Perl for Biologists

Session 14
June 3, 2015

Practical example

Robert Bukowski
Session 13 review

Process is an object of UNIX (Linux) kernel identified by process id (PID, an integer), having an allocated region in memory containing code (binary instructions), execution queue(s), data (variables), environment (variables, arguments etc.), communication handles, etc.

Operating system can create a process at a request by another process. Request can be made using

• system() function – child process starts on the same machine, parent process waits for the completion of the child process before proceeding

• ssh – child process starts on a different machine, parent waits for the child process to finish before proceeding

• fork() function – child process is created on the same machine and is a clone of the parent process (it has an identical copy of all data, instruction set, etc.); parent process does not wait (it continues right after the clone is created).
  • Parent and clone processes have different process IDs
  • Parent and clone run concurrently
  • Parent can proceed to create more clones, and clones may spawn their own clones,…

Processes can communicate via pipes, files, signals, and/or special libraries (like Message Passing Interface [MPI] library)
**Session 13 review**

**Multithreading**: a way to execute multiple (possible different) sets of commands within a single process by using multiple execution queues

- All threads (execution queues) have access to the same memory, environment, etc.
  - Unlike `fork()`, where each process can only access its own memory segment

- No (or minimal) data replication
  - Unlike `fork()`, where whole process memory is replicated

- Good if parallelization requires access to whole large data set in each thread (rather than just to a part of the data set)

**Mixed parallelization model**:
- Multiple processes created with `fork()`
- Each (or some) of these processes may be multithreaded
- Considerations: CPU, memory, and I/O requirements
Session 13 review: fork()

```perl
my $pid = fork();
if ($pid < 0)
{
    #error
    print "ERROR: Cannot fork $!

    #further error handling code
    exit;
}
elsif ($pid == 0)
{
    #child code
    child_exec();
    exit;
}
else
{
    #master code
    master_exec();
    exit;
}
```

Function used to monitor condition of child process

```perl
my $result = waitpid($pid, WNOHANG);
```

WNOHANG flag tells `waitpid` to return the status of the child process with given pid `$pid` WITHOUT waiting for this process to complete.
Multi-process parallelization example using `fork()`

- Execute a set of independent tasks listed in a file (one command per line)
- Master process devoted to process control
- Master will create child processes, up to maximum allowed limit
- Child processes will execute the “work” part, each task in a separate directory
- Master will monitor child processes, when a child process finishes, master will create another child process if there are unprocessed tasks left
- Master will measure execution time of each task and report total time and average time per task

**Red**: modifications to `script1.pl` example assigned as homework

**Solution**: in `/home/jarekp/perl_13`

- `exercise1.pl` script (modified `script1.pl`)
- `exercise1exe.pl` toy script (to be executed as task)
- `exercise1.tasks` list of tasks to be run through `exercise1.pl`
For each position on reference genome, scan through the stack of aligning reads (pileup) and determine the numbers of reads carrying each allele (allele depths) and average base qualities.

Call SNP if:
- enough overall depth at the site (>=5?)
- non-reference alleles are significant proportion of total depth (>=0.25?)
- sufficient average base quality (>=20?)

This is a very primitive approach to SNP calling – **DO NOT use in real research**

However, more involved (and more correct) SNP callers can be developed using the allele depth and quality information.
Simple SNP caller workflow

Reference genome sequence (reference.fa, reference.fa.fai)

Alignment (alignments.bam, alignments.bam.bai)

Region to process (chromosome, start, end)

Generate pileup information (output one line per position) using samtools program

```
samtools mpileup -f reference.fa alignments.bam -r Chr1:200-5000
```

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Position</th>
<th>Ref</th>
<th>Alt</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chr1</td>
<td>997</td>
<td>T</td>
<td></td>
<td>.^k.</td>
</tr>
<tr>
<td>Chr1</td>
<td>1000</td>
<td>G</td>
<td></td>
<td>..aT</td>
</tr>
<tr>
<td>Chr1</td>
<td>1006</td>
<td>G</td>
<td></td>
<td>..-3GCT,,-3gct.-3GCT</td>
</tr>
<tr>
<td>Chr1</td>
<td>1007</td>
<td>G</td>
<td></td>
<td>.*,**,</td>
</tr>
<tr>
<td>Chr1</td>
<td>1008</td>
<td>C</td>
<td></td>
<td>.*,**</td>
</tr>
<tr>
<td>Chr1</td>
<td>1009</td>
<td>T</td>
<td></td>
<td>.*,**,</td>
</tr>
<tr>
<td>Chr1</td>
<td>1025</td>
<td>C</td>
<td></td>
<td>..+4CGTA,+4cgta,.</td>
</tr>
<tr>
<td>Chr1</td>
<td>1036</td>
<td>G</td>
<td></td>
<td>.$.,.</td>
</tr>
</tbody>
</table>

For each output line (position on the genome)
- extract allele depth from base pileup string
- extract base qualities and compute per-allele averages
- decide if allele depths and qualities indicate a SNP
  - if yes, output position, SNP, and allele depths

in Perl
Implementation

We will implement the workflow as a function `snp_call_range()`. This function, as well as a couple of other auxiliary functions, will be collected in file `simple_snpcaller.pl`.

The main program calling this function is simple:

```perl
#!/usr/local/bin/perl

# Load the functions
require "simple_snpcaller.pl";

# Read the input parameters
my ($bamfile,$reffasta,$chr,$range_start,$range_end) = @ARGV[0..4];

# Do the SNP-calling
snp_call_range($bamfile,$reffasta,$chr,$range_start,$range_end);

print "SNP calling done";
```
sub snp_call_range
{
    my ($bamfile, $reffasta, $chr0, $range_start, $range_end) = @_;
    my $range = $range_start . "-" . $range_end;

    my $cmd = "samtools mpileup -f $reffasta $bamfile -r $chr0:$range ";
    open (in, "$cmd |");
    open (out, ">$output.$range");

    my ($line, $chr, $pos, $refbase0, $refbase, $pstr, $qstr, $depth);
    my @aux; my @allele_nums; my @allele_qc;
    while ($line = <in>)
    {
        chomp $line;
        @aux = split "\t", $line;
        ($chr, $pos, $refbase0) = @aux[0..2];
        ($depth, $pstr, $qstr) = @aux[3..5];

        $refbase = uc $refbase0;
        if ($depth > 0)
        {
            ($allele_nums[0..5], $allele_qc[0..5]) = analyze_pileup_strs($pstr, $qstr, $refbase);
            my ($issnp, $majornonref) = primitive_snp_caller($allele_nums, $allele_qc, $refbase);
            if ($issnp)
            {
                print out "$chr\t$pos\t$refbase/$majornonref";
                for (my $i=0; $i<=5; $i++)
                {
                    print out "\t$allele_nums[$i]";
                }
            }
        } else {
            print out "\n";
        }
    }
}

close in; close out;
}
samtools mpileup: how does it work

**Example alignment**

<table>
<thead>
<tr>
<th>997</th>
<th>1000</th>
<th>1006</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>G</td>
<td>GGCT</td>
</tr>
<tr>
<td>AGTGTGAATGATGATGAAAATGGAC</td>
<td>C-----</td>
<td>ATCGTGTACG TGCCCGTGC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1025</th>
<th>1036</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>AGTGTGAATGATGATGAAAATGGAC</td>
<td>C-----</td>
</tr>
</tbody>
</table>

**Samtools mpileup output**

<table>
<thead>
<tr>
<th>Chr1</th>
<th>997</th>
<th>T</th>
<th>4</th>
<th>.^k.,.</th>
<th>CHH;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chr1</td>
<td>1000</td>
<td>G</td>
<td>4</td>
<td>..aT</td>
<td>8; ?A</td>
</tr>
<tr>
<td>Chr1</td>
<td>1006</td>
<td>G</td>
<td>5</td>
<td>..-3GCT,,-3gct.-3GCT</td>
<td>&lt;?ABC</td>
</tr>
<tr>
<td>Chr1</td>
<td>1007</td>
<td>G</td>
<td>5</td>
<td>.*,,**</td>
<td>C!H!!</td>
</tr>
<tr>
<td>Chr1</td>
<td>1008</td>
<td>C</td>
<td>5</td>
<td>.*,,**</td>
<td>1!9!!</td>
</tr>
<tr>
<td>Chr1</td>
<td>1009</td>
<td>T</td>
<td>5</td>
<td>.*,,**</td>
<td>7!9!!</td>
</tr>
<tr>
<td>Chr1</td>
<td>1025</td>
<td>C</td>
<td>5</td>
<td>..+4CGTA,+4cgta,</td>
<td>&gt;&gt;&gt;&gt;&gt;;&gt;</td>
</tr>
<tr>
<td>Chr1</td>
<td>1036</td>
<td>G</td>
<td>5</td>
<td>.$.,,</td>
<td>-/0HE</td>
</tr>
</tbody>
</table>

Session 14: Practical example  Perl for Biologists 1.2
analyze_pileup_strs()

**Purpose:** extract allele depths and average qualities from pileup strings

First, ignore (remove) start and end of read markers

```perl
$pstr =~ s/\$/\g/;
$pstr =~ s/^\[[x00-x7F]//g;
```

Process deletions

```perl
my @matches = ( $pstr =~ m/([-0-9]+[ACGTNacgtn]+)/g );
foreach $mtch (@matches)
{
    $mtch =~ /-\d+$/;
    $len = $1 + 0;
    $torepl = "$len" . substr($mtch,length($1)+1,$len);
    $pstr =~ s/-$torepl//g;
}
$pstr =~ s/\*/D/g;
```

(all reads carrying the deletion * will have allele “D”)

Session 14: Practical example
Perl for Biologists 1.2
Process insertions:

```perl
@matches = ( $pstr =~ m/\+[0-9]+[ACGTacgtn]+/g );
foreach $mtch (@matches)
{
    $mtch =~ /\+[0-9]+/;
    $len = $1 + 0;
    $torepl = "$len" . substr($mtch,length($1)+1,$len);
    # replace each insertion by an "allele" I, regardless of length
    or sequence of the insert
    $pstr =~ s/\+$torepl/I/g;
    $pstr =~ s/\+$torepl/I/g;
}
```

All reads carrying insertion will have allele called “I”, regardless of sequence or length of the insertion.

String $pstr now contains only characters [.,ACGTacgtIDNn] (., stand for reference allele)

Now we can count occurrences and **average base qualities** of various alleles....
About base quality score notation

Base qualities reported by a sequencing platform are given in terms of the phred score \( Q \) which is an integer number such that

\[
e = 10^{-0.1Q}
\]

is the probability that the base call is wrong.

In the FASTQ format (and in BAM files), the phred score is represented by a single character with ASCII code

\[ Q + 33 \]

(this is the so-called phred+33 representation; older Illumina platforms used numbers other than 33).

Thus, in perl, the phred score can be computed as

\[
\text{my } \$q = \text{ord}(\$qualchar) - 33;
\]

where \$qualchar is the variable holding a character read from FASTQ or BAM file.
Now we can count occurrences and average base qualities of various alleles:

```perl
# Convert the base string to array of characters:
my @apstr = split //, $pstr;
# Convert the quality string to array of characters:
my @aqstr = split //, $qstr;
# Loop over all elements of the arrays
for($i=0;$i<=$#apstr;$i++)
{
    # Set the allele and update its counter
    $base = uc $refbase;
    if($apstr[$i] ne "." && $apstr[$i] ne ",")
    {
        $base = uc $apstr[$i] ;
    }
    $basecount{$base}++;
    # Update the quality score for this allele
    # quality scores are given in phred+33 notation
    $myqc = ord($aqstr[$i]) - 33; # ord() converts char into number (ASCII code)
    $qscore{$base} += $myqc;
}
```
Compute averages of quality scores (per allele)

```perl
# Normalize the quality scores
foreach $base (keys %basecount)
{
    if($basecount{$base} eq "") { next; }
    if($basecount{$base} > 0)
    {
        $qscore{$base} = $qscore{$base}/$basecount{$base};
    }
}
```
Wrap it all in a function:

```perl
sub analyze_pileup_strs {
    use strict;

    my $pstr = @_[0];
    my $qstr = @_[1];
    my $refbase = @_[2];

    # Process the base string
    #......

    # Count alleles and calculate average base qualities
    #...........

    # Return the results
    my @allele_nums = ($basecount{"A"}, $basecount{"C"}, $basecount{"G"}, $basecount{"T"}, $basecount{"I"}, $basecount{"D"});
    my @allele_qc = ($qscore{"A"}, $qscore{"C"}, $qscore{"G"}, $qscore{"T"}, $qscore{"I"}, $qscore{"D"});

    return (@allele_nums, @allele_qc);
}
```

The function is called from within main program as:

```perl
# get $pstr (base pileup string) and $qstr (base quality string) from samtools pileup
my @allele_nums;
my @allele_qc;
(@allele_nums[0..5], @allele_qc[0..5]) = analyze_pileup_strs($pstr,$qstr,$refbase);
```
### primitive_snp_caller()

Call SNP if

- enough overall depth at the site (>=5?)
- non-reference alleles are significant proportion of total depth (>=0.25?)
- sufficient average base quality (>=20?)

This is a very primitive approach to SNP calling – **DO NOT use in real research**

However, more involved (and more correct) SNP callers can be developed using the allele depth and quality information returned by `analyze_pileup_strs` routine.

If SNP detected, write position, SNP, and allele depths to an output file

- Name of the output file should reflect the range of coordinates processed (e.g., `output.2000-50000`, etc.)
This is a very primitive SNP caller - don't use for real research...

```perl
sub primitive_snp_caller {
    my @alleles = ("A","C","G","T","I","D");
    my @allele_nums;
    my @allele_qc;
    my $refbase;
    (@allele_nums[0..5], @allele_qc[0..5], $refbase) = @_; # Count total depth and the depth of non-reference alleles
    # Record the major non-reference allele
    my $totdepth = 0;
    my $nonrefdepth = 0;
    my $majornonrefbase = "";
    my $maxnonref = 0;
    my $avqc = 0;
    for(my $i=0;$i<=5;$i++)
    {
        $totdepth += $allele_nums[$i];
        $avqc += $allele_qc[$i] * $allele_nums[$i];
        if($alleles[$i] ne $refbase)
        {
            $nonrefdepth += $allele_nums[$i];
            if($allele_nums[$i] > $maxnonref) { $maxnonref = $allele_nums[$i];
                $majornonrefbase = $alleles[$i];
            }
        }
    }
    $avqc = $avqc/$totdepth;

    # ... function continues on next slide......
}
```

In the main program, the function is called like this

```perl
my ($issnp, $majornonref)=primitive_snp_caller(@allele_nums, @allele_qc, $refbase);
```

If $issnp=1, the SNP has been found and $majornonref contains the most abundant alternative allele.
Stitching it all together:

• Collect all three functions in one file, say `simple_snpcaller.pl`
• The functions are re-usable – can be called from any higher-level perl program
• Write a simple main program to test everything (`main.snp_caller.pl`)

```perl
#!/usr/local/bin/perl

# Load the functions
require "simple_snpcaller.pl";

# Read the input parameters
my ($bamfile,$reffasta,$chr,$range_start,$range_end) = @ARGV[0..4];

# Do the SNP-calling
snp_call_range($bamfile,$reffasta,$chr,$range_start,$range_end);

print "SNP calling done";
```
Running the program

Create your working directory (if not yet there)
```
cd /workdir
mkdir abc123  (if directory not yet there; substitute your login ID for “abc123”)
cd abc123
```

Copy the example data files and scripts, link to reference sequence file
```
cp /shared_data/misc/maizev2/maize.fa 1
cp /home/jarekp/perl_14/*.bam* .
cp /home/jarekp/perl_14/*.pl .
```

Run the example
```
./main_snp_caller.pl alignments.bam maize.fa 10 50000000 53000000 >& log &
```

Fragment of output file (after about 2 minutes):

```
output.50000000-53000000
10   50000142   A/T   4   0   0   5   0   0
10   50001220   T/C   0   7   0   0   0   0
10   50001355   G/T   0   0   6   6   0   0
10   50001791   A/I   4   0   0   0   9   0
10   50002119   C/A   6   3   0   0   0   0
10   50002227   C/T   0   8   0   3   0   0
10   50002322   G/T   1   0   11   5   0   0
10   50002816   A/T   3   0   0   5   0   0
10   50002847   A/G   6   0   3   0   0   0
10   50003085   G/A   3   0   8   1   0   0
10   50003534   C/T   0   5   0   4   0   0
10   50004378   G/A   3   0   4   0   0   0
```
Homework:

Parallelize the SNP calling by splitting the chromosome region into smaller sub-regions and processing multiple such sub-regions concurrently using a pre-defined number of CPU cores.

**Hint:** modify `script1.pl` of Session 13:

- require "simple_snpcaller.pl"
- Read in all needed parameters from command line in the beginning of `script1.pl`
- Convert `main_snpcaller.pl` into a function `child_exec()` [see `script1.pl`] that accepts appropriate arguments
- Modify function `start_task()` [see `script1.pl`] to accept appropriate arguments