High Performance R

David Walling, Yaakoub Y. El Khamra

walling@tacc.utexas.edu

Yaakoub@tacc.utexas.edu

Disclaimer

I am just the guy who built the packages, has a decent understanding of how they work, how they fail and some understanding of how to use it. I am not an R expert, I just know enough R to get by. There are R experts in the room and they will field all the R questions.

Parallel R @ TACC

- We will be rolling out a new Parallel R module shortly
- Preview is installed in a publicly accessible directory
- Packages include: pbdR, Rmpi, SNOW and many others
- Heavily optimized build with Intel compilers, MKL and an alpha release of MVAPICH2
- Supports automatic offloading for the Intel Xeon Phi

Make it go faster

- Running R on compute nodes
  - Most R packages/functions use a single core
  - You would be using 1/16 of the compute power of the compute node
- Introduce parallelism to R
  - Threads or shared memory: limited to one node
    - Implicit: minor changes (if at all) required
    - Explicit: some assembly required
  - MPI/Sockets or distributed memory: one or more nodes (explicit only)
- Use Xeon Phi Coprocessor?

Xeon Phi Hands-on

- ssh into stampede (or exit the interactive node if you are on one)
- Go to your $HOME directory (type `cd` and hit enter)
- Extract the lab materials and go to the Xeon Phi lab directory

```
tar  zxf  ~train00/parallel_r.tar.gz
cd parallel_r/XeonPhi
```

- Check that you have the right allocation
  - If you registered through the TACC portal, your allocation is: 20130927DataIntensiv
  - If you registered through the XSEDE portal, your allocation is: TG-TRA120009
- By default, all job submission scripts have the following

```
#SBATCH -A 20130927DataIntensiv
##SBATCH -A TG-TRA120009
```

Xeon Phi Hands-on

- The first line is an SBATCH/SLURM directive, the second line is a comment
- If you registered through the TACC portal, submit as is
- `sbatch ex_07.slurm`
- If you registered through the XSEDE portal, comment the first line (insert a #) and uncomment the second line, then submit:
- `sbatch ex_07.slurm`
- You can use your own allocation, just edit the allocation id
- More information on job submission at the stampede user guide: http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide

Xeon Phi Hands-On

- Submit the Xeon Phi Example
- This example will run 4 cases: single thread on the host, 16 threads on the host, 16 threads on host + 240 threads on the coprocessor, 16 threads on host + 480 threads on two coprocessors
- Examine the output in the slurm-<job id>.out file. What do you notice?

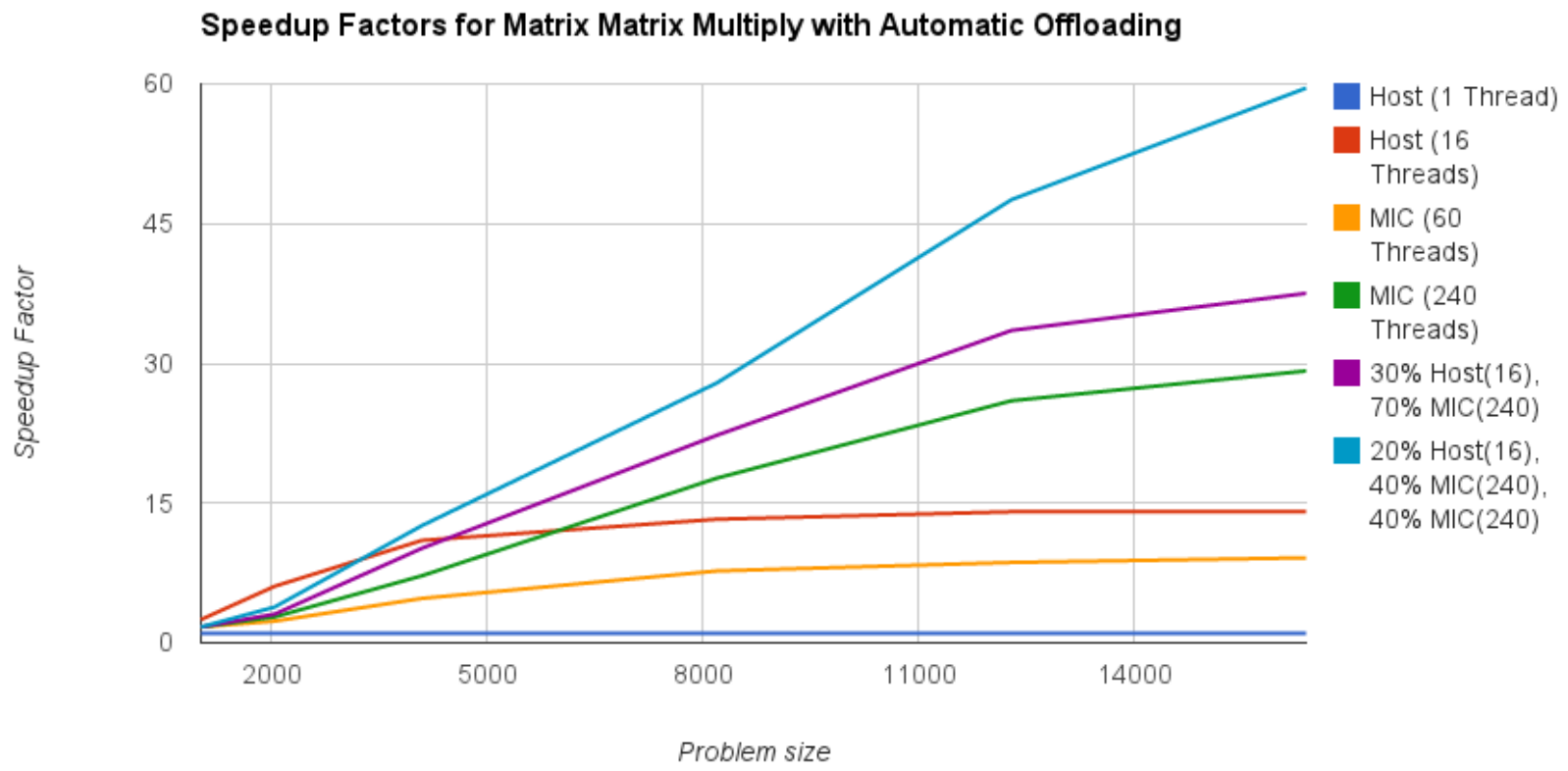
Xeon Phi Coprocessor

- For some routines, R calls BLAS/LAPACK routines
- MKL provides BLAS/LAPACK routines that can "offload" to the Xeon Phi Coprocessor, reducing total time to solution
- Automatic offloading will not always happen:
  - Depends on function being called
  - Depends on problem/argument size

Xeon Phi Coprocessor

- No changes need to be made to your code
- Some environment variables need to be added to your script
  - MKL_MIC_ENABLE=1
  - MIC_MKL_NUM_THREADS=240
  - MKL_HOST_WORKDIVISION=0.3
  - MKL_MIC_WORKDIVISION=0.7
- You have more/slightly-different environment variables for compute nodes with 2 Xeon Phi coprocessors

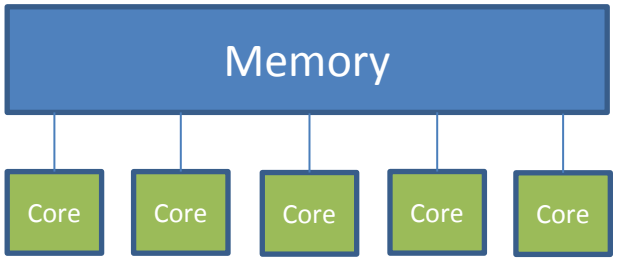
Xeon Phi Coprocessor

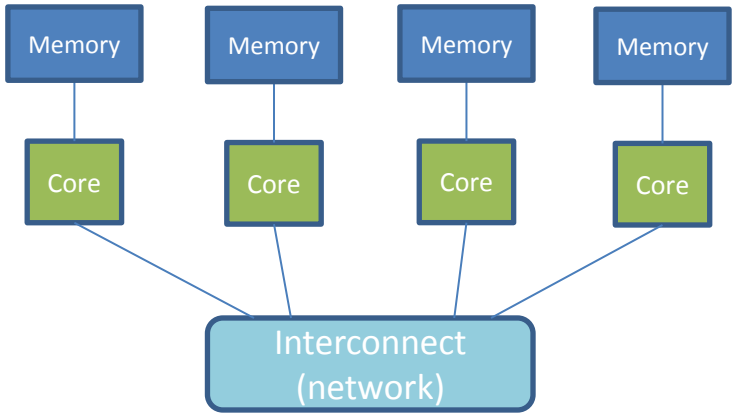


Parallel Packages in R

- We will introduce some of the packages and a bit about distributed versus shared memory
- More detailed documentation is available through CRAN
- We will go through some examples in the hands-on section
- Many packages have demos

Aside: Shared vs Distributed Memory





Shared Memory: Implicit Parallel Packages in R

- pnmath package uses OpenMP to provide implicit parallelism by providing internal replacements to R functions
  - Last release was June of 2012
- mchof provides functions such as Filter, Fold, ZipWith and Partition (mcFilter, mcFold, mcZipWith, mcPartition) and a few others
- Sprint: parallel versions of some libraries (same as mchof): pcor instead of cor, psvm instead of svm, pRSadvance instead of advance
- Rdsm: thread based, similar threading paradigm as unix threads

Shared Memory: Explicit Parallel Packages

- fork: wraps over unix fork, signal, wait, waitpid, kill and exit
- foreach: iterations over elements in a collection without the use of an explicit loop counter

Do Not Cross the Streams

- Do not oversubscribe a machine by using too many threads:
  - Do not combine threaded blas/lapack calls in R with shared memory packages
  - Do not combine multiple shared memory packages
- Check your answers: Race conditions
- You _can_ mix distributed memory and shared memory parallel packages, but be careful of oversubscribing a node

Shared Memory Hands-On

- Coming soon
- R and intel compilers:
  - Much faster than gcc, much better performance
  - Sometimes threads hang
- I am working on it...

Distributed Memory: MPI

- MPI: Message Passing Interface
  - Standard came out in 1994, currently version 3.0
  - Many implementations: mvapich2, Intel MPI and OpenMPI
  - Available on all HPC clusters
- Two types of communication
  - Point to Point: send/receive
  - Collective: broadcast, scatter, gather, reduce

Distributed Memory: MPI

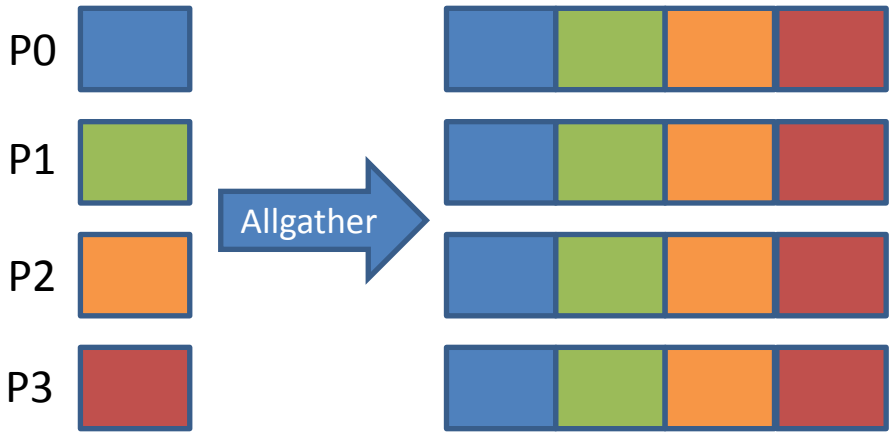
- Advantages of MPI
  - High bandwidth, low latency
  - Portable, standard across clusters (for the most part)
- Disadvantages of MPI
  - Some work involved, possible change of algorithm
- There are libraries that are built on top of MPI (snow, pbdR) that provide higher level functionality

Distributed Memory: MPI

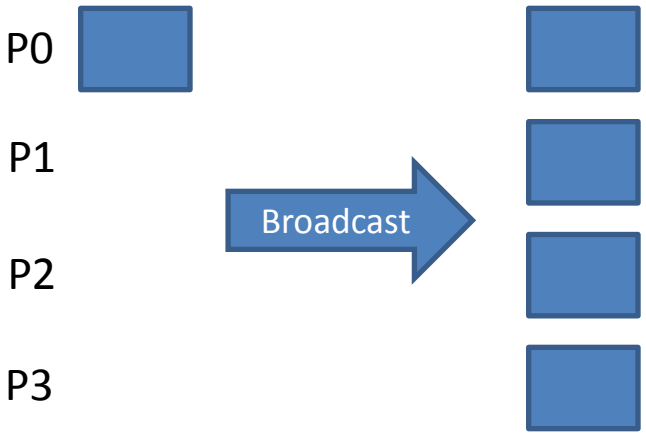
- Keep in mind:
  - Communicators are collections of processes, running on different cores/processors
  - Each process has a unique identifier in the communicator called a rank
  - Total number of processes in a communicator is the size of the communicator
- Send/receive do exactly that: send “stuff” from one rank to another
- Allgather: gathers data from all ranks and distributes the combined data to all ranks
- Barrier: you shall not pass… until all ranks in a communicator reach the barrier.

Allgather and Broadcast

MPI_Allgather



MPI_Broadcast



For more on MPI

- TACC Two Day Training: http://www.tacc.utexas.edu/user-services/training/
- TACC course material online: http://www.tacc.utexas.edu/user-services/training/course-materials
- Google “MPI tutorials”

Rmpi vs pbdMPI vs SNOW

	Rmpi	pbdMPI	SNOW
Low Level MPI calls	Yes	Yes	No
SPMD	Yes	Yes	Yes
Master/Worker	Yes	No	Yes
High Level Functionality	No	Yes	Yes
Trivial/embarrassingly parallel	You have to write it	You have to write it	Built-in
Complex parallel execution	You have to write it	You have to write it	No
Support/development/maintenance	Decent	Good	Only one developer, infrequent releases

Rmpi

- Interfaces a lot of MPI functions to R
- Supports SPMD (single program multiple data) and Master/Worker
  - master process spawns workers to work in parallel
  - SPMD commands are issued from the perspective of the current process, all processes are equal
- Bug in MVAPICH2 breaks spawning

  do _not_ use mpi.spawn.Rslaves

Rmpi: HelloWorld

```
library(Rmpi)
```

Initialize mpi

```
id <- mpi.comm.rank(comm = 0)
np <- mpi.comm.size(comm = 0)
hostname <- mpi.get.processor.name()
```

Retrieve the rank of the process

Retrieve the size of the communicator

Retrieve the hostname

```
msg <- sprintf("Hello world from
process %03d of %03d, on host %s\n",
id, np, hostname)
```

Create a message containing rank, size and hostname

```
cat(msg)
```

Print the message: all processes have their own stdout that is collected into one file

```
mpi.barrier(comm = 0)
```

Wait for all at the barrier

```
mpi.finalize()
```

Shutdown MPI and all the processes

Rmpi Hands-On

- Let's go one directory up: `cd ..`
- And into the Rmpi example directory: `cd Rmpi`
- We have 3 examples: `ex_01.slurm`, `ex_02.slurm`, `ex_03.slurm`
- Edit scripts to use correct project.
- `sbatch ex_01.slurm`
- `sbatch ex_02.slurm`
- `sbatch ex_03.slurm`
- And let's have a look at the input/output
- ex_01 and ex_02 both print hello world in two different ways
- ex_03 writes files, be careful when writing files in parallel!

pbdR

- http://r-pbd.org/
- More recent package from RDAV at NICS. It includes:
  - pbdMPI: like Rmpi, slightly different
  - pbdDMAT, pbdSLAP, pbdBASE: dense distributed matrices in parallel, scalable linear algebra (pblas/scalapack) and host of matrix and statistics operations: everything from `[` to lm.fit()
- Unlike Rmpi, pbdMPI supports SPMD almost exclusively (i.e. no spawning)
- Supports some high level functionality: global any and all, apply and lapply, global sort, etc.

pbdMPI Hello World (and extras)

```
library(pbdMPI, quiet = TRUE)
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

msg <- sprintf("Hello world from process %d\n", .comm.rank)
comm.cat("Say hello:\n", quiet = TRUE)
comm.cat(msg, all.rank = TRUE)

k <- 10
x <- rep(.comm.rank, k)
comm.cat("\nOriginal x vector:\n", quiet = TRUE)
comm.print(x, all.rank = TRUE)

y <- allgather(x, unlist = TRUE)
A <- matrix(y, nrow = k, byrow = FALSE)
comm.cat("\nAllgather matrix (only showing process 0):\n", quiet = TRUE)
comm.print(A)

finalize()
```

Notice no mpi. Prefix

Initialize mpi

Get size and rank

Say hello from all processes

Create a vector, print it from all processes, ordered by rank

Call allgather

Create a matrix from the result of the allgather

Print the result to screen

Finalize mpi (shutdown and terminate processes)

pbdR Hands-On

- One directory up and to the pbdR examples: `cd ../pbdR`
- We have 2 examples, `ex_04.slurm ex_05.slurm`
- Edit scripts to use correct project.
- `sbatch ex_04.slurm`
- `sbatch ex_05.slurm`
- And let's look at the input/output
- Note in ex_04: pbdMPI looks a lot like Rmpi
- Note in ex_05: we are using 2 MPI processes, one per node and running the demos included in pbdR: scatter, sort and solve, check the –N and –n in slurm script

SNOW: Simple Network Of Workstations

- For embarrassingly parallel problems
- Simple interface, can be built on top of Rmpi or sockets (we use Rmpi)
- You do not need to know Rmpi to make the most out of SNOW
- Functionality includes: clusterCall, clusterApply, clusterSplit, parRapply, parCapply, parLapply, parSapply, parMM

SNOW Usage

- SNOW makeMPIcluster
  - same bug in MPI spawning, do _not_ use
  - getMPIcluster does the same thing
- To launch SNOW scripts on TACC systems:

  ibrun RMPISNOW < ./SimpleSNOW.R
- RMPISNOW launches R (not Rscript)
- ibrun is TACC's mpirun/mpiexec

SNOW Example

```
library(Rmpi)
library(snow)
cluster <- getMPIcluster()

sayhello <- function()
{ info <- Sys.info()[c("nodename", "machine")]
  paste("Hello from", info[1], "with CPU type", info[2]) }

names <- clusterCall(cluster, sayhello)
print(unlist(names))

parallelSum <- function(m, n)
{       A <- matrix(rnorm(m*n), nrow = m, ncol = n)
        row.sums <- parApply(cluster, A, 1, sum)
        print(sum(row.sums))   }

parallelSum(500, 500)

stopCluster(cluster)
```

You always need Rmpi for SNOW

Always getMPIcluster, never makeMPIcluster

A function to say hello and call function on the whole cluster

Function to compute row sums.

Note that we use parApply. parApply is the parallel version of the R apply function. In the arugment list: '1' indicates rows, '2' indicates columns, 'c(1,2)' indicates rows and columns. 'sum' is the function.

Invoke the function

Stop the cluster, shutdown mpi and finalize

Snow Hands-On

- One directory up and to SNOW:
  `cd ../SNOW`
- One example, `ex_06.slurm`
- Edit scripts to use correct project.
- `sbatch ex_06.slurm`
- Check the input and output
- Note we need Rmpi for SNOW
- Always getMPIcluster, guaranteed failure otherwise
- Always stopCluster, otherwise job will only terminate when time in the queue is done