Parallel Computing with the *Bioconductor* Amazon Machine Image

Valerie Obenchain
vobencha@fhcrc.org

Fred Hutchinson Cancer Research Center
Seattle, WA

August 2014

Introduction

This presentation provides an introduction to parallel computing with the *Bioconductor* Amazon Machine Image (AMI). It is designed for those who are familiar with *R* and *Bioconductor*.

We'll review parallel infrastructure available in *Bioconductor* packages, specifically *BiocParallel* and *GenomicFiles*. Problem types well suited for parallel execution are discussed as well as advantages / disadvantages of computing in a single machine vs cluster environments.

Exercises develop familarity with the AMI resources and explore data import and storage options. Steps in a copy number analysis are used to demonstrate parallel applications over different dimensions and problem configurations.

Parallel programming

Computationally intensive problems that involve many independent calculations

Involves:

- Decomposing an algorithm or data into parts ('split')
- Distributing the tasks to multiple processors to be worked on simultaneously ('apply')
- Gathering results ('combine')

Considerations:

- Type of parallel architecture
- Type of processor communication

Candidate problems:

- Simulations, bootstrap, cross validation, local convergence algorithms
- Repeated operation on many genomic ranges or files such as coverage, counting, normalization, etc.

Other approaches

Limiting resource consumption:

- Restriction: Appropriate when query only requires a fraction of the data (e.g., `ScanBamParam`)
- Compression: Represent the same data with fewer resources (`Rle`, `Views`, `CompressedList` classes)
- Iterating: Chunking through data to meet resource constraints (`yieldSize`)

... and more at `http://www.imstat.org/sts/future_papers.html`.

Packages with parallel infrastructure

parallel

Incorporates *multicore* and *snow* packages; included in *base R*.

BatchJobs

Schedules jobs on batch systems

foreach

Framework with do* backends, *doMC*, *doSNOW*, *doMPI*, etc.

BiocParallel

Incorporates *parallel* and *BatchJobs* packages.

(... and there are others)

BiocParallel

Why use *BiocParallel*?

- Load one package for all parallel back-ends
- Mirrored terminology of existing *apply functions bplapply, bpmapply, bpsapply, bpvec ...
- Unified interface to back-ends via *param* object

```
> registered()

$MulticoreParam
class: MulticoreParam; bpisup: TRUE; bpworkers: 4; catch.errors: TRUE
setSeed: TRUE; recursive: TRUE; cleanup: TRUE; cleanupSignal: 15; verbose:
  FALSE

$SnowParam
class: SnowParam; bpisup: FALSE; bpworkers: 4; catch.errors: TRUE
cluster spec: 4; type: PSOCK

$BatchJobsParam
class: BatchJobsParam; bpisup: TRUE; bpworkers: NA; catch.errors: TRUE
cleanup: TRUE; stop.on.error: FALSE; progressbar: TRUE

$SerialParam
class: SerialParam; bpisup: TRUE; bpworkers: 1; catch.errors: TRUE
```

GenomicFiles package (devel branch)

`reduceByFile()`, `reduceByRange()`:

- Use *BiocParallel* under the hood to perform parallel computations by file or by range
- MAP and REDUCE concepts to reduce the dimension of the data

`reduceByYield()`:

- Iterates through a file reducing output to single result (no built-in parallelism)

Single machine with multiple CPUs

- Multi-core approach uses *fork* system call to create workers
- Shared memory enables data sharing and code initialization
- No Windows support
- Appropriate for computationally-limiting problems, not memory-limiting

```
> bpworkers()

[1] 4

> MulticoreParam()

class: MulticoreParam; bpisup: TRUE; bpworkers: 4; catch.errors: TRUE
setSeed: TRUE; recursive: TRUE; cleanup: TRUE; cleanupSignal: 15; verbose:
  FALSE
```

Clusters

Ad hoc cluster of multiple machines:

- Create a cluster of workers communicating with MPI or sockets
- Memory not shared; all data must be passed, code initialized
- Need SSH access to all machines

```
> SnowParam(workers = 4, type = "PSOCK")

class: SnowParam; bpisup: FALSE; bpworkers: 4; catch.errors: TRUE
cluster spec: 4; type: PSOCK

> SnowParam(workers = 4, type = "MPI")

class: SnowParam; bpisup: FALSE; bpworkers: 4; catch.errors: TRUE
cluster spec: 4; type: MPI
```

Cluster with formal scheduler:

- Uses software application to submit, control and monitor jobs
- Specify resources and script, software creates the cluster

```
> BatchJobsParam(workers = 4, resources = list(ncpus=1))

class: BatchJobsParam; bpisup: TRUE; bpworkers: 4; catch.errors: TRUE
cleanup: TRUE; stop.on.error: FALSE; progressbar: TRUE
```

Bioconductor AMI

What is it?

- Amazon Machine Image that runs in Elastic Compute Cloud (EC2)
- Comes pre-loaded with many *Bioconductor* software and annotation packages and their dependencies

Why use it?

- AMIs for different versions of *R*
- Add CPUs and/or memory in flexible configurations
- Run from any device, PC, tablet or other

Bioconductor AMI: Getting started

General steps:

- Create AWS account
- Create key pair
- Single machine: Launch AMI, choose resources, create stack
- Cluster: Install StarCluster, edit config file, start cluster
- Access AMI vis SSH or RStudio
- Full details: `http://www.bioconductor.org/help/bioconductor-cloud-ami/#overview`

Bioconductor AMI: Available resources

- Instances optimized for compute, memory, storage, GPU ...
- Instance details: `http://aws.amazon.com/ec2/instance-types/`
- Pricing details: `http://aws.amazon.com/ec2/pricing/`
- Free usage tier to encourage new users ...
  `http://aws.amazon.com/free/`
- Max resources currently available:
  - Up to 32 virtual cores per instance
  - Up to 244 GIG RAM per instance

Selecting and allocating resources

- Experiment with a small example to understand cpu (time) and memory (space) needs of your program.
- General rule of thumb is to specify 1 worker per core.
- May want to allocate more workers than cores if a process were limiting or irregular (i.e., data input). This would result in a pool of idle cores waiting for next available chunk of data vs stalling a pipeline.
  What usually happens is that workers are running at the same time and swapping in and out of the available processors as they compete for time. The overhead seen here is that swapping - sometimes referred to as 'over-subscribing'.

```
> workers <- c(bpworkers(), bpworkers() + 5, bpworkers() + 10)
> FUN <- function(i)
+   system.time(bplapply(1:1000, sqrt, BPPARAM = MulticoreParam(i)))
> do.call(rbind, lapply(workers, FUN))

     user.self sys.self elapsed user.child sys.child
[1,]     0.004    0.020   0.065      0.068     0.024
[2,]     0.020    0.020   0.074      0.156     0.072
[3,]     0.016    0.044   0.092      0.132     0.132
```

Exercise 1: AMI resources

In this exercise we investigate local resources associated with the c3.4xlarge AMI created for this lab. Full descriptions of the AWS instances can be found at `http://aws.amazon.com/ec2/instance-types/`.

a. Processors: Load *BiocParallel* package and check the number of workers with `bpworkers`.

b. Local storage: Check the amount of local storage on the instance by selecting *Shell* from the *Tools* drop-down menu. In the box type df -h. The *df* command stands for "disk filesystem" and the -h option shows disk space in human readable form (i.e., units along with raw numbers).

c. RAM: Again using the Shell, type 'free -m -h'.

d. Compare the statistics for the c3.4xlarge AMI to those of the conference AMI (m3.large).

Data storage options

EC2 instance store:

- Storage physically attached to instance
- Evaporates when instance is shut down

EBS:

- Persistent block-level storage for use with EC2 instances
- Network attached so some latency can be expected
- Volume is attached to instance in same availability zone

S3:

- Data storage service not a filesystem
- Transfer between S3 and EC2 is free.
- Best for storing items that must persist - push nightly backups, results etc.

Data storage options

No clear consensus regarding best I/O performance.

A Systematic Look at EC2 I/O

`http://blog.scalyr.com/2012/10/a-systematic-look-at-ec2-io/`
Take home was that performance varies widely across instances; offers a few general guidelines depending upon I/O situation.

Cloud Application Architechtures

`http://www.amazon.com/dp/0596156367/?tag=stackoverfl08-20`

Table: Comparison of EC2 data storage options

	S3	Instance	EBS
Speed	Low	Unpredictable	High
Reliability	Medium	High	High
Durability	Super high	Super low	High

Data access from and transfer to EC2

RStudio:

- *Import Dataset*: imports file into workspace
- *Upload files*: uploads file to local EC2 storage

SSH:

- Use wget, ftp, Dropbox to transfer files to local storage

Other command line tools:

- wget, web applications
- s3cmd (`http://s3tools.org/s3cmd`)
- aws cli (`http://aws.amazon.com/cli/`)
- *RAmazonS3* package

Exercise 2: S3 buckets

This exercise explores the data in Amazon S3 storage. We use the *RAmazonS3* package instead of a command line tool because it allows us to browse and download public S3 data without authorization credentials. Package details are available at `http://www.omegahat.org/RAmazonS3/`

a. Load the *RAmazonS3* package.

b. Look at the man page for `listBucket`. Get a list of items in the S3 1000 genomes bucket by calling `listBucket` with arguments `name = "1000genomes"`, `auth = NA`, and `maxKeys = 500`.

c. Retrieve the HG00096 chromosome 20 low coverage bam file. First construct a url composed of the key (file path) and bucket name.

```
> bucketname <- "1000genomes"
> key <- "data/HG00096/alignment/
+         HG00096.chrom20.ILLUMINA.bwa.GBR.low_coverage.20120522.bam"
> url <- paste0("http://s3.amazonaws.com/", bucketname, "/", key)
```

Upload the file to the EC2 local storage using the *Upload file* tab in RStudio or the `download.file` function.

Copy Number Analysis

The following exercises use data from a copy number analysis performed in "High-resolution mapping of copy-number alterations with massively parallel sequencing", `http://www.nature.com/nmeth/journal/v6/n1/abs/nmeth.1276.html`.

We use one of the three matched tumor normal pairs presented in the paper, the HCC1954 cell line from breast adenocarcinoma. Thanks to Sonali Arora who downloaded the raw data from the Short Read Archive, converted the files to fastq, checked quality and aligned the reads with bowtie2 to create the BAM files used here.

Exercise 3: Set-up

a. Load the *BiocParallel*, *Rsamtools*, *cn.mops*, and *DNAcopy* packages.
b. Locate the 4 BAM files (2 tumor, 2 normal) exp_srx036695.bam, exp_srx036696.bam, exp_srx036692.bam and exp_srx036697.bam.
c. Create a `BamFileList` from the BAM filenames.
d. Create a character vector `group` of "tumor" and "control" corresponding to the order of the files in the `BamFileList`.
e. Extract the `SeqInfo` calling `seqinfo` on on of the BamFiles.

Exercise 4: Shared memory vs non

Before we assess copy number let's look at the read coverage in a region of interest on chromosome 4.

a. Extract chromosome 4 from the `SeqInfo` object created in Exercise 3. Create a tiling of ranges over chromosome 4 with `tileGenome`; use the chromosome 4 `SeqInfo` as the `seqlengths` argument and set `tilewidth = 1e4`.

b. Create a `ScanBamParam` using the tiled ranges as the as the `which` argument. `which` requires a `GRanges` so the tiled ranges (`GRangesList`) need to be unlisted first.

c. Load the *GenomicAlignments* package.

d. Create a `MulticoreParam` and a `SnowParam` with enough workers to iterate over the BAM files.

Exercise 4 continued ...

e. Create two versions of a function, 'FUN1' and 'FUN2', that compute coverage on the BAM files in the regions specified by the `ScanBamParam`. 'FUN1' should take advantage of shared memory by
   - not passing (i.e., making a copy of) large objects defined in the workspace
   - not loading / initializing libraries currently loaded in the workspace.

   The man page for `coverage` on a BamFile is at

   ```
   ?`coverage,BamFile-method`
   ```

   (NOTE: you must type the quotes.)

f. Use `bplapply` to compute coverage on each file in the `BamFileList` with 'FUN1' and the `MultiCoreParam` as the `BPPARAM` argument to `bplapply`.

g. Repeat the coverage calculation with 'FUN2' and the `SnowParam`. We don't have a cluster available for testing; configuring the problem with the `SnowParam` simulates the non-shared memory environment of a cluster.

Exercise 5A: Nested evaluation (built-in)

Next we count reads with `getReadCountsFromBAM` and compute copy number with `referencecn.mops`. Read counting is an independent operation and can be done separately by file. Computing copy with `referencecn.mops` is not independent and requires that the case/control data be analyzed together. Many *R* / *Bioconductor* functions offer built in parallel evaluation. `getReadCountsFromBAM` can compute in parallel over the files. In this example we couple parallel evaluation over the chromosomes with the built-in parallel evaluation over the files.

a. 'FUN3' counts reads with `getReadCountsFromBAM` and computes copy number with `referencecn.mops`.

```
> FUN3 <- function(chrom, files, WL, group, ...) {
+   library(cn.mops)
+   counts <- getReadCountsFromBAM(files, WL = WL, mode = "unpaired",
+                                  refSeqName = chrom,
+                                  parallel = length(files))
+   referencecn.mops(cases = counts[,group == "tumor"],
+                    controls = counts[,group == "normal"])
+ }
```

Exercise 5A continued ...

b. Create a vector of chromosomes of interest, `chrom <- c("chr1", "chr4", "chr17")`.

c. Create a `SnowParam` with one worker per chromosome.

d. How many total workers are required (SnowParam + parallel argument)? Are there enough workers on the machine for the job?

e. Use `bplapply` to apply 'FUN3' over the `chrom` vector.

Exercise 5B: nested evaluation (roll your own)

Let's assume `getReadCountsFromBAM` does not have built-in parallel evaluation. In this case we can 'roll our own' second level of execution.

a. Modify 'FUN3' by wrapping `getReadCountsFromBAM` in a `bplapply` statement with a `MulticoreParam` as the `BPPARAM`. (Remove the `parallel` argument from `getReadCountsFromBAM`.

   Using a `SnowParam` to distribute the first level of parallel work followed by a `MulticoreParam` simulates how a job might be run on a particular cluster configuration. The first distribution passes data across cluster workers (could be individual nodes; no shared memory) while the second round of workers are spawned in a shared memory environment.

b. Call `bplapply` over `chrom` with the modified 'FUN3'.

Resources

- CRAN task view: `http://cran.r-project.org/web/views/HighPerformanceComputing.html`
- Scalable Genomic Computing and Visualization with *R* and *Bioconductor* (2014) `http://www.imstat.org/sts/future_papers.html`
- State of the Art Parallel Computing with *R* (2009) `http://www.jstatsoft.org/v31/i01/paper`
- Scalable Integrative Bioinformatics with Bioconductor `http://www.bioconductor.org/help/course-materials/2014/ISMB2014/`

Acknowledgements

- *Bioconductor* AMI author: Dan Tenenbaum
- *BiocParallel* authors: Martin Morgan, Michel Lang, and Ryan Thompson
- *Bioconductor* team: Marc Carlson, Hervé Pagès, Dan Tenenbaum, Sonali Arora, Nate Hayden, Paul Shannon, Martin Morgan
- Technical advisory council: Vincent Carey, Wolfgang Huber, Robert Gentleman, Rafael Irizzary, Sean Davis, Kasper Hansen, Michael Lawrence.
- Scientific advisory board: Simon Tavaré, Vivian Bonazzi, Vincent Carey, Wolfgang Huber, Robert Gentleman, Rafael Irizzary, Paul Flicek, Simon Urbanek.
- NIH / NHGRI U41HG0004059
- . . . and the *Bioconductor* community