Working with Large Data

Martin Morgan

June 23 – 28, 2013

\[^1\text{mtmorgan@fhcrc.org}\]
Write efficient R code

1. **Vectorize**
2. Pre-allocate and fill
3. Exploit existing software
4. Use appropriate algorithms
5. Avoid expensive conveniences

**Symptom** It takes too long to perform a simple mathematical calculation

**Solution** Vectorize, e.g., \( \log(x) \), \( x \times y \)
Write efficient R code

1. Vectorize
2. **Pre-allocate and fill**
3. Exploit existing software
4. Use appropriate algorithms
5. Avoid expensive conveniences

**Symptom** A loop appends to a result vector; it starts quickly but then gets slower and slower!

**Solution** Fill a pre-allocated result vector or, better, use `result <- lapply(x, function(...) ...) ...`).
Write efficient R code

1. Vectorize
2. Pre-allocate and fill
3. **Exploit existing software**
4. Use appropriate algorithms
5. Avoid expensive conveniences

**Symptom**  Everyone must be doing this, why do I feel like I’m re-inventing the wheel (and doing it poorly!)?

**Solution** Use established, performant software, e.g., *limma, DESeq2*
Write efficient R code

1. Vectorize
2. Pre-allocate and fill
3. Exploit existing software
4. **Use appropriate algorithms**
5. Avoid expensive conveniences

**Symptom** Execution time increases dramatically as data size increases

**Solution** Choose algorithms that scale linearly or better with data size.
Write efficient \textit{R} code

1. Vectorize
2. Pre-allocate and fill
3. Exploit existing software
4. Use appropriate algorithms
5. \textbf{Avoid expensive conveniences}

\textbf{Symptom} \hspace{1cm} \textbf{Solution}
An over-zealous ‘feature’ hurts performance
Avoid the feature, e.g.,
\begin{verbatim}
l = list(a=1:3);
unlist(l, use.names=FALSE)
\end{verbatim}
Write efficient *R* code

1. Vectorize
2. Pre-allocate and fill
3. Exploit existing software
4. Use appropriate algorithms
5. Avoid expensive conveniences

*There are many fast ways to get the wrong answer* – R. Gentleman
Manage data and processors

1. **Restriction**
2. **Sampling**
3. **Iteration**
4. **Parallel evaluation**

```r
library(Rsamtools)
gr <- GRanges("chr7",
              IRanges(100000, width=100))
param <- ScanBamParam(
    what = c("rname", "pos", "cigar"),
    which = gr)
scanBam("a.bam", param = param)
```
Manage data and processors

1. Restriction
2. **Sampling**
3. Iteration
4. Parallel evaluation

```r
library(ShortRead)
samp <- FastqSampler("end1.fastq")
yield(samp)
set.seed(123); yield(samp)
```
Manage data and processors

1. Restriction
2. Sampling
3. **Iteration**
4. Parallel evaluation

```r
library(Rsamtools)
bf <- BamFile("a.bam", yieldSize=1e6)
open(bf)
repeat {
  gl <- readGAlignments(bf)
  if (length(gl) == 0)
    break;
  ## do work
  ##
}
close(bf)
```
Manage data and processors

1. Restriction
2. Sampling
3. Iteration
4. **Parallel evaluation**

```r
library(parallel)
options(mc.cores=detectCores)
fls <- c("a.bam", "b.bam")

## sequential
x0 <- lapply(fls, countBam)

## parallel, 1 core per BAM file
x1 <- mclapply(fls, countBam)

identical(x0, x1) # TRUE

Other parallel solutions possible, e.g., clusters, Windows
```
Acknowledgments

- Vince Carey
- Michael Lawrence
- Hervé Pagès
- Valerie Obenchain
- Marc Carlson
- Paul Shannon
- Dan Tenenbaum