Running Gotcha with parallel computing in slurm

Code

This is a tutorial for processing GoT-ChA genotyping libraries. The Gotcha R package required for this tutorial can be downloaded here.

Use of BatchMutationCalling function

First, we load the Gotcha R library:

Hide

```
library(Gotcha)
```

```
Checking if r-reticulate-gotcha virtual environment is available...
Virutal environment r-reticulate-gotcha is available | use_virtualenv(r-reticulate-gotcha)
```

To define the path to the fastq files:

Hide

```
# Path to the folder where the .fastqs are:
path_to_fastq = "/gpfs/commons/home/fizzo/GoTChA/raw_data/Tutorial/"

# Path where to store split and filtered .fastq files:
path_out = "/gpfs/commons/home/fizzo/GoTChA/raw_data/Tutorial/outs/"
```

First, we need to split the fastq files in chunks to allow for parallel processing. This can be done by running:

Hide

```
FastqSplit(path = path_to_fastq,
           out = path_out,
           reads = 10000000,
           ncores = 12)
```

```
------- CREATING FILE INDEX -------
------- CREATING OUTPUT FOLDER -------
------- SPLITTING FASTQS -------
------- DONE! -------
```

After splitting the fastq files, a folder for each chunk will be created in the specified path. Next, we have to filter out those reads that contain low quality scores, particularly at the mutation site of interest:

Hide

```
FastqFiltering(out = path_out,
               min.quality = 15,
               min.bases = 1,
               which.read = "R1",
               read.region = c(31:34),
               ncores = 12)
```

```
------- BEGIN FASTQ FILTERING FUNCTION -------
------- FASTQ FILES IDENTIFIED -------
------- PER READ INDEX CREATED -------
------- RAW FASTQ FILES LOADED -------
------- FILTERING COMPLETE -------
------- FASTQ FILTERING METRICS -------
------- mean number of filtered reads = 9321577.14285714
------- % of remaining reads after filtering= 93.216 +/- 10.709 (mean +/- sd)
------- FASTQ FILTERING FUNCTION COMPLETE -------
```

Since we are running Gotcha with parallel computing in slurm, we can use the *BatchMutationCalling* function and submit one cluster job per fastq chunk:

Hide

```
BatchMutationCalling(out = path_out,
                     whitelist.file.path = "/gpfs/commons/home/fizzo/GoTChA/Whitelist/737K-cratac-v1.txt",
                     wt.max.mismatch = 0,
                     mut.max.mismatch = 0,
                     keep.raw.reads = F,
                     reverse.complement = T,
                     testing = F,
                     which.read = "R1",
                     primer.sequence = "GTGTAACAGTTCCTGCATGGGCGGCATGAAC",
                     primed.max.mismatch = 3,
                     atac.barcodes = F,
                     atac.barcodes.file.path = NA,
                     wt.sequence = "CGG",
                     mut.sequence= "CAG",
                     mutation.start = 31,
                     mutation.end = 34,
                     ncores = 12,
                     soptions = list(output ='%x_%j.log', mem = '40g', 'cpus-per-task' = 12)
)
```

```
------- GENERATING CHUNK INDEX -------
------- GENERATING PARAMETERS -------
------- SUBMITTING SLURM JOBS TO CLUSTER -------
Submitted batch job 26844369
Submitted batch job 26844370
Submitted batch job 26844371
Submitted batch job 26844372
Submitted batch job 26844373
Submitted batch job 26844374
Submitted batch job 26844375
------- DONE! -------
The working directory was changed to /gpfs/commons/home/fizzo/GoTChA/raw_data/Tutorial/outs/Split/Filtered inside
a notebook chunk. The working directory will be reset when the chunk is finished running. Use the knitr root.dir
option in the setup chunk to change the working directory for notebook chunks.
```

This will submit a job for each fastq chunk. Once the submitted jobs are completed, we can merge the outputs of each BatchMutationCalling job into one single data frame using the *MergeMutationOuts* function. This will generate a new folder containing a .Rdata class object that can be directly loaded into R:

Hide

```
MergeMutationCalling(out = path_out)
```

```
------- LOADING MUTATION CALLING OUTPUTS -------
------- MERGING MUTATION CALLING OUTPUTS -------
------- COLLAPSE BARCODE METRICS -------
------- Number of matched barcodes = 135856
------- OUTPUT SAVED -------
```

Hide

```
load(file = paste0(path_out,"Split/Filtered/MergedOuts/outs.collapsed.Rdata"))
```

To check the output from the *MutationCalling* function:

Hide

```
load(file = paste0(path_out,"Split/Filtered/MergedOuts/outs.collapsed.Rdata"))
```

Hide

```
outs.collapse$Sample = "RM30"
dset1 <- head(outs.collapse[order(rowSums(outs.collapse[,c("WTcount","MUTcount")]), decreasing = T),],10)
knitr::kable(dset1, caption = "Gotcha counts table", format = "html", digits = 5)  %>% kable_styling()
```

Gotcha counts table

WhiteListMatch	WTcount	MUTcount	MUTfraction	Sample
ATGTCGATCTGGCACG	415	62524	0.99341	RM30
GACCGACCAATCATCG	46198	59	0.00128	RM30
TTGTTCAGTACAACGG	87	45919	0.99811	RM30
ACATGCACATGGAGGT	59	40078	0.99853	RM30
CTCAGCTGTCCCGTGA	48	39328	0.99878	RM30
TCTCTGGTCGGATGTT	38040	69	0.00181	RM30
AGTCCGGTCGCTAGTA	35414	2670	0.07011	RM30
GTCACTCTCAAGGCAG	37870	65	0.00171	RM30
TTGCACCCATGGCCTG	35607	155	0.00433	RM30
GGAGAACTCAGTGTGT	150	35490	0.99579	RM30

We can now write a comma separated values (csv) file to use in the next steps:

Hide

```
write.csv(x = outs.collapse, file = paste0(path_out,"metadata.csv"))
```