実践演習１　HISAT2/StringTie を用いたRNA-seq解析

ここではNature Protocol に掲載された、HISAT2/StringTie/Ballgown を使ったRNA-seq解析手順(Pertea, M. et al. Nat. Protoc. 11, 1650, 2016)に基づき、Bias5上でこの解析を行ってみよう。論文は、以下のサイトから取得できるので、手順の詳細については論文を参照すること。
https://pubmed.ncbi.nlm.nih.gov/27560171/

1. コース用ディレクトリの下のex1に移動し、その中のデータファイルchrX_data.tar.gzを展開する（以下、$で始まる行は実行するコマンドを表す）。

```
$ cd ~/bias2111/ex1
$ tar xvfz chrX_data.tar.gz
```

展開してできたディレクトリchrX_dataの中の以下のファイルを使用する：
chrX_data/samples: 入力となるリード配列（FASTQ ファイル）
chrX_data/indexes: リファレンスとして用いるゲノムのインデックスファイル
chrX_data/genes/chrX.gtf: 遺伝子アノテーション

2. chrX_data/samplesの下にはpaired endで12サンプルの配列データが入っている。このうち、とりあえず１サンプル分の解析を行ってみよう。以下はサンプルERR188044について、HISAT2でマッピングし、StringTieでアセンブルするという一連の解析を行うqsubスクリプトである。赤字で示した部分がqsubに特化した記述である。

qsub_hisat2.sh

```
#!/bin/sh
#PBS -l ncpus=4
#PBS -N hisat2
#PBS -q small

cd $PBS_O_WORKDIR

fname=ERR188044_chrX
sample_name=`echo $fname | sed 's/_chrX//'`
input_dir=chrX_data/samples
seq_dir=chrX_data/indexes
gene_dir=chrX_data/genes
output_dir=output

mkdir -p $output_dir
hisat2 -p $NCPUS --dta -x $seq_dir/chrX_tran \
  -1 $input_dir/${fname}_1.fastq.gz -2 $input_dir/${fname}_2.fastq.gz \
  -S $output_dir/${fname}.sam
samtools sort -@ $NCPUS -o $output_dir/${fname}.bam $output_dir/${fname}.sam
stringtie -p $NCPUS -G $gene_dir/chrX.gtf -o $output_dir/${fname}.gtf \
  -l ${sample_name}  $output_dir/${fname}.bam
```

スクリプト先頭でPBSのオプションとして-l ncpus=4が指定されているので、スクリプト内の変数$NCPUSに４が代入されることに注意しよう。３箇所ある$NCPUSは、それぞれのコマン

ドで使用するCPU（スレッド）数を指定しており、ここでは各コマンドが一貫して4CPUを使って動作することになる。

以下のコマンドでジョブをサブミットする

```
$ qsub qsub_hisat2.sh
```

ジョブの状態を確認する。*USERNAME*は自分のユーザ名を入れること。

```
$ qstat -u USERNAME
```

Status（後ろから２カラム目）がR（実行中）になっていることを確認しよう。ここがQの場合は実行待ち状態なので、実行開始されるまでしばらく待つ必要がある。

qstatで何も表示されなくなるとジョブは終了している。ls -ltで新たに出力されたファイルを確認しよう。正しく動作していれば、outputディレクトリが作成され、結果がこの中に格納されるはずである。この他に、コマンド実行時の標準出力、標準エラー出力がそれぞれhisat2.oXXXXX、hisat2.eXXXXX （XXXXXにはジョブIDが入る）というファイルに格納される。中身を見て、異常終了している様子などがないかを確認しよう。

また、以下のコマンドを実行して、実行ログとジョブが使用したリソースを確認しよう。

```
$ tracejob XXXXX （XXXXXは上記のファイル名と同じジョブIDを入れる）
```

使用リソースは終了状態を示す最後の行にresource_used.*resource_type*=###の形式で表示されており、*resource_type* としてcputはCPU時間、walltimeは実時間、ncpusはCPU数、memは使用メモリ量を表している。

3. 前項で行った解析を、fnameで指定したサンプル名を変えつつ、12サンプル分繰り返して行う。これはfor文を使ったスクリプトを書いて実行しても良いが、ここではアレイジョブを使って実行してみよう。アレイジョブでは、スクリプト中に$PBS_ARRAY_INDEXという変数を埋め込み、これに指定した範囲の値を順次代入した複数のジョブを生成し、並列に実行する。アレイジョブを使うには、入力ファイル、出力ファイルともに、file.1, file.2, ... のように同じ名前に番号を付加したファイル名として扱い、これらをスクリプト内でfile.$PBS_ARRAY_INDEXのように指定するのが基本である。

   今回の入力ファイルは残念ながらそのような名前になっていない。この場合、シンボリックリンクで各入力ファイルに上記形式の別名をつけるのがわかりやすい方法であるが、ここではややトリッキーだが、より簡潔に書けるbashの配列変数を使ったやり方でやってみよう。配列変数は複数の要素を持ち、各要素を添字で取り出せるようなデータ構造で、bashでは以下のようにして定義する。

   ```
   $ Array=( fileA fileB fileC )
   ```

   添字は０から始まるので、例えば最初の要素を取り出して表示するには

   ```
   $ echo ${Array[0]}
   ```

   とする（中括弧が必要なので注意）。
   この配列変数を使って、前項のスクリプトを、アレイジョブを使うように修正したのが以

下のスクリプトである。修正箇所は青字で表示されている。

qsub_hisat2_array.sh

```
#!/bin/sh
#PBS -l ncpus=4
#PBS -N hisat2
#PBS -q small
#PBS -J 0-11

cd $PBS_O_WORKDIR

input_dir=chrX_data/samples
sample_name=`echo $fname | sed 's/_chrX//'`
seq_dir=chrX_data/indexes
gene_dir=chrX_data/genes
output_dir=output

input_files=(`ls $input_dir | grep _1.fastq | sed 's/_1.fastq.gz//'`)
fname=${input_files[$PBS_ARRAY_INDEX]}

mkdir -p $output_dir
hisat2 -p $NCPUS --dta -x $seq_dir/chrX_tran \
  -1 $input_dir/${fname}_1.fastq.gz -2 $input_dir/${fname}_2.fastq.gz \
  -S $output_dir/${fname}.sam
samtools sort -@ $NCPUS -o $output_dir/${fname}.bam $output_dir/${fname}.sam
stringtie -p $NCPUS -G $gene_dir/chrX.gtf -o $output_dir/${fname}.gtf \
  -l ${sample_name} $output_dir/${fname}.bam
```

input_filesが12個のサンプルに対応する入力ファイル名を格納した配列変数である。定義がややトリッキーだが、どんな値が入るかはバッククオートで囲まれている以下のコマンドを実行してみれば良い（バッククオートはその中のコマンドを実行してその結果をそのまま貼り付けることを指示している）。

```
$ ls chrX_data/samples | grep _1.fastq | sed 's/_1.fastq.gz//'
```

これで、配列変数にはERR188044_chrX、ERR188104_chrX等の12個のサンプルに対応する名前が入ることが確認できる。次の行で、この配列から$PBS_ARRAY_INDEXで指定した要素を一つ取り出してfnameに代入している。配列変数の添字は０から始まるので、$PBS_ARRAY_INDEXのとる値(-Jオプション)を0から11の間としている点に注意。

ジョブをサブミットする

```
$ qsub qsub_hisat2_array.sh
```

qstatで状態を確認しよう。デフォルトではアレイジョブは１つのジョブとして表示され、StatusはBとなっている。個々のジョブ（サブジョブ）の状態を確認したい場合は、qstat -tオプションを用いる。

```
$ qstat -u USERNAME -t
```

終了したら、outputディレクトリに12個のサンプルに対する解析結果が格納されていることを確認しよう。また、標準出力、標準エラー出力を格納したファイルがサブジョブごとに作成されていることも確認しよう。

4. ここまでで、各サンプルのリードをリファレンス配列にマッピングしてアセンブルし、サンプルごとに転写配列／遺伝子の位置を記録したGTFファイルとして作成するところまで終わっている。このあと、サンプルごとのアセンブル結果をマージして一つのアノテーションファイルとして作成し、これを用いて各サンプルのマッピング結果から転写配列／遺伝子ごとのカウントデータを作成するステップに進むが、これを実行するスクリプトをpostproc.shとして作成している。これをqsubで実行せよ。

   （以下オプショナル）このスクリプトの後半では、サンプルごとにカウントデータを作成する処理を、forループを使って行っている。

```
for bamfile in $output_dir/ERR*.bam; do
        fname=`basename $bamfile .bam`
        sample_name=`echo $fname | sed 's/_chrX//'`
        stringtie -e -B -p ${NCPUS} -G stringtie_merged.gtf \
         -o ${ballgown_out}/${sample_name}/${fname}.gtf $bamfile
done
```

   これは、一つのジョブ内で逐次的に実行されるために効率が悪い。この部分の処理を、アレイジョブを使って行うスクリプトを作成してみよ。
   （解答例はansディレクトリの下にある）

5. （オプショナル：Rの実行）ここまでで元論文のステップ６までが終了し、ballgownディレクトリ内に、サンプルごとに作成した転写配列／遺伝子のカウントデータがGTFファイル、およびBallgown用のテーブル形式のデータとして格納されている。これ以降を続けて行う場合、Rを使った解析になる。ローカルマシン上のRを使う場合は、ballgown以下のファイルをscpでダウンロードしてローカル環境で解析を行うこと。Bias5上でRを実行することもできるが、その場合は、以下の点に注意すること。１）足りないパッケージが自分でインストールすることができる。その際は、各自のホームディレクトリ上にインストールされる。２）グラフィックス表示を行うにはX Windows環境がローカルに設定されていることが必要になる。その際はログイン時にssh -Y オプションを指定すること。

6. （オプショナル：データベースの参照）今回は、リファレンス配列とその検索用インデックスがあらかじめ用意されていたが、実際に解析する際はこれらを自分で用意する必要がある。その際は、bias5上に置かれた以下のデータベースファイルを参照することができる。

データベース	パス

Illumina iGenomes	/bio/db/igenomes
Ensembl	/bio/ftp/ensemble
NCBI Genomes	/bio/ftp/genomes/refseq（または genbank）

これらを用いてリファレンス配列を用意し、hisat2-buildでインデックスを作成するスクリプトとしてpreproc.shを用意したので、興味がある人は内容を確認した上で実行してみよう。このスクリプトは、デフォルトではiGenomesからデータを取得するが、コメントを付け直すことでEnsemblやNCBI　Genomesからデータを取得するように変更できる。

7. (Tips：スクリプトによる複数qsub コマンドの実行)

qsub コマンドは、ジョブをサブミットすると直ちに終了して、シェルに制御が戻り、コマンド入力が可能になる。通常はこれで問題ないが、複数のqsubコマンドを一連の処理としてスクリプトで実行したい場合は、前の処理が終わる前に次の処理が始まってしまうことが生じるため、うまく動作しない。これを避けるやり方はいくつかあるが、簡単な方法のひとつとしてqsubのオプションに-W block=true を加える方法がある。このオプションは、qsubでサブミットしたジョブの実行が終わるまで、シェルに制御が戻らずに待ち状態になるため、通常のコマンド実行と同じ感覚で使用できるようになる。

以下のスクリプトは、これまでに実行した前処理(preproc.sh)、本処理(qsub_hisat2_array.sh)、後処理(postproc.sh)の３つのqsubコマンドを一連の処理として実行する。

qsub_all.sh

```
#!/bin/sh
qsub -W block=true preproc.sh
qsub -W block=true qsub_hisat2_array.sh
qsub postproc.sh
```

実践演習２　アレイジョブを使ったBLASTの実行

PBS利用法の講義で、クエリ配列を分割してアレイジョブとして実行する例が出てきた。ここではそれを実際にやってみよう。コース用ディレクトリの下のex2に移動しよう。クエリ配列は、講義で使ったのと同じ出芽酵母ゲノムの全遺伝子翻訳配列 sce_prot.fastaである。

配列ファイルを分割するにはいろいろな方法があるが、ここでは配列ファイル操作ユーティリティプログラムSeqKitのsplitサブコマンドを使って行ってみよう。

```
$ seqkit split sce_prot.fasta -p 50
```

このコマンドでは、配列ファイル中の配列を-pオプションで指定した数のファイルに分割する。結果はsce_prot.fasta.splitというディレクトリに格納される。この中身を確認しよう。sce_prot.part_*nnn*.fastaという名前（*nnn*は001から050の数値）の50個のファイルに分割されている。

以下のスクリプトは、これらのファイルを入力としてアレイジョブとしてblastを実行する。

qsub_blast.sh

```
#!/bin/sh
#PBS -l ncpus=8
#PBS -l mem=12gb
#PBS -N blast
#PBS -q small
#PBS -J 1-50

cd ${PBS_O_WORKDIR}

seqn=`printf "%03d" $PBS_ARRAY_INDEX`
query=sce_prot.fasta.split/sce_prot.part_${seqn}.fasta

db=swissprot
outdir=sce_prot.${db}.blast_out
output=$outdir/sce_prot.part_${seqn}.tab
mkdir -p $outdir

blastp -num_threads $NCPUS -db $db -outfmt 6 -query $query \
  -evalue 0.001 -out $output
```

ただし、青字はアレイジョブに特有の記述、赤字はその他のPBSに特有の記述である。

オプション-Jを指定しているため、変数$PBS_ARRAY_INDEXに　-Jで指定した1-50の値を順に代入した　50個のサブジョブからなるアレイジョブとして実行される。ただし、ここでは入力ファイル名が０から始まる３桁の数値になっているため、この形式に変換するためにprintfコマンドを用いている。これはC言語でデータをフォーマット出力する関数printfと同じ仕様のコマンドで、"%03d"で０から始まる３桁の整数値への変換を指示している。例えば以下のコマンドは（改行なしで）010を出力する。

```
$ printf "%03d" 10
```

また、（サブ）ジョブあたりで確保するリソースとしてCPU数８個とメモリ12GBを指定している。BLASTは使えるメモリがある限りメモリ上にデータベースを展開して実行されるので、大きなデータベースを使用する際はメモリを最大限確保する必要がある。このためblastキューが用意されているが、今回用いるデータベースswissprotはそこまで大きくないので、通常のsmallキューでも十分である。

これをqsub で実行しよう。

```
$ qsub qsub_blast.sh
```

qstat -t でサブジョブごとの実行状況を確認する。

```
$ qstat -u USERNAME -t
```

このとき、すべてのジョブが実行状態　R　にはなっておらず、一部は待ち状態Qになっているはずである。なぜか。いま、50個のジョブがキューに入っており、ジョブあたり8個のCPUを確保して実行している。そこで、これらのジョブが使用するCPU数の合計は400個になっており、これはsmallキューにおいて、ユーザあたりで最大同時に実行できるCPU数300を超えている（Biasにおけるキュー構成のテーブルを参照のこと）。この場合、同時に実行されるジョブ数は300/8 = 37.5を超えることができず、残りは待ち状態となる。

一般に独立したジョブを多数流す場合、ジョブあたりのCPU数を増やすよりは、ジョブ数を増やした方が、同じCPU数を使っても効率が良いことが多い。したがって、このようにジョブあたりのCPU数を増やして待ち状態が生じると、一般に実行効率は低下する。今回の場合は、ジョブあたりのCPU数を6個以下にすればこのような制約による待ち状態は生じない。

上記の実行が終わったら、ファイルを編集してCPU数を６に変更して（ncpus=6）再実行してみよう。ただし、他のユーザも含めたキューの混み具合によっても同時に実行されるジョブ数は変わってくるので注意すること。