Genome-wide nucleosome positioning using NGS and HPC: from peaks to reads

SeqAhead Workshop on High Performance Computing for Next Generation Sequencing data analysis

May 22nd, 2012
Valencia
Why do we use HPC?
Why do we use HPC?

To perform more tasks in less time
Why do we use HPC?

To perform more tasks in less time

- Faster tasks (optimization)
- Multiple tasks (parallelism)
Why do we use HPC?

To perform more tasks in less time

Faster tasks (optimization)  Multiple tasks (parallelism)

Programming time:

Execution time/task:
Why do we use HPC?

To perform more tasks in less time

Faster tasks (optimization)  Multiple tasks (parallelism)

Programming time: 🕒 🕒 🕒
Execution time/task: 🕒
Why do we use HPC?

To perform more tasks in less time

Faster tasks (optimization)

Multiple tasks (parallelism)

Programming time:

Execution time/task:
Why do we use HPC?

To perform more tasks in less time

Faster tasks (optimization)

Multiple tasks (parallelism)

Programming time:  ⦿  ⦿  ⦿

Execution time/task:  🖥  🖥  🖥
Why do we use HPC?

To perform more tasks in less time

Faster tasks (optimization)

Multiple tasks (parallelism)

Programming time:  

Execution time/task:  

Implementation time vs Execution time
Why do we use HPC?

To perform more tasks in less time

- Faster tasks (optimization)
- Multiple tasks (parallelism)

Programming time:

Execution time/task:

Implementation time vs Execution time

- Portability
- Reusability
- Efficiency
- Interconnectivity
- Reliability (testing)
- Usability
- Maintainability
- ...
Which language do we use for HPC?
Which language do we use for HPC?

- execution time
+ programming time
Which language do we use for HPC?

- execution time
+ programming time

+/- execution time
+/- programming time
Which language do we use for HPC?

- execution time
+/- programming time

+/ execution time
- programming time

+/ execution time
- programming time
My story
My story
My story
My story
My story
My story
My story
My story
My story
My story
My story
Introducing R

- Language and environment for statistical computing
- Multiplatform and free (GNU)
- High-level interpreted language
  - Environment written in C, C++ and Fortran
- Array programming language
  - Forget about loops
Introducing R

- Language and environment for statistical computing
- Multiplatform and free (GNU)
- High-level interpreted language
  - Environment written in C, C++ and Fortran
- Array programming language
  - Forget about loops

```plaintext
for(int i=0; i<data.length; i++) { func(data[i]); }

apply(data, func)
```
“Bioconductor is an open source, open development software project to provide tools for the analysis and comprehension of high-throughput genomic data”

An extensive source of good-quality, well-documented, updated and community supported software for working with biological data.

Packages in release branch up to May-2012:
- 536 in software
- 624 in annotation
- 117 in experimental data
Chromatin and Nucleosomes

- Fundamental repeating unit on eukaryotic chromatin.
- They consist of 147bp wrapping around the histone octamers
- Separated by a short fragment (10-20bp) of linker DNA
- They allow 2 meters of DNA to be packed inside the cell nuclei of 10μm diameter
- About 75 – 90 % of DNA is occluded by nucleosomes. This is called the nucleosomal DNA.
Obtaining Nucleosomes

- Nucleosomal DNA is obtained by digesting the chromatin to the mononucleosomes with microccocal nuclease
Obtaining Nucleosomes

- Nucleosomal DNA is obtained by digesting the chromatin to the mononucleosomes with microccocal nuclease.
Obtaining Nucleosomes

- Nucleosomal DNA is obtained by digesting the chromatin to the mononucleosomes with microccocal nuclease.
Reads and Coverage
Efficient data structures (and operators) for common problems:

- Sequencing reads: ShortRead
- Sequences/Genomes: Biostrings
- Ranges: IRanges/GRanges
- Microarrays: ExpressionSet
R for bioinformatics (2)

- Efficient data structures (and operators) for common problems:
  - Sequencing reads: `ShortRead`
  - Sequences/Genomes: `Biostrings`
  - Ranges: `IRanges/GRanges`
  - Microarrays: `ExpressionSet`

SAM format (uncompressed BAM)

```
1:497:R:-272+13M17D24M 113 1 497 37 37M 15 100338662 0
CGGGTCTGACCTGAGGAGAACTGTGCTCCGCCTTCAG 0;==-
==9;>>>>>>>>>>>>>>>>>>XT:A:U NM:i:0 SM:i:37
AM:i:0 XO:i:1 X1:i:0 XM:i:0 XO:i:0 XG:i:0 MD:Z:37
19:20389:F:275+18M2D19M 99 1 17644 0 37M = 17919 314
TATGACTGCTAATAATACCTACACATGTTAGAACCAT
>>>>>>>>>>>>>>>>>>>><<>>><<>>4::>>:<9 RG:Z:UM0098:1 XT:A:R
NM:i:0 SM:i:0 AM:i:0 XO:i:4 X1:i:0 XM:i:0 XO:i:0 XG:i:0
MD:Z:37
19:20389:F:275+18M2D19M 147 1 17919 0 18M2D19M = 17644 -314
GTAGACTGCTAATAATACCTACACATGTTAGAACCAT
;44999;499<8<8<<<<<7;<<<<<< XT:A:R NM:i:2 SM:i:0
AM:i:0 XO:i:4 X1:i:0 XM:i:0 XO:i:1 XG:i:2 MD:Z:18^CA19
9:21597+10M2I25M:R:-209 83 1 21678 0 8M2I27M = 21469 -244
CACCACATCACA CATACACACATCGTGGCTGTCTCTCTCTCTCT
<;9<<<<<<<<<<<9<<<<<<9<<<<<< XT:A:R NM:i:2 SM:i:0
AM:i:0 XO:i:5 X1:i:0 XM:i:0 XO:i:1 XG:i:2 MD:Z:35
```
R for bioinformatics (2)

- Efficient data structures (and operators) for common problems:
  - Sequencing reads: ShortRead
  - Sequences/Genomes: Biostrings
  - Ranges: IRanges/GRanges
  - Microarrays: ExpressionSet

SAM format (uncompressed BAM)

```
1:497:R:-272+13M17D24M 113 1 497 37 37M 15 100338662 0
CGGGCTGACCTGAGGAACTGTGCTCCGCCTTCAG 0;==-=
AM:i:0 X0:i:1 X1:i:0 XM:i:0 XO:i:0 XG:i:0 MD:Z:37
19:20389:F:275+18M2D19M 99 1 17644 0 37M = 17919 314
TATGACTGCTAATAATACCTACACATGTTAGAACCAT
'>>>>>>>>>>>>>>>>'<<>>><<<<>>8::>>:<9 RG:Z:UM0098:1 XT:A:R
NM:i:0 SM:i:0 AM:i:0 X0:i:4 X1:i:0 XM:i:0 XO:i:0 XG:i:0
MD:Z:37
19:20389:F:275+18M2D19M 147 1 17919 0 18M2D19M = 17644 -314
CACCACATCACATATACCAAGCCTGGCTGTGTCTTCT
';9<<5<<<<<<<<<<<<<><9>>>>>> XT:A:R NM:i:2 SM:i:0
AM:i:0 X0:i:5 X1:i:0 XM:i:0 XO:i:1 XG:i:2 MD:Z:35
```

Ranged Data

```
RangedData with 4 rows and 1 value column across 25 spaces

<table>
<thead>
<tr>
<th>space</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;character&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;factor&gt;</td>
</tr>
<tr>
<td>1</td>
<td>chrX [17406833, 17406867]</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>chrX [17406844, 17406878]</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>chrX [17406844, 17406878]</td>
<td>+</td>
</tr>
<tr>
<td>4</td>
<td>chrX [17406844, 17406878]</td>
<td>-</td>
</tr>
</tbody>
</table>
```
Efficient data structures (and operators) for common problems:

- Sequencing reads: `ShortRead`
- Sequences/Genomes: `Biostrings`
- Ranges: `IRanges/GRanges`
- Microarrays: `ExpressionSet`

Coverage (numeric vector)

```
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 2 2 2 2 2 2 3 4
3 3 2 2 1 1 1 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0
1 2 1 1 1 1
```
Efficient data structures (and operators) for common problems:

- Sequencing reads: ShortRead
- Sequences/Genomes: Biostrings
- Ranges: IRanges/GRanges
- Microarrays: ExpressionSet

Coverage (numeric vector)

| 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 2 2 2 2 2 3 4 |
| 3 3 2 2 1 1 1 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 |
| 1 2 1 1 1 1 |

Coverage (Running Length Encoding)

'numeric' Rle of length 50 with 15 runs

- Lengths: 11 3 6 1 1 2 2 3 1 1 1 1 12 1 1 4
- Values : 0 1 2 3 4 3 2 1 0 1 2 0 1 2 1
Case of use: R for bioinformatics
Case of use: R for bioinformatics

```r
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])
```
#Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) #Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

SimpleRleList of length 16
$chrI
'numeric' Rle of length 230259 with 82632 runs
  Lengths: 3 2 ... 1
  Values: 14.7783718007107 15.0391665971938 ... 0.08693159882771

$chrII
'numeric' Rle of length 813223 with 306993 runs
  Lengths: 1043 50 ... 2
  Values: 0 0.08693159882771 ... 2.17328997069275
Case of use: R for bioinformatics

#Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) #Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

#Substract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")

cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

# Subtract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

SimpleRleList of length 16
$chrI
'numeric' Rle of length 230259 with 140647 runs
  Lengths:            3                  2 ...                     1
  Values:  11.0971720351962  11.3579668316793 ... -3.66372476603614

$chrII
'numeric' Rle of length 813223 with 556296 runs
  Lengths:                43                10 ...                      2
  Values:  0 -0.0694565993493306 ... 1.96492017264476
#Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) #Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

#Substract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

#Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
Case of use: R for bioinformatics

```r
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

# Substract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

# Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
```

```r
$chrI
IRanges of length 331
start   end width
[1] 579   608  30
[2] 682   684   3
[3] 735   777  43
[4] 1111 1121  11
... ... ... ...

$chrII
IRanges of length 969
start   end width
[1]  1 1043 1043
[2] 1094 1095  2
... ... ... ...
```
Case of use: R for bioinformatics

```r
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

# Subtract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

# Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
uncovered <- RangedData(IRangesList(uncovered))
```

```text
$chrI
IRanges of length 331
  start  end  width
[1]  579  608   30
[2]  682  684   3
[3]  735  777   43
[4] 1111 1121   11
...

$chrII
IRanges of length 969
  start  end  width
[1]   1 1043 1043
[2] 1094 1095   2
...
```
Case of use: R for bioinformatics

```r
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

# Subtract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

# Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
uncovered <- RangedData(IRangesList(uncovered))
```

---

RangedData with 14891 rows and 0 value columns across 16 spaces

<table>
<thead>
<tr>
<th>space</th>
<th>ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;character&gt;</td>
<td>&lt;IRanges&gt;</td>
</tr>
<tr>
<td>1</td>
<td>chrI [579, 608]</td>
</tr>
<tr>
<td>2</td>
<td>chrI [682, 684]</td>
</tr>
<tr>
<td>3</td>
<td>chrI [735, 777]</td>
</tr>
<tr>
<td>4</td>
<td>chrI [1111, 1121]</td>
</tr>
<tr>
<td>5</td>
<td>chrI [1205, 1240]</td>
</tr>
<tr>
<td>6</td>
<td>chrI [1412, 1425]</td>
</tr>
<tr>
<td>7</td>
<td>chrI [1655, 1922]</td>
</tr>
<tr>
<td>8</td>
<td>chrI [1973, 2034]</td>
</tr>
<tr>
<td>9</td>
<td>chrI [2095, 2192]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14883</td>
<td>chrXVI [937906, 937916]</td>
</tr>
</tbody>
</table>
# Case of use: R for bioinformatics

```r
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

# Subtract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

# Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
uncovered <- RangedData(IRangesList(uncovered))

# Get sequences and nucleotide frequencies of those regions
seq_uncov <- getSeq(BSgenome.Scerevisiae.UCSC.sacCer3, uncovered)
```

RangedData with 14891 rows and 0 value columns across 16 spaces

<table>
<thead>
<tr>
<th>space</th>
<th>ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;character&gt;</td>
<td>&lt;IRanges&gt;</td>
</tr>
<tr>
<td>1</td>
<td>chrI [579, 608]</td>
</tr>
<tr>
<td>2</td>
<td>chrI [682, 684]</td>
</tr>
<tr>
<td>3</td>
<td>chrI [735, 777]</td>
</tr>
<tr>
<td>4</td>
<td>chrI [1111, 1121]</td>
</tr>
<tr>
<td>5</td>
<td>chrI [1205, 1240]</td>
</tr>
<tr>
<td>6</td>
<td>chrI [1412, 1425]</td>
</tr>
<tr>
<td>7</td>
<td>chrI [1655, 1922]</td>
</tr>
<tr>
<td>8</td>
<td>chrI [1973, 2034]</td>
</tr>
<tr>
<td>9</td>
<td>chrI [2095, 2192]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14883</td>
<td>chrXVI [937906, 937916]</td>
</tr>
</tbody>
</table>
Case of use: R for bioinformatics

```r
#Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) #Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

#Substract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

#Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
uncovered <- RangedData(IRangesList(uncovered))

#Get sequences and nucleotide frequencies of those regions
seq_uncov <- getSeq(BSgenome.Scerevisiae.UCSC.sacCer3, uncovered)
```

A DNAStringSet instance of length 6

```
width  seq
[1] 30  TACCCATAACGCCCATCATTATCCACATTT
[2]  3    CAC
[3] 43  ATCCCCACAAAAATCACCTAAACATAAAAATATTCTACTTTTC
[4] 11  TAGTCAAGACG
[5] 36  TACTAAAACGGACGTTACGATATTGTCTCACTTCAT
[6] 14  ATCTGCAATCTTGT
```
# Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1]) # Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

# Subtract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

# Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
uncovered <- RangedData(IRangesList(uncovered))

# Get sequences and nucleotide frequencies of those regions
seq_uncov <- getSeq(BSgenome.Scerevisiae.UCSC.sacCer3, uncovered)
dinuc_tra <- oligonucleotideFrequency(seq_uncov, width=2, simplify.as="collapsed")
Case of use: R for bioinformatics

```r
#Import reads using ShortRead package
reads <- readAligned("/my_exp_dir/", "sample.bwt|control.bwt", type="bowtie")
cover_exp <- coverage(reads[1])  #Returns RleList, one entry per chromosome
cover_ctr <- coverage(reads[2])

#Substract control from sample, chromosome wise
cover_dif <- cover_exp - cover_ctr

#Get ranges of uncovered regions (not mapped?)
uncovered <- lapply(cover_dif, function(x) IRanges(x == 0))
uncovered <- RangedData(IRangesList(uncovered))

#Get sequences and nucleotide frequencies of those regions
seq_uncov <- getSeq(BSgenome.Scerevisiae.UCSC.sacCer3, uncovered)
dinuc_tra <- oligonucleotideFrequency(seq_uncov, width=2, simplify.as="collapsed")
```

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>170837</td>
<td>67886</td>
<td>70941</td>
</tr>
<tr>
<td>C</td>
<td>78151</td>
<td>43328</td>
<td>33238</td>
</tr>
<tr>
<td>G</td>
<td>75542</td>
<td>38350</td>
<td>40958</td>
</tr>
<tr>
<td>T</td>
<td>132986</td>
<td>76455</td>
<td>75909</td>
</tr>
</tbody>
</table>
HPC in R

Distributed memory clusters
- Many machines
  - Order of 100’s
- Few cores per host
  - 8-16 cores
- Few memory per host
  - Usually 1Gb x core

Shared memory
- Single machine
  - Independent
- Some cores in board
  - 32-64 cores
- Many memory
  - 1-2 Tb

Purpose-specific
- GPUs
- MD-machines
- ...
Explicit parallelism (OpenMPI, Rmpi) (native support from v.2.14)
Implicit parallelism (multicore, fork) (native support from v.2.14)
GPU parallelism (rgpu, OpenCL)
Grid, Hadoop (GridR, RHIPE)

Native code execution and performance profiling
- .C(), .Fortran(), Rcpp
- Rprof

http://cran.r-project.org/web/views/HighPerformanceComputing.html

(Google: “HPC in R”)

HPC in R (2)
Working with chromatin

- Number of reads for H.Sapiens K562 cell line of ENCODE project: **1850M**
  - BAM Files size (in disk, compressed): **90Gb**
  - Imported RangedData objects size (in memory, uncompressed): **20Gb**  (3.75Gb in disk)

- Number of positions in the coverage map: **3000M** (genome length)
  - Coverage map size (in memory, numeric vector): **23.1Gb**
  - Coverage map size (in memory, RleList - Run Length Encoding): **12.7Gb**
- Genome wide studies
- Memory intensive
- Usual tasks: ranges operations
  - Annotation, dataset crossing...
- Parallelization by chromosome/gene
Working with chromatin (2)

- Genome wide studies
- Memory intensive
- Usual tasks: ranges operations
  - Annotation, dataset crossing...
- Parallelization by chromosome/gene

Shared memory + Implicit parallelism (multithreading)
Working with chromatin (2)

- Genome wide studies
- Memory intensive
- Usual tasks: ranges operations
  - Annotation, dataset crossing...
- Parallelization by chromosome/gene

Shared memory + Implicit parallelism (multithreading)

```r
> mclapply(my_genes, my_func, cores=32)
```
Use case: nucleosome calling
Use case: nucleosome calling

- Previous methods for nucleosome calling
Use case: nucleosome calling

- Previous methods for nucleosome calling
  - Hidden Markov Models
Previous methods for nucleosome calling

- Hidden Markov Models
- Enrichment comparison (ChIP-Seq)
Use case: nucleosome calling

- Previous methods for nucleosome calling
  - Hidden Markov Models
  - Enrichment comparison (ChIP-Seq)
  - Other (complex) probabilistic methods
Criticism:

- HMM: Needs modeling (subjective), huge amount of memory needed (backtracking)
- Chip-Seq like methods: Based on enriched regions, not useful for nucleosomes
- Other (complex) probabilistic methods: Difficult to interpret, heavy computing
Use case: nucleosome calling

- Criticism:
  - HMM: Needs modeling (subjective), huge amount of memory needed (backtracking)
  - Chip-Seq like methods: Based on enriched regions, not useful for nucleosomes
  - Other (complex) probabilistic methods: Difficult to interpret, heavy computing

- All in all, the problem doesn’t seems so complicated...
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads
- But we can trim the ends of the read and keep only the middle
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads
- But we can trim the ends of the read and keep only the middle

Diagram:

**Reads trimming (paired ended)**

- + strand
- - strand
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads.
- But we can trim the ends of the read and keep only the middle.

![Reads trimming (paired ended)]
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads
- But we can trim the ends of the read and keep only the middle

Reads trimming (paired ended)

Reads centering (single ended)
Use case: nucleosome calling

- The peaks of a nucleosome coverage map can be detected as:
  - Global maxima (over a threshold)
The peaks of a nucleosome coverage map can be detected as:

- Global maxima (over a threshold)
- Local maxima
The peaks of a nucleosome coverage map can be detected as:

- Global maxima (over a threshold)
- Local maxima

Setting a **threshold** is a problem for global, **noise** is a problem for local
Use case: nucleosome calling
Use case: nucleosome calling

Fourier Space Transform
Describes the original input signal as a combination of single components, each one with different importance
Use case: nucleosome calling

**Fourier Space Transform**
Describes the original input signal as a combination of single components, each one with different importance.

**Component knock-out and inverse transform**
Frequencies with low power use to be echoes of higher ones or just random noise, so they can be removed without losing relevant information.
Use case: nucleosome calling

Fourier Space Transform
Describes the original input signal as a combination of single components, each one with different importance

Component knock-out and inverse transform
Frequencies with low power use to be echoes of higher ones or just random noise, so they can be removed without losing relevant information.
Use case: nucleosome calling

Fourier Space Transform
Describes the original input signal as a combination of single components, each one with different importance

Component knock-out and inverse transform
Frequencies with low power use to be echoes of higher ones or just random noise, so they can be removed without losing relevant information.
Use case: nucleosome calling

- Solution: standard signal theory -> noise filtering

**Fourier Space Transform**
Describes the original input signal as a combination of single components, each one with different importance

**Component knock-out and inverse transform**
Frequencies with low power use to be echoes of higher ones or just random noise, so they can be removed without losing relevant information.

- With 2% of components the signal has a $R^2 > 0.99$ with unfiltered coverage
Use case: nucleosome calling

**nucleR** (Flores & Orozco, 2011. *Bioinformatics*)

**nucleR** (local maxima)

**ChipSeqR** (global maxima)

**ChipSeqR** (local maxima)
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads
- But we can trim the ends of the read and keep only the middle
- Local maxima search over a filtered signal is trivial
  - \( \text{coverage}[i] > \text{coverage}[i+1] \)
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads
- But we can trim the ends of the read and keep only the middle
- Local maxima search over a filtered signal is trivial
  - coverage[i] > coverage[i+1]
Use case: nucleosome calling

- Peaks are not very easy to see with full-length reads
- But we can trim the ends of the read and keep only the middle
- Local maxima search over a filtered signal is trivial
  - $\text{coverage}[i] > \text{coverage}[i+1]$
  - $O(N)$
- Ok, detection is almost for free, but what about FFT filter?
(not so) Fast Fourier Transform
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}
```
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}

> mclapply(coverage, fftFilter, cores=16)
```
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}

> mclapply(coverage, fftFilter, cores=16)
```

about 4 days
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
    ...
    temp = fft(x)
    ...
    res = fft(temp, inverse=TRUE)
    ...
}

> mclapply(coverage, fftFilter, cores=16)
```

```r
> mclapply(coverage, fftFilter, cores=1)
```

about 4 days
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}
```

```r
> mclapply(coverage, fftFilter, cores=16)
```

about 4 days

```r
> mclapply(coverage, fftFilter, cores=1)
```

about 5 days
(not so) Fast Fourier Transform

fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}

> mclapply(coverage, fftFilter, cores=16)

about 4 days

> mclapply(coverage, fftFilter, cores=1)

about 5 days
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}
```

```r
> mclapply(coverage, fftFilter, cores=16)
```

about 4 days

```r
> mclapply(coverage, fftFilter, cores=1)
```

about 5 days
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}
```

```r
> mclapply(coverage, fftFilter, cores=16)
```

```r
> mclapply(coverage, fftFilter, cores=1)
```

In short:
- FFT depends on the number of components
- Efficiency depends on factorization
- Worst case: $O(N^2)$
- Best case: $O(N \log N)$
(not so) Fast Fourier Transform

```r
fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}

> mclapply(coverage, fftFilter, cores=16)

> mclapply(coverage, fftFilter, cores=1)
```

In short:

- FFT depends on the number of components
- Efficiency depends on factorization
- Worst case: $O(N^2)$
- Best case: $O(N \log N)$
In short:
- FFT depends on the number of components
- Efficiency depends on factorization
- Worst case: $O(N^2)$
- Best case: $O(N \log N)$
(not so) Fast Fourier Transform

fftFilter <- function(x)
{
  ...
  temp = fft(x)
  ...
  res = fft(temp, inverse=TRUE)
  ...
}

fftFilter <- function(x)
{
  //Enlarge the components vector
  //till nearest power of 2 (0-padding)
  x2 = pad(x, 0, 2^ceiling(ln2(length(x))))
  temp = lapply(x2, fft)
  ...
}

> mclapply(coverage, fftFilter, cores=16)
> mclapply(coverage, fftFilter, cores=1)

In short:
- FFT depends on the number of components
- Efficiency depends on factorization
- Worst case: $O(N^2)$
- Best case: $O(N \log N)$
Use case: Nucleosome Dynamics
Use case: Nucleosome Dynamics
Use case: Nucleosome Dynamics
Use case: Nucleosome Dynamics
Use case: Nucleosome Dynamics

Nucleosome Dynamics:

- Technique to infer the most probable movement at single nucleosome molecule level between two reference states

- Dynamics include the translational movement, eviction or reclusion of nucleosomes

- Automatic detection of hotspots:
  - Shifts \[\rightarrow\rightarrow\rightarrow\]
  - Dispersions \[\leftarrow\leftarrow\rightarrow\rightarrow\]
  - Concentrations \[\rightarrow\rightarrow\leftarrow\leftarrow\]
Use case: Nucleosome Dynamics

- Possible combinations per window: $10^{2000}$ (a few billions up or down)
  - Windows of 600bp, about 500-1000 reads per window, shifts < 50bp

- An analytical answer is impossible

- Optimization problem solved with a genetic algorithm
Use case: Nucleosome Dynamics

- Possible combinations per window: $10^{2000}$ (a few billions up or down)
  - Windows of 600bp, about 500-1000 reads per window, shifts < 50bp

- An analytical answer is impossible

- Optimization problem solved with a genetic algorithm

Diagram:

1. Initial random population
2. Evaluation (fitting)
3. Best solutions found
4. End?
5. Crossover (mating)
6. Random changes (mutations)
7. New random solutions
Use case: Nucleosome Dynamics

- Possible combinations per window: $10^{2000}$ (a few billions up or down)
  - Windows of 600bp, about 500-1000 reads per window, shifts < 50bp

- An analytical answer is impossible

- Optimization problem solved with a genetic algorithm

O(N=5000) → O(N^2) → O(N) → end? → O(< N) → O(< N) → random changes (mutations) → New random solutions → O(< N)

O(N=5000)
Use case: Nucleosome Dynamics

- Possible combinations per window: $10^{2000}$ (a few billions up or down)
  - Windows of 600bp, about 500-1000 reads per window, shifts < 50bp

- An analytical answer is impossible

- Optimization problem solved with a genetic algorithm

\[ O(N=5000) \] \[ O(N^2) \] \[ O(N) \] \[ O(< N) \] \[ O(< N) \]
Use case: Nucleosome Dynamics

<table>
<thead>
<tr>
<th>% total</th>
<th>total seconds</th>
<th>% self</th>
<th>self seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>23161.90</td>
<td>0.0</td>
<td>1.90</td>
<td>&quot;nucleosomeDynamics&quot;</td>
</tr>
<tr>
<td>100.0</td>
<td>23159.10</td>
<td>0.1</td>
<td>14.10</td>
<td>&quot;FUN&quot;</td>
</tr>
<tr>
<td>100.0</td>
<td>23150.36</td>
<td>0.0</td>
<td>9.20</td>
<td>&quot;lapply&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23144.24</td>
<td>1.3</td>
<td>301.82</td>
<td>&quot;standardGeneric&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23135.80</td>
<td>0.5</td>
<td>111.60</td>
<td>&quot;unlist&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23133.34</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;mclapply&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23133.32</td>
<td>0.1</td>
<td>12.38</td>
<td>&quot;.fitScore&quot;</td>
</tr>
<tr>
<td>89.3</td>
<td>20675.18</td>
<td>0.1</td>
<td>15.32</td>
<td>&quot;.local&quot;</td>
</tr>
<tr>
<td>86.7</td>
<td>20087.60</td>
<td>0.0</td>
<td>1.34</td>
<td>&quot;.findOverlaps&quot;</td>
</tr>
<tr>
<td>71.9</td>
<td>16643.40</td>
<td>71.8</td>
<td>16638.10</td>
<td>&quot;.Call&quot;</td>
</tr>
<tr>
<td>69.6</td>
<td>16123.32</td>
<td>0.0</td>
<td>0.80</td>
<td>&quot;.IntervalTreeCall&quot;</td>
</tr>
<tr>
<td>15.2</td>
<td>3509.22</td>
<td>13.8</td>
<td>3207.78</td>
<td>&quot;processSelfMatching&quot;</td>
</tr>
<tr>
<td>7.4</td>
<td>1719.12</td>
<td>0.0</td>
<td>8.10</td>
<td>&quot;IRanges&quot;</td>
</tr>
</tbody>
</table>
Use case: Nucleosome Dynamics

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td>seconds</td>
<td>name</td>
</tr>
<tr>
<td>100.0</td>
<td>23161.90</td>
<td>0.0</td>
<td>1.90</td>
<td>&quot;nucleosomeDynamics&quot;</td>
</tr>
<tr>
<td>100.0</td>
<td>23159.10</td>
<td>0.1</td>
<td>14.10</td>
<td>&quot;FUN&quot;</td>
</tr>
<tr>
<td>100.0</td>
<td>23150.36</td>
<td>0.0</td>
<td>9.20</td>
<td>&quot;lapply&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23144.24</td>
<td>1.3</td>
<td>301.82</td>
<td>&quot;standardGeneric&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23135.80</td>
<td>0.5</td>
<td>111.60</td>
<td>&quot;unlist&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23133.34</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;mclapply&quot;</td>
</tr>
<tr>
<td>99.9</td>
<td>23133.32</td>
<td>0.1</td>
<td>12.38</td>
<td>&quot;.fitScore&quot;</td>
</tr>
<tr>
<td>89.3</td>
<td>20675.18</td>
<td>0.1</td>
<td>15.32</td>
<td>&quot;.local&quot;</td>
</tr>
<tr>
<td>86.7</td>
<td>20087.60</td>
<td>0.0</td>
<td>1.34</td>
<td>&quot;.Call&quot;</td>
</tr>
<tr>
<td>71.9</td>
<td>16643.40</td>
<td>71.8</td>
<td>16638.10</td>
<td>&quot;.IntervalTreeCall&quot;</td>
</tr>
<tr>
<td>69.6</td>
<td>16123.32</td>
<td>0.0</td>
<td>0.80</td>
<td>&quot;processSelfMatching&quot;</td>
</tr>
<tr>
<td>15.2</td>
<td>3509.22</td>
<td>13.8</td>
<td>3207.78</td>
<td>&quot;IRanges&quot;</td>
</tr>
</tbody>
</table>
Use case: Nucleosome Dynamics

- **Distribution of execution time:**
  - 10 iterations of the GA, N=5000, default parameters

- **GA functions** (selection, mating, mutation, ...): 0.01% (50 sec.)
  - Most of the code, more than 300 lines of R code
  - High level functions (`sample`, `apply`, `sort`, subsetting...).
    
    Example: `sel0 = sample(which(tmp==0), size=selsize)`

- **Evaluation** (fitting function): 99.9% (6 hours)
  - 10 lines of R code
  - Only a couple of calls and some basic operations
Use case: Nucleosome Dynamics

Steps in the fitting function (given GA element):

1. Calculate modified reads:
   
   ```r
   new_reads <- IRanges(starts+shifts)
   ```
Use case: Nucleosome Dynamics

- Steps in the fitting function (given GA element):
  1. Calculate modified reads:
     
     ```r
     new_reads <- IRanges(starts+shifts)
     ```
  2. Count overlapping shifts:
     
     ```r
     findOverlaps(new_reads)
     ```
Use case: Nucleosome Dynamics

Steps in the fitting function (given GA element):

1. Calculate modified reads:
   
   ```r
   new_reads <- IRanges(starts+shifts)
   ```

2. Count overlapping shifts:
   
   ```r
   findOverlaps(new_reads)
   ```

3. Calculate the new coverage with proposed changes
   
   ```r
   coverage(new_reads)
   ```
Use case: Nucleosome Dynamics

Steps in the fitting function (given GA element):

1. Calculate modified reads:
   
   ```r
   new_reads <- IRanges(starts+shifts)
   ```

2. Count overlapping shifts:
   
   ```r
   findOverlaps(new_reads)
   ```

3. Calculate the new coverage with proposed changes
   
   ```r
   coverage(new_reads)
   ```

4. Compare the coverages, sum scores and blah blah
Use case: Nucleosome Dynamics

- Count overlapping shifts (100 executions):
  ```
  findOverlaps(new_reads)
  ```
Use case: Nucleosome Dynamics

- Count overlapping shifts (100 executions):

  \[\text{findOverlaps}(\text{new}\_\text{reads})\]

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td>seconds</td>
<td>name</td>
</tr>
<tr>
<td>100.0</td>
<td>15.58</td>
<td>0.9</td>
<td>0.14</td>
<td></td>
<td>&quot;standardGeneric&quot;</td>
</tr>
<tr>
<td>100.0</td>
<td>15.58</td>
<td>0.0</td>
<td>0.00</td>
<td></td>
<td>&quot;findOverlaps&quot;</td>
</tr>
<tr>
<td>93.6</td>
<td>14.58</td>
<td>0.0</td>
<td>0.00</td>
<td></td>
<td>&quot;.local&quot;</td>
</tr>
<tr>
<td>77.2</td>
<td>12.02</td>
<td>77.2</td>
<td>12.02</td>
<td></td>
<td>&quot;.Call&quot;</td>
</tr>
<tr>
<td>75.3</td>
<td>11.74</td>
<td>0.1</td>
<td>0.02</td>
<td></td>
<td>&quot;.IntervalTreeCall&quot;</td>
</tr>
<tr>
<td>13.7</td>
<td>2.14</td>
<td>12.2</td>
<td>1.90</td>
<td></td>
<td>&quot;processSelfMatching&quot;</td>
</tr>
</tbody>
</table>

...
Use case: Nucleosome Dynamics

- Count overlapping shifts (100 executions):
  
  ```
  findOverlaps(new_reads)
  ```

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>self</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>total</td>
<td>%</td>
<td>self</td>
<td>self</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>15.58</td>
<td>0.9</td>
<td>0.14</td>
<td>&quot;standardGeneric&quot;</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>15.58</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;findOverlaps&quot;</td>
<td></td>
</tr>
<tr>
<td>93.6</td>
<td>14.58</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;.local&quot;</td>
<td></td>
</tr>
<tr>
<td>77.2</td>
<td>12.02</td>
<td>77.2</td>
<td>12.02</td>
<td>&quot;.Call&quot;</td>
<td></td>
</tr>
<tr>
<td>75.3</td>
<td>11.74</td>
<td>0.1</td>
<td>0.02</td>
<td>&quot;.IntervalTreeCall&quot;</td>
<td></td>
</tr>
<tr>
<td>13.7</td>
<td>2.14</td>
<td>12.2</td>
<td>1.90</td>
<td>&quot;processSelfMatching&quot;</td>
<td></td>
</tr>
</tbody>
</table>

```c
void covlp(int *out, int *p, int *s, int *len)
{
    int i, j, l = *len;
    out[0] = 0;
    for(i=0; i<l-1; i++)
    {
        for(j=i+1; j<l; j++)
        {
            if(s[i]!=0 & s[j]!=0 & p[i] < p[j] & (p[i]+s[i]) > (p[j]+s[j])) out[0]++;
        }
    }
}
```
Use case: Nucleosome Dynamics

- Count overlapping shifts (100 executions):
  
  ```r
  findOverlaps(new_reads)
  ```

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td>seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>15.58</td>
<td>0.9</td>
<td>0.14</td>
<td></td>
<td>&quot;standardGeneric&quot;</td>
</tr>
<tr>
<td>100.0</td>
<td>15.58</td>
<td>0.0</td>
<td>0.00</td>
<td></td>
<td>&quot;findOverlaps&quot;</td>
</tr>
<tr>
<td>93.6</td>
<td>14.58</td>
<td>0.0</td>
<td>0.00</td>
<td></td>
<td>&quot;.local&quot;</td>
</tr>
<tr>
<td>77.2</td>
<td>12.02</td>
<td>77.2</td>
<td>12.02</td>
<td></td>
<td>&quot;.Call&quot;</td>
</tr>
<tr>
<td>75.3</td>
<td>11.74</td>
<td>0.1</td>
<td>0.02</td>
<td></td>
<td>&quot;.IntervalTreeCall&quot;</td>
</tr>
<tr>
<td>13.7</td>
<td>2.14</td>
<td>12.2</td>
<td>1.90</td>
<td></td>
<td>&quot;processSelfMatching&quot;</td>
</tr>
</tbody>
</table>

...
Use case: Nucleosome Dynamics

- Calculate the coverage (10000 executions)

  \texttt{coverage(new\_reads)}
Use case: Nucleosome Dynamics

- Calculate the coverage (10000 executions)

```
coverage(new_reads)
```

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>%</th>
<th>self</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>total</td>
<td>seconds</td>
<td></td>
<td>%</td>
<td>self</td>
<td>seconds</td>
</tr>
<tr>
<td>100.0</td>
<td>39.12</td>
<td>0.2</td>
<td>0.08</td>
<td>&quot;coverage&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.8</td>
<td>39.06</td>
<td>20.8</td>
<td>8.12</td>
<td>&quot;standardGeneric&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81.0</td>
<td>31.68</td>
<td>0.7</td>
<td>0.28</td>
<td>&quot;.local&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40.5</td>
<td>15.86</td>
<td>0.1</td>
<td>0.06</td>
<td>&quot;restrict&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39.3</td>
<td>15.36</td>
<td>6.8</td>
<td>2.68</td>
<td>&quot;Ranges.restrict&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36.0</td>
<td>14.10</td>
<td>2.2</td>
<td>0.86</td>
<td>&quot;unsafe.update“</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>&quot;...&quot;</td>
</tr>
<tr>
<td>8.5</td>
<td>3.34</td>
<td>8.3</td>
<td>3.24</td>
<td>&quot;.Call&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Use case: Nucleosome Dynamics

- Calculate the coverage (10000 executions)

**coverage(new_reads)**

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>self</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td></td>
<td>self</td>
<td></td>
<td></td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>100.0</td>
<td>39.12</td>
<td>0.2</td>
<td>0.08</td>
<td>0.08</td>
<td>&quot;coverage&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.8</td>
<td>39.06</td>
<td>20.8</td>
<td>8.12</td>
<td>8.12</td>
<td>&quot;standardGeneric&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81.0</td>
<td>31.68</td>
<td>0.7</td>
<td>0.28</td>
<td>0.28</td>
<td>&quot;.local&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40.5</td>
<td>15.86</td>
<td>0.1</td>
<td>0.06</td>
<td>0.06</td>
<td>&quot;restrict&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39.3</td>
<td>15.36</td>
<td>6.8</td>
<td>2.68</td>
<td>2.68</td>
<td>&quot;Ranges.restrict&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36.0</td>
<td>14.10</td>
<td>2.2</td>
<td>0.86</td>
<td>0.86</td>
<td>&quot;unsafe.update&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;.Call&quot;</td>
</tr>
<tr>
<td>8.5</td>
<td>3.34</td>
<td>8.3</td>
<td>3.24</td>
<td>3.24</td>
<td></td>
<td></td>
<td>&quot;.Call&quot;</td>
</tr>
</tbody>
</table>

```c
void ccov(int *out, int *sta, int *len_sta, int* width)
{
    int i, j, w = *width, l = *len_sta;
    for(i=0; i<l; i++) for(j=0; j<w; j++) out[sta[i]+j]++;
}
```
Use case: Nucleosome Dynamics

- Calculate the coverage (10000 executions)

```r
coverage(new_reads)
```

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td></td>
<td></td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>100.0</td>
<td>39.12</td>
<td>0.2</td>
<td>0.08</td>
<td></td>
<td></td>
<td>&quot;coverage&quot;</td>
</tr>
<tr>
<td>99.8</td>
<td>39.06</td>
<td>20.8</td>
<td>8.12</td>
<td></td>
<td></td>
<td>&quot;standardGeneric&quot;</td>
</tr>
<tr>
<td>81.0</td>
<td>31.68</td>
<td>0.7</td>
<td>0.28</td>
<td></td>
<td></td>
<td>&quot;.local&quot;</td>
</tr>
<tr>
<td>40.5</td>
<td>15.86</td>
<td>0.1</td>
<td>0.06</td>
<td></td>
<td></td>
<td>&quot;restrict&quot;</td>
</tr>
<tr>
<td>39.3</td>
<td>15.36</td>
<td>6.8</td>
<td>2.68</td>
<td></td>
<td></td>
<td>&quot;Ranges.restrict&quot;</td>
</tr>
<tr>
<td>36.0</td>
<td>14.10</td>
<td>2.2</td>
<td>0.86</td>
<td></td>
<td></td>
<td>&quot;unsafe.update&quot;</td>
</tr>
</tbody>
</table>

...  
8.5  3.34  8.3  3.24  ".Call"

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td></td>
<td></td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>100.0</td>
<td>2.96</td>
<td>2.7</td>
<td>0.08</td>
<td></td>
<td></td>
<td>&quot;.ccov&quot;</td>
</tr>
<tr>
<td>85.8</td>
<td>2.54</td>
<td>85.8</td>
<td>2.54</td>
<td></td>
<td></td>
<td>&quot;.C&quot;</td>
</tr>
<tr>
<td>10.8</td>
<td>0.32</td>
<td>10.8</td>
<td>0.32</td>
<td></td>
<td></td>
<td>&quot;as.integer&quot;</td>
</tr>
<tr>
<td>0.7</td>
<td>0.02</td>
<td>0.7</td>
<td>0.02</td>
<td></td>
<td></td>
<td>&quot;vector&quot;</td>
</tr>
<tr>
<td>0.7</td>
<td>0.02</td>
<td>0.0</td>
<td>0.00</td>
<td></td>
<td></td>
<td>&quot;integer&quot;</td>
</tr>
</tbody>
</table>
Use case: Nucleosome Dynamics
Use case: Nucleosome Dynamics

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>143.54</td>
<td>1.5</td>
<td>2.10</td>
<td></td>
<td>&quot;nucleosomeDynamics&quot;</td>
</tr>
<tr>
<td>98.2</td>
<td>141.02</td>
<td>3.9</td>
<td>5.60</td>
<td></td>
<td>&quot;FUN&quot;</td>
</tr>
<tr>
<td>92.9</td>
<td>133.38</td>
<td>0.3</td>
<td>0.38</td>
<td></td>
<td>&quot;lapply&quot;</td>
</tr>
<tr>
<td>88.8</td>
<td>127.44</td>
<td>1.9</td>
<td>2.80</td>
<td></td>
<td>&quot;standardGeneric&quot;</td>
</tr>
<tr>
<td>84.0</td>
<td>120.62</td>
<td>0.1</td>
<td>0.08</td>
<td></td>
<td>&quot;unlist&quot;</td>
</tr>
<tr>
<td>82.3</td>
<td>118.10</td>
<td>0.0</td>
<td>0.00</td>
<td></td>
<td>&quot;mclapply&quot;</td>
</tr>
<tr>
<td>82.2</td>
<td>118.00</td>
<td>4.2</td>
<td>6.00</td>
<td></td>
<td>&quot;.fitScore&quot;</td>
</tr>
<tr>
<td>65.7</td>
<td>94.26</td>
<td>64.7</td>
<td>92.90</td>
<td></td>
<td>&quot;.C&quot;</td>
</tr>
<tr>
<td>45.3</td>
<td>65.02</td>
<td>0.2</td>
<td>0.36</td>
<td></td>
<td>&quot;.covlp&quot;</td>
</tr>
<tr>
<td>24.2</td>
<td>34.80</td>
<td>0.3</td>
<td>0.44</td>
<td></td>
<td>&quot;.ccov&quot;</td>
</tr>
<tr>
<td>6.7</td>
<td>9.68</td>
<td>0.1</td>
<td>0.22</td>
<td></td>
<td>&quot;apply&quot;</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>self</td>
<td></td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------</td>
<td>----------</td>
<td>----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>total</td>
<td>total</td>
<td>self</td>
<td>total</td>
<td>self</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>143.54</td>
<td>1.5</td>
<td>2.10</td>
<td>&quot;nucleosomeDynamics&quot;</td>
<td></td>
</tr>
<tr>
<td>98.2</td>
<td>141.02</td>
<td>3.9</td>
<td>5.60</td>
<td>&quot;FUN&quot;</td>
<td></td>
</tr>
<tr>
<td>92.9</td>
<td>133.38</td>
<td>0.3</td>
<td>0.38</td>
<td>&quot;lapply&quot;</td>
<td></td>
</tr>
<tr>
<td>88.8</td>
<td>127.44</td>
<td>1.9</td>
<td>2.80</td>
<td>&quot;standardGeneric&quot;</td>
<td></td>
</tr>
<tr>
<td>84.0</td>
<td>120.62</td>
<td>0.1</td>
<td>0.08</td>
<td>&quot;unlist&quot;</td>
<td></td>
</tr>
<tr>
<td>82.3</td>
<td>118.10</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;mclapply&quot;</td>
<td></td>
</tr>
<tr>
<td>82.2</td>
<td>118.00</td>
<td>4.2</td>
<td>6.00</td>
<td>&quot;.fitScore&quot;</td>
<td></td>
</tr>
<tr>
<td>65.7</td>
<td>94.26</td>
<td>64.7</td>
<td>92.90</td>
<td>&quot;.C&quot;</td>
<td></td>
</tr>
<tr>
<td>45.3</td>
<td>65.02</td>
<td>0.2</td>
<td>0.36</td>
<td>&quot;.covlp&quot;</td>
<td></td>
</tr>
<tr>
<td>24.2</td>
<td>34.80</td>
<td>0.3</td>
<td>0.44</td>
<td>&quot;.ccov&quot;</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>9.68</td>
<td>0.1</td>
<td>0.22</td>
<td>&quot;apply&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Use case: Nucleosome Dynamics
Use case: Nucleosome Dynamics

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td>seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>143.54</td>
<td>1.5</td>
<td>2.10</td>
<td>&quot;nucleosomeDynamics&quot;</td>
<td></td>
</tr>
<tr>
<td>98.2</td>
<td>141.02</td>
<td>3.9</td>
<td>5.60</td>
<td>&quot;FUN&quot;</td>
<td></td>
</tr>
<tr>
<td>92.9</td>
<td>133.38</td>
<td>0.3</td>
<td>0.38</td>
<td>&quot;lapply&quot;</td>
<td></td>
</tr>
<tr>
<td>88.8</td>
<td>127.44</td>
<td>1.9</td>
<td>2.80</td>
<td>&quot;standardGeneric&quot;</td>
<td></td>
</tr>
<tr>
<td>84.1</td>
<td>120.62</td>
<td>0.1</td>
<td>0.08</td>
<td>&quot;unlist&quot;</td>
<td></td>
</tr>
<tr>
<td>82.2</td>
<td>118.10</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;mclapply&quot;</td>
<td></td>
</tr>
<tr>
<td>82.2</td>
<td>118.00</td>
<td>4.2</td>
<td>6.00</td>
<td>&quot;.fitScore&quot;</td>
<td></td>
</tr>
<tr>
<td>65.7</td>
<td>94.26</td>
<td>64.7</td>
<td>92.90</td>
<td>&quot;.C&quot;</td>
<td></td>
</tr>
<tr>
<td>45.3</td>
<td>65.02</td>
<td>0.2</td>
<td>0.36</td>
<td>&quot;.covlp&quot;</td>
<td></td>
</tr>
<tr>
<td>24.2</td>
<td>34.80</td>
<td>0.3</td>
<td>0.44</td>
<td>&quot;.ccov&quot;</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>9.68</td>
<td>0.1</td>
<td>0.22</td>
<td>&quot;apply&quot;</td>
<td></td>
</tr>
</tbody>
</table>

0.03 hours with 1 core
(x200 times faster)
## Use case: Nucleosome Dynamics

A use case for nucleosome dynamics analysis is presented, demonstrating performance metrics for various computational tasks.

<table>
<thead>
<tr>
<th>%</th>
<th>total</th>
<th>%</th>
<th>self</th>
<th>seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>seconds</td>
<td>self</td>
<td>seconds</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>143.54</td>
<td>1.5</td>
<td>2.10</td>
<td>&quot;nucleosomeDynamics&quot;</td>
<td></td>
</tr>
<tr>
<td>98.2</td>
<td>141.02</td>
<td>3.9</td>
<td>5.60</td>
<td>&quot;FUN&quot;</td>
<td></td>
</tr>
<tr>
<td>92.9</td>
<td>133.38</td>
<td>0.3</td>
<td>0.38</td>
<td>&quot;lapply&quot;</td>
<td></td>
</tr>
<tr>
<td>88.8</td>
<td>127.44</td>
<td>1.9</td>
<td>2.80</td>
<td>&quot;standardGeneric&quot;</td>
<td></td>
</tr>
<tr>
<td>84.0</td>
<td>120.62</td>
<td>0.1</td>
<td>0.08</td>
<td>&quot;unlist&quot;</td>
<td></td>
</tr>
<tr>
<td>82.3</td>
<td>118.10</td>
<td>0.0</td>
<td>0.00</td>
<td>&quot;mclapply&quot;</td>
<td></td>
</tr>
<tr>
<td>82.2</td>
<td>118.00</td>
<td>4.2</td>
<td>6.00</td>
<td>&quot;.fitScore&quot;</td>
<td></td>
</tr>
<tr>
<td>65.7</td>
<td>94.26</td>
<td>64.7</td>
<td>92.90</td>
<td>&quot;.C&quot;</td>
<td></td>
</tr>
<tr>
<td>45.3</td>
<td>65.02</td>
<td>0.2</td>
<td>0.36</td>
<td>&quot;.covlp&quot;</td>
<td></td>
</tr>
<tr>
<td>24.2</td>
<td>34.80</td>
<td>0.3</td>
<td>0.44</td>
<td>&quot;.ccov&quot;</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>9.68</td>
<td>0.1</td>
<td>0.22</td>
<td>&quot;apply&quot;</td>
<td></td>
</tr>
</tbody>
</table>

*0.03 hours with 1 core (x200 times faster)*
Take home messages

- 80/20 law: 80% of the execution time is spent on 20% of the code
  - Do a fast prototype with high-level code
  - Profile
  - Improve the performance at the bottlenecks with low level code

- “More cores” is not a synonym of “faster tasks”

- HPC aim: performing more tasks in less time
  - But not only “execution” time!

- R *can be* a good language for HPC
Acknowledgments:

Özgen Deniz
Modesto Orozco
IRB Biostatistics unit:
    David Rosell
    Evarist Planet

Thank you for your attention!

and all other MMB - EBL members: