Assembly of the Chr1 of Arabidopsis thaliana

As the oak assembly is too fragmented ,our PacBio coverage is not sufficient (20x) and a high level of heterozygosity is complicating it even further (>1%), we wont be able to assemble pieces of the oak tree currently.

Instead we will assemble another dploid plant - *Arabidopsis thaliana*.
The raw data and the assembly stem from a very recent publication:

High contiguity Arabidopsis thaliana genome assembly with a single nanopore flow cell link

We have the following data available:

- 1 raw PacBio Sequel cell in `.bam` - format link to ENA
- 1 Nanopore run in `.fq` - format link ENA

Unfortunalty the raw `.f5` format from Nanopore is not available.

We will try to assemble the chromosome 4 from Arabidopsis which has a size of ~20 Mbp and is the smallest chromosome.

One part of the course will assemble parts of it with PacBio data and the other one with Nanopore data.

In order to have reads which are chromosome "exclusive", we already generated a chromosome 4 read sub-set.
If time allows it later we will repeat the sub-setting of chromosomal reads ourself.

Assembly of chromosome 4

The following steps have been already executed and are descibed below:

- convert RAW data in a suitable format
- map reads onto chromosome 4 (we split it further into 4 regions)
- filter reads which are suitable (remove multimapping or other problematic ones)
- subset initial total reads to reads of interest

conversion of reads to FASTA.

PacBio

PacBio sequencing data come these days in `BAM` format, previously in `h5`.
To extract the subreads one can use multiple tools.
Here we use bamtools but there are other options (e.g. DEXTRACTOR)

```
bsub 'module add UHTS/Analysis/bamtools/2.4.1; \
bamtools convert -in ../RAW/pb.bam -out SUBREADS/pb.fasta -format fasta'
```

ONT

For ONT we only have to convert here the data to `FASTA` for some of the steps.
As we have multiple `''` and `""` signs in the command it is easier to write a quick bsub script rather to submit a command directly.

```
cat fastq2fasta.sh

#!/bin/bash
zcat ../RAW/ont.fq.gz | awk  '{if(NR%4==1) {printf(">%s\n",substr($0,2));} else if(NR%4==2) print;}' >
SUBREADS/ont.fasta
```

This can then be submitted

```
bsub < fastq2fasta.sh
```

mapping reads onto chromosome 4

Next, we map all reads against the chromosome 4 in order to find reads of interest

PacBio

```
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 \
'module add UHTS/Analysis/minimap2/2.8; \
 module add UHTS/Analysis/samtools/1.4;  \
 minimap2 ../chrom4.fasta SUBREADS/pb.fasta -x map-pb -a -t 10 | samtools view -bhS | samtools sort
-o PBonChrom4.bam'
```

Which has to be indexed as well

```
bsub 'module add UHTS/Analysis/samtools/1.4; samtools index PBonChrom4.bam '
```

ONT

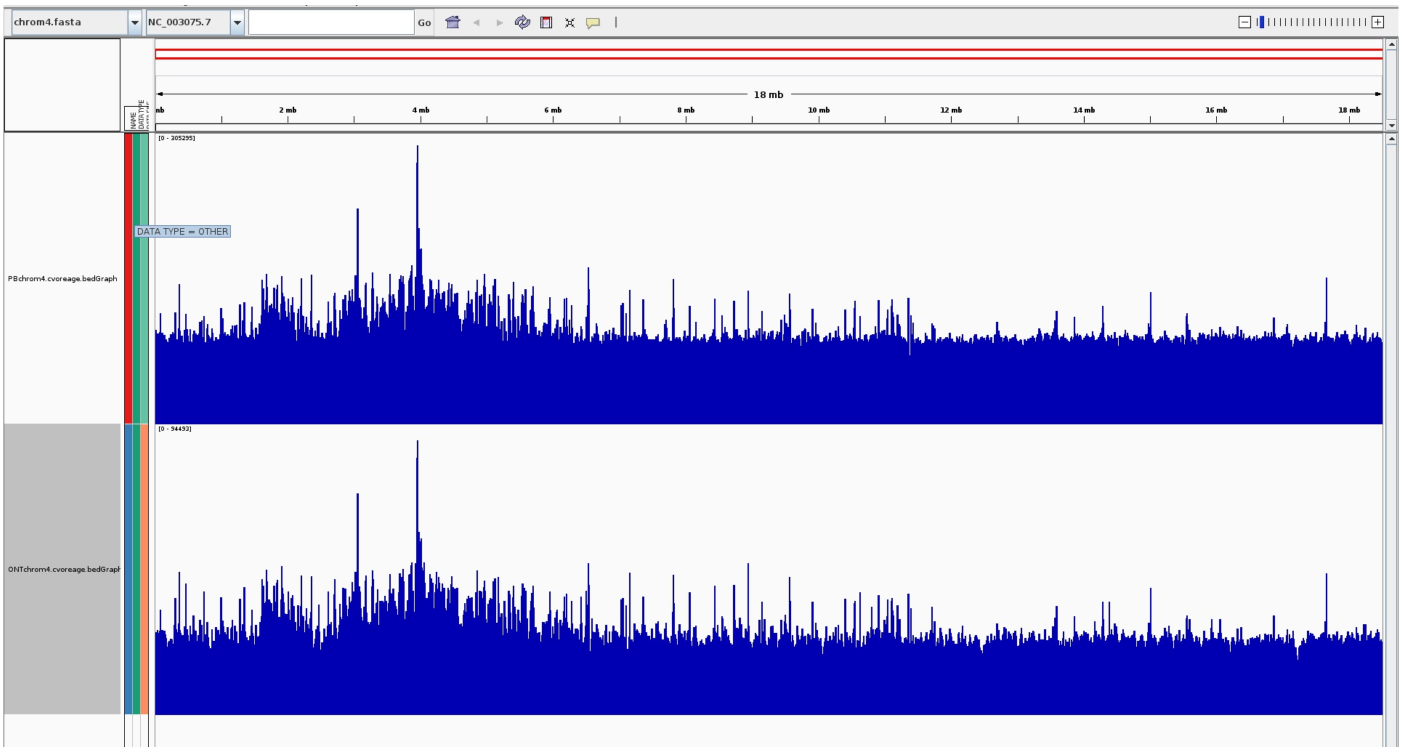
```
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 \
'module add UHTS/Analysis/minimap2/2.8; \
module add UHTS/Analysis/samtools/1.4;\
minimap2 ../chrom4.fasta SUBREADS/ont.fasta -x map-ont -a -t 30 | samtools view -bhS | samtools sort
-o ONTonChrom4.bam &&  samtools index ONTonChrom4.bam'
```

Important: the `-x map-ont` or pb specifies which data-type we are providing.

We can visualize now the coverage in order to get an idea whether we have gaps or repeats for each data set.

```
  bsub      'module add UHTS/Analysis/BEDTools/2.26.0; \
    bedtools genomecov -ibam PBonChrom4.bam -bga -g ../chrom4.fasta > PBchrom4.cvoreage.bedGraph'


  bsub      'module add UHTS/Analysis/BEDTools/2.26.0;\
  bedtools genomecov -ibam ONTonChrom4.bam -bga -g ../chrom4.fasta > ONTchrom4.cvoreage.bedGraph '
```



There is a big repeat at around 4 Mbp (maybe centromeric region), but in general we dont have any regions missing.

Lets now split the chromosome into 4 to build later 4 groups in the course:

0-5 Mbp
4-9 Mbp
8-14 Mbp
13-End Mbp

As we generate 4 overlapping regions we hopefully can later stich them together again.

Now we want all the read names which are associated with reads matching to each region:

filter reads which are suitable

We will use the alignment file here and extract all reads which have a Sam flag "0".

PacBio

```
cat  extractRegionsPB.sh

#!/bin/bash
#BSUB -e PBsplitting.error.txt
#BSUB -e PBsplitting.infor.txt

module add UHTS/Analysis/samtools/1.4;
samtools view PBonChrom4.bam "NC_003075.7:1-5000000"         | awk '{if(/^m54052/ && $2==0) print $1}'
> PB_chrom4_S1.reads &
samtools view PBonChrom4.bam "NC_003075.7:4000000-9000000"   | awk '{if(/^m54052/ && $2==0) print $1}'
> PB_chrom4_S2.reads &
samtools view PBonChrom4.bam "NC_003075.7:8000000-14000000"  | awk '{if(/^m54052/ && $2==0) print $1}'
> PB_chrom4_S3.reads &
samtools view PBonChrom4.bam "NC_003075.7:13000000-18585056" | awk '{if(/^m54052/ && $2==0) print $1}'
> PB_chrom4_S4.reads &
```

In order to later polish with PacBio we need a BAM file as input. We therefore need to 1st subset the raw BAM file based on the above information.
We made a little tools which takes the ZMV and extracts the entries:

```
for i in {1..4};
do
  bsub "perl -npe 's/.*\/([0-9]+)\/.*/\$1/' PB_chrom4_S${i}.reads | sort -n | uniq >
PB_chrom4_S${i}.zmv" &
done

for i in {1..4};
do
  bsub "/scratch/cluster/monthly/tschuepb/bin/FilterBAM ../RAW/pb.bam PB_chrom4_S${i}.bam
PB_chrom4_S${i}.zmv  > log 2>&1 &"
done
```

Note: this subsettings is almost never necessary as one would normally assemble the entire genome. We therefore wrote a quick script on the fly. There might ne as well the possibility to use samtools/bamtools to subset these alternatively but we did not try.

ONT

Again, this is in principle similar than above except that we dont need to subset a `BAM` file but (which is easier) a `FASTA` file.

```
cat  extractRegionsONT.sh



#!/bin/bash
#BSUB -e PBsplitting.error.txt
#BSUB -e PBsplitting.infor.txt

module add UHTS/Analysis/samtools/1.4;
samtools view ONTonChrom4.bam "NC_003075.7:1-5000000"         | awk '{if(/Basecall/ && $2==0) print
$1}' > ONT_chrom4_S1.reads &
samtools view ONTonChrom4.bam "NC_003075.7:4000000-9000000"   | awk '{if(/Basecall/ && $2==0) print
$1}' > ONT_chrom4_S2.reads &
samtools view ONTonChrom4.bam "NC_003075.7:8000000-14000000"  | awk '{if(/Basecall/ && $2==0) print
$1}' > ONT_chrom4_S3.reads &
samtools view ONTonChrom4.bam "NC_003075.7:13000000-18585056" | awk '{if(/Basecall/ && $2==0) print
$1}' > ONT_chrom4_S4.reads &
```

extract the subreads as FASTA for Miniasm (or Canu)

pacbio

This is similar to the above where we extract all reads from the `BAM` file

```
for i in {1..4};
do
bsub 'module add UHTS/Analysis/bamtools/2.4.1; \
  bamtools convert -in PB_chrom4_S${i}.bam -out PB_chrom4_S${i}.fasta -format fasta' &
done
```

ONT

Now we want to extract the reads in FASTA for the assembly. There are many ways to do that and I recommend you writing your own script in your prefered language. Ways to do that:

- samtools faidx
- BEDTools
- blastcmd
- ....

I used my own script (available but with no guarantee !)

for i in {1..4};
do
bsub "../software/extract_scaffold_version4.pl -i ONT_chrom4_S${i}.reads -f ont.fasta -s > ONT_chrom4_S${i}.reads.fasta " &
done

Assembly of the Data

The minimap2 + miniasm approach is extremly rapid and allows us to obtain within short time the assembly of chromosome4. It manages such an advantage in speed by assembling directly the raw reads without an hierarchical approach & correcting the reads. There are limitations though:

- large repeats and repeats with very high similarity are much more difficult to resolve
- the assembly has the original PacBio error rate
- often many non-assembled fragments remain in the assembly

It is separated into 2 steps:

1. overlapping of all reads with each otherwise
2. overlap-graph and steps to simplify it further (bubble popping, removing dead ends)

PacBio & miniasm

Overlapping:

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "module add
UHTS/Analysis/minimap2/2.8; \
      minimap2 -x ava-pb -t 5 PB_chrom4_S${i}.fasta PB_chrom4_S${i}.fasta >
PB_chrom4_S${i}.overlap.paf" &
done
```

Assemlby:

```
for i in {1..4};
do
  bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "module add
UHTS/Analysis/miniasm/0.2.r159.dirty; \
  miniasm -f PB_chrom4_S${i}.fasta PB_chrom4_S${i}.overlap.paf -R -c 10 >
PB_chrom4_S${i}.miniasm.gfa" &
done
```

The `gfa` format is very useful as it keeps the assembly path and how elements are connected.
Unfortunately most down-stream tools dont accept it yet and we need to convert the `.gfa` -format to `.fasta` -format:

The next script are very lightweight and can quickly be executed without bsub

```
for i in {1..4};
do
  awk '/^S/{print ">"$2"\n"$3}' PB_chrom4_S${i}.miniasm.gfa | fold > PB_chrom4_S${i}.miniasm.fasta
done

for i in {1..4};
do
    /Home/eschmid/tools/perl_script_contigs/assesV4.2.pl -t 1 -s 1 -i PB_chrom4_S${i}.miniasm.fasta
-l -p > PB_chrom4_S${i}.miniasm.assess.txt;
done
```

Important*: replace the above script with a general available one instead

Nanopore & Miniasm

Overlapping:

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "module add
UHTS/Analysis/minimap2/2.8; \
  minimap2 -x ava-ont -t 10 ONT_chrom4_S${i}.reads.fasta ONT_chrom4_S${i}.reads.fasta >
ONT_chrom4_S${i}.overlap.paf " "&
done
```

Assemlby:

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "module add
UHTS/Analysis/miniasm/0.2.r159.dirty; \  miniasm -f ONT_chrom4_S${i}.reads.fasta
ONT_chrom4_S${i}.overlap.paf -R > ONT_chrom4_S${i}.miniasm.gfa" &
done
```

Conversion

```
for i in {1..4};
do
  awk '/^S/{print ">"$2"\n"$3}' ONT_chrom4_S${i}.miniasm.gfa | fold > ONT_chrom4_S${i}.miniasm.fasta
done

for i in {1..4};
do
    /Home/eschmid/tools/perl_script_contigs/assesV4.2.pl -t 1 -s 1 -i ONT_chrom4_S${i}.miniasm.fasta
-l -p > ONT_chrom4_S${i}.miniasm.assess.txt;
done
```

Now importantly, these assembled sequences still have the average PacBio error rate (>12%).
In order to make it more useable we have to polish.
This can be done with the same Nanopore reads (nanopolish, racon) or Illumina reads (Pillon,Racon).
In general, polishing takes a lot of time - often more time than the assembly itself (especially with miniasm).

PacBio & Canu

The command for the assembly:

```
module add UHTS/Assembler/canu/1.6;

for i in {1..4};
do
  canu  -pacbio-raw PB_chrom4_S${i}.fasta  -d CanuPB${i} -p PB${i} genomeSize=5m &
done
```

ONT & Canu

```
module add UHTS/Assembler/canu/1.6;

for i in {1..4};
do
  canu  -nanopore-raw ONTchrom4.reads.fasta  -d CanuONT -p ONT genomeSize=20m
done
```

Polishing of the assemblies

Now we still have a rather high error rate of > 12% with miniasm which we need to correct for.
This rate is much lower with canu (less than 1%?) but it still benefits from polishing.

We will primarily use Racon for polishing to keep the work-flow similar between the PacBio and Nanopore analysis. In general, the tool quiver from PacBio is much more efficient in cleaning residual errors as well as nanopolish for ONT.

For all polishing tools, we 1st have to map reads onto the assembly which is then used to generate consensus sequences.

Racon

For Racon we can use minimap2 again in order to map the reads which is great as it is extremly fast.

PacBio Racon

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "module add UHTS/Analysis/minimap2/2.8;
minimap2 -t 10 -ax map-pb PB_chrom4_S${i}.miniasm.fasta PB_chrom4_S${i}.fasta > PB_chrom4_S${i}.miniasm.remap.sam" &
done
```

Here we need to specify a set of nodes as the software needs the chipset features "SSEv4_1 SSEv4_2 AVX" in order to function correctly. Otherwise we get core dumps.

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 -m "cpt129 cpt131 cpt132 cpt133 cpt171 cpt172 cpt181 cpt182 cpt183 cpt184 cpt185
cpt186" " module add UHTS/Assembler/racon/1.0.1; racon PB_chrom4_S${i}.fasta PB_chrom4_S${i}.miniasm.remap.sam
PB_chrom4_S${i}.miniasm.fasta -t 5 > PB_chrom4_S${i}.miniasm.racon.fasta" &
done
```

ONT racon

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "module add UHTS/Analysis/minimap2/2.8;
minimap2 -t 10 -ax map-ont ONT_chrom4_S${i}.miniasm.fasta ONT_chrom4_S${i}.reads.fasta > ONT_chrom4_S${i}.miniasm.remap.sam" &
done
```

```
for i in {1..4};
do
bsub -n 5 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 -m "cpt129 cpt131 cpt132 cpt133 cpt171 cpt172 cpt181 cpt182 cpt183 cpt184 cpt185
cpt186" " module add UHTS/Assembler/racon/1.0.1; racon ONT_chrom4_S${i}.reads.fasta ONT_chrom4_S${i}.miniasm.remap.sam
ONT_chrom4_S${i}.miniasm.fasta -t 5 > ONT_chrom4_S${i}.miniasm.racon.fasta" &
done
```

technology specific Polishing

PacBio quiver

1st we have to map again the `BAM` files onto the assembly. This does **not** work with the extracted `FASTA` files.
And it only works with the PacBio in-house tool `blasr` which is more sensitive than `minimap2` but slower.
The below used `pbalign` is essentially a wrapper from PacBio which maps and indexes the alignments.

```
for i in {1..4};
do
bsub -n 8 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "/scratch/beegfs/monthly/eschmid/smrtlink/smrtcmds/bin/pbalign
PB_chrom4_S${i}.bam PB_chrom4_S${i}.miniasm.fasta PB_chrom4_S${i}.miniasm.pbalign.bam" &
done
```

Next we can run the consensus algorithm "arrow" on it:

```
for i in {1..4};
do
bsub -n 8 -R "span[hosts=1]" -R "rusage[mem=8192]" -M 8388608 "/scratch/beegfs/monthly/eschmid/smrtlink/smrtcmds/bin/quiver
PB_chrom4_S${i}.miniasm.pbalign.bam --referenceFilename PB_chrom4_S${i}.miniasm.fasta -o PB_chrom4_S${i}.miniasm.arrow.fasta --diploid --
threaded --algorithm arrow" &
done
```

ONT nanopolish

Here we can use the same approach as for Racon, meaning mapping with minimap2