Guide to demultiplex scifi-RNA-seq data

2020/11/09
by Daniele Barreca
edited by Andre Rendeiro

To demultiplex the scifi-RNAseq data we're using a custom version of picard tools. You can download the source code and the JAR file on our public fork: https://github.com/DanieleBarreca/picard/releases/2.19.2-CeMM.

This custom version splits the demultiplexing workflow in two parts:

The first steps reads the Illumina raw data and creates an unaligned, undemultiplexed bam file:

```
java \
    -Xmx20G \
    -Djava.util.concurrent.ForkJoinPool.common.parallelism=2 \
    -Djava.io.tmpdir=./tmp \
    -jar ${picard_jar} \
    IlluminaBasecallsToMultiplexSam \
    RUN_DIR= ${illumina_run_folder} \
    LANE=1 \
    OUTPUT= ${undemultiplexed_file} \
    SEQUENCING_CENTER=BSF \
    NUM_PROCESSORS=2 \
    APPLY_EAMSS_FILTER=false \
    INCLUDE_NON_PF_READS=false \
    TMP_DIR=tmp \
    CREATE_MD5_FILE=false \
    FORCE_GC=false \
    MAX_READS_IN_RAM_PER_TILE=9000000 \
    MINIMUM_QUALITY=2 \
    VERBOSITY=INFO \
    QUIET=false \
    VALIDATION_STRINGENCY=STRICT \
    CREATE_INDEX=false \
    GA4GH_CLIENT_SECRETS=client_secrets.json
```

The second step performs the actual demultiplexing:

```
java \
    -Xmx20G \
    -Djava.io.tmpdir=./tmp \
    -jar ${picard_jar} \
    IlluminaSamDemux \
    INPUT= ${undemultiplexed_file} \
    OUTPUT_DIR= ${output_dir} \
    OUTPUT_PREFIX= ${output_prefix} \
    LIBRARY_PARAMS= ${sample_annotation} \
    METRICS_FILE= ${output_metrics_file} \
    TMP_DIR=./tmp \
    COMPRESSION_LEVEL=9 \
    CREATE_MD5_FILE=true \
    OUTPUT_FORMAT=bam \
    BARCODE_TAG_NAME=BC \
    BARCODE_QUALITY_TAG_NAME=QT \
    MAX_MISMATCHES=1 \
    MIN_MISMATCH_DELTA=1 \
    MAX_NO_CALLS=2 \
    MINIMUM_BASE_QUALITY=0 \
    VERBOSITY=INFO \
    QUIET=false \
    VALIDATION_STRINGENCY=STRICT \
    MAX_RECORDS_IN_RAM=500000 \
    CREATE_INDEX=false \
```

GA4GH_CLIENT_SECRETS=client_secrets.json \
USE_JDK_DEFLATER=false \
USE_JDK_INFLATER=false \
DEFLATER_THREADS=4 \
MATCHING_THREADS=4 \
READ_STRUCTURE= **8M13B8S8B16M70T** \
TAG_PER_MOLECULAR_INDEX=RX \
TAG_PER_MOLECULAR_INDEX=r2

We decided to customize the picard tools to better suit our archiving needs (we archive the un-demultiplexed bam files which contain an header that can store structured information and metadata about the file) and to adapt to the evolving needs of UMIs placed arbitrarily in the reads and custom demultixplexing strategies.

In this example the sequencing run was 29+8+16+70 bases. However, the demultiplexer will split this into the structure **8M13B8S8B16M70T**:

- The first 8 bases are the UMI (tag RX)
- The next 13 bases are used as a barcode (round 1 * well-specific barcode)
- The next 8 are skipped
- The next 8 bases are also used as a barcode (sample specific barcode)
- The next 16 bases are marked as UMI and are put into the r2 tag (round two – 10x barcode)
- The last 70 bases are the transcriptome read

In this way we can create one bam file per well and per sample and parallelize the downstream analysis. The **${sample_annotation}** file looks like the following. You can see that for each sample and well there is one line and barcode 1 is the round 1 (well-specifc) barcode, while barcode 2 is the "classical" i7 sample index.

SAMPLE_NAME	BARCODE_1	BARCODE_2
PD212_scifi_1N_4lines_7650_nuclei_A01_01	AAGTGATTAGCAA	TAAGGCGA

A paired read in the un-demultiplexed (lane) bam file – output of the first demultiplexing step - would look like the following:

HWI-A00245_BSF_0774:1:1101:10013:21402 77 * 0 0 * * 0 0 **TGATCGTCAAGTGATTAGCATTTTTTTTA**
FFFFFFFFFFFFFFFFFFFFFFFFFF, BC:Z:**TAAGGCGAGGGTTATCAGGGTACA** RG:Z:HNNGMDMXX_1
QT:Z:FFFFFFFFFF:FFFFFFFFFFF

HWI-A00245_BSF_0774:1:1101:10013:21402 141 * 0 0 * * 0 0
GAGCCAGTGGGCAAAGTTGTACATTGCCCAAGCGTTCTGATAGCGAACTTAAAGATGAAAAACCAAAGAG
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,FFFFFFFFFFFFFFF:FFF RG:Z:HNNGMDMXX_1

These lines get reduced to a single line in demultiplexed bam file, which would look like the following:

HWI-A00245_BSF_0774:1:1101:10013:21402#PD212_scifi_1N_4lines_7650_nuclei_A01_01 4 * 0 0 * * 0 0
GAGCCAGTGGGCAAAGTTGTACATTGCCCAAGCGTTCTGATAGCGAACTTAAAGATGAAAAACCAAAGAG
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,FFFFFFFFFFFFFFF:FFF
r2:Z:**GGGTTATCAGGGTACA** BC:Z:**AAGTGATTAGCATTAAGGCGA**
RG:Z:HNNGMDMXX_1#PD212_scifi_1N_4lines_7650_nuclei_A01_01 QT:Z:FFFFFFFFFFFFFFFFFFFF
QX:Z:FFFFFFFF~FFF:FFFFFFFFFFF RX:Z:**TGATCGTC**

Frequently asked questions:

- **Q**: The first step is done LANE by LANE. As I have multiple lanes, I would merge the unaligned bam files after step 1 and perform step 2 on the merged file. Is that correct? Or would you recommend to perform step2 LANE by LANE and merge the bam files after step 2?.
- **A**: In our current setup we are running each lane separately and the resulting files from each lane are given as input to the downstream analysis.
- **Q**: for the **${illumina_run_folder}** do you specify the root directory of the illumina run, or do you provide the path to the BaseCalls directory?
- **A**: This is the path to the root folder of the illumina run.
- **Q**: Last question about the picard fork you are using: do I need a specific version of HTSJDK? Will it work with the lastest release, or do I need to build your fork of Picard with your fork of HTSJDK that I can find here (https://github.com/DanieleBarreca/htsjdk)? Or any specific HTSJDK version?

- **A**: The custom Picard release 2.19.2-CeMM depends on the custom htsjdl 2.19.1-CeMM release (https://github.com/DanieleBarreca/htsjdk/releases/tag/2.19.1-CeMM). However, if you download directly the jar file from the release page of our custom picard, in theory you should not need the htsjdk.