information specific to each *rank*.

```
comm.chunk(N, form = "number", type = "balance", lo.side = "left", al
           p = comm.size(), rank = comm.rank())
```

right

when p divides N with no remainder, all types are same

Generalizes `get.jid()`, in `pbdMPI` package.

comm.chunk() examples

```
library(pbdIO)
```

```
## Loading required package: pbdMPI
```

p = comm.size()
rank = comm.rank()

```
comm.chunk(61, p = 8, rank = 1)
```

```
## [1] 8
```

```
comm.chunk(61, p = 8, all.rank = TRUE)
```

```
## [1] 8 8 8 8 8 7 7 7
```

rank: 0 1 2 3 4 5 6 7

```
comm.chunk(61, p = 8, rank = 5)
```

```
## [1] 7
```

lo.side="right" is default
with lo.side="left"
we get
7 7 7 8 8 8 8

comm.chunk() examples

```
comm.chunk(61, form = "vector", p = 8, rank = 0)
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
comm.chunk(61, form = "vector", p = 8, rank = 1)
```

```
## [1]  9 10 11 12 13 14 15 16
```

```
finalize()
```

All distributed codes need a finalize() to shut down MPI properly.

rf_mpi.R

```
source("../mnist/mnist_read.R")
suppressMessages(library(randomForest))
suppressMessages(library(pbdIO))
comm.set.seed(seed = 123, diff = TRUE)

n = nrow(train)
n_test = nrow(test)
my_trees = comm.chunk(512)
my_test_rows = comm.chunk(nrow(test), form = "vector")

my_rf = randomForest(train, y = train_lab, ntree = my_trees, norm.vot
all_rf = allgather(my_rf)
all_rf = do.call(combine, all_rf)

my_pred = as.vector(predict(all_rf, test[my_test_rows, ]))

correct = reduce(sum(my_pred == test_lab[my_test_rows]))
comm.cat("Proportion Correct:", correct/n_test, "\n")

finalize()
```

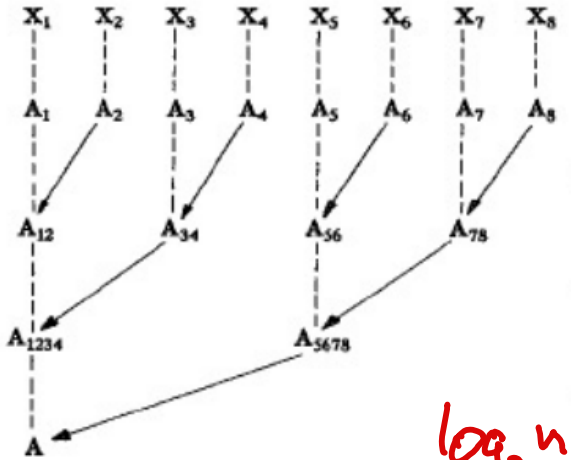
compare to serial code at mnist/mnist_rf-serial.R

pbdMPI reduce and gather collective communication

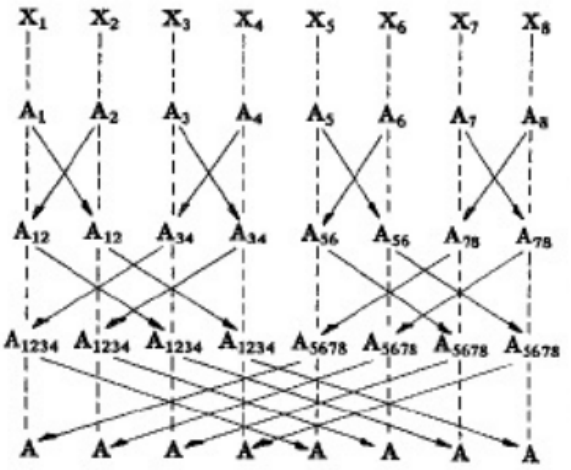
all are associative and commutative operations

$A = \sum_{i=1}^{8} X_i$ Reductions: +, *, min, max, (more via package **cop**)

reduce(X)



allreduce(X)



$\log_2 n$ steps for n objects

$A = [X_1|X_2|\cdots|X_8]$

gather(X)

allgather(X)