Perl for Biologists

Session 15

June 10, 2015

Practical Examples

Jaroslaw Pillardy

Session 14 Exercises Review

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_snp_caller.pl into a function child_exec() [see script1.pl] that accepts appropriate arguments

- Modify function start_task() [see script1.pl] to accept appropriate arguments

exercise1.pl (1)

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

require "simple_snpcaller.pl";

use POSIX ":sys_wait_h";

if($#ARGV != 6)
{
  print "USAGE: ./exercise1.pl bam_file reference_file chromosome range_start range_end
         max_task ntasks\n";
  exit;
}

my ($bamfile,$reffasta,$chr,$range_start,$range_end, $maxtask, $ntasks) = @ARGV;

my $block = int(($range_end - $range_start + 1) / $ntasks);
my $rest = ($range_end - $range_start + 1) - $block*$ntasks;

print "start: $range_start, end: $range_end, block: $block, rest: $rest\n";
my @task_start;
my @task_end;
```

exercise1.pl (2)

```
for(my $i=0; $i<$ntasks; $i++)
{
  if($i == 0)
  {
     $task_start[$i] = $range_start;
  }
  else
  {
     $task_start[$i] = $task_end[$i-1] + 1;
  }
  $task_end[$i] = $task_start[$i] +  $block - 1;
  if($i<$rest){$task_end[$i]++;}
  print "task " . ($i+1) . " range " . $task_start[$i] . " - " . $task_end[$i] . "\n";
}

#initial fork child processes
my @procs;
my @procs_tasks;
my $task = 0;
print "STARTING: maxtask=$maxtask ntasks=$ntasks\n";
my $outfile = "output.$range_start-$range_end";
unlink($outfile);
for(my $i=0; $i<$maxtask; $i++)
{
  $task++;
  print "starting child $i task $task \n";
  $procs[$i] = start_task($bamfile,$reffasta,$chr,$task_start[$i],$task_end[$i],
                          @procs);
  $procs_task[$i] = $task;
  print "    pid $procs[$i]\n";
}
```

exercise1.pl (3)

```
#waiting for child processes to finish and execute remaining tasks in their place
while(1)
{
  sleep(1); #there is no need to check every milisecond - it would use too much CPU
  my $n=0;
  for(my $i=0; $i<=$#procs; $i++)
  {
    if($procs[$i] != 0)
    {
      my $kid = waitpid($procs[$i], WNOHANG);
      if($kid <= 0)
      {
        #process exists
        $n++;
      }
      else
      {
        print "Child " . ($i+1) . " task " . $proc_task[$i]  . " finished (pid=" .
                  $procs[$i] . ")\n";
        my $ct = $procs_task[$i] - 1;
        system("cat output." . $task_start[$ct] . "-" . $task_end[$ct] . " >>
                  $outfile");
        unlink("output." . $task_start[$ct] . "-" . $task_end[$ct]);
        $procs[$i] = 0 ;
```

```
        if($task < $ntasks)
        {
          $task++;
          $procs[$i] = start_task($bamfile,$reffasta,$chr,$task_start[$i],$task_end[$i],
                                  @procs);
          $procs_task[$i] = $task;
          print " child " . ($i+1) . " restarted for task $task with pid $procs[$i]\n";
          $n++;
        }
      }
    }
  }
  if($n==0){last;}
}

print "ALL DONE\n";

sub child_exec
{
  my ($bamfile,$reffasta,$chr,$range_start,$range_end) = @_;
  print("tmp$range_start.$range_end\n");
  mkdir("tmp$range_start.$range_end");
  chdir("tmp$range_start.$range_end");
  system("ln -s ../$reffasta $reffasta");
  system("ln -s ../$bamfile $bamfile");
  system("ln -s ../$bamfile.bai $bamfile.bai");
  snp_call_range($bamfile,$reffasta,$chr,$range_start,$range_end);
  system("mv output* ..");
  chdir("..");
  system("rm -rf tmp$range_start.$range_end");
}
```

```
sub start_task
{
  my ($bamfile,$reffasta,$chr,$range_start,$range_end, @procs) = @_;

  my $pid = fork();
  if($pid < 0)
  {
  #error
    print "\n\nERROR: Cannot fork child $i\n";
    for(my $j=0; $j<=$#procs; $j++)
    {
      system("kill -9 " . $procs[$j]);
    }
    exit;
  }
  if($pid == 0)
  {
  #child code
    child_exec($bamfile,$reffasta,$chr,$range_start,$range_end);
    exit;
  }
  #master - continue, $pid contains child pid
  return $pid;
}
```

Sorting sequences based on BLAST results

We have a fasta file with sequences and BLAST results obtained using these sequences.

The task is to save the sequences to separate fasta files based on the e-values.

There are 42 sequences in `sgn.fasta` file, the blast results are in `blastresults` file.

Sorting sequences based on BLAST results

Blast results were obtained against TAIR protein database.

```
makeblastdb -in TAIR7_pep_20070320 -dbtype prot
```

BLAST command was

```
blastx -query sgn.fasta -db TAIR7_pep_20070320 -outfmt 6 -out
blastresults -evalue 1e-5 -max_target_seqs 1 -num_threads 8
```

Sorting sequences based on BLAST results

BLAST tabular output -outfmt 6

1.	**qseqid**	**query (e.g., gene) sequence id**
2.	sseqid	subject (e.g., reference geome) sequence id
3.	pident	percentage of identical matches
4.	length	alignment length
5.	mismatch	number of mismatches
6.	gapopen	number of gap openings
7.	qstart	start of alignment in query
8.	qend	end of alignment in query
9.	sstart	start of alignment in subject
10.	send	end of alignment in subject
11.	evalue	expect value
12.	bitscore	bit score

Sorting sequences based on BLAST results

Our script will need three parameters:

- Original fasta file
- BLAST results file name
- File with e-values determining bins:
    evalue1
    evalue2
    …
    evalueN

script1.pl

```
#!/usr/local/bin/perl
#make sure we have proper command line parameters

#get input fasta file name, check if the file exists

#get blast results file name, check if the file exists

#get evalues file name, check if the file exists
```

script1.pl

```
#!/usr/local/bin/perl
#make sure we have proper command line parameters

#get input fasta file name, check if the file exists

#get blast results file name, check if the file exists

#get evalues file name, check if the file exists
#open evalues file, read evalues and store in an array

#order the evalues array ascending

#open blast results

#LOOP start
#read results, store evalues in hash with seq id as key
#split each line by \t, id->[0], evalue->[10]
#if multiple hits, keep the best evalue
#LOOP end
```

script1.pl

```
#!/usr/local/bin/perl
#make sure we have proper command line parameters

#get input fasta file name, check if the file exists

#get blast results file name, check if the file exists

#get evalues file name, check if the file exists
#open evalues file, read evalues and store in an array

#order the evalues array ascending

#open blast results

#LOOP start
#read results, store evalues in hash with seq id as key
#split each line by \t, id->[0], evalue->10
#if multiple hits, keep the best evalue
#LOOP end

#using Bio::Perl open output files for sequences
#using Bio::Perl read input fasta file

#LOOP start
#print sequence to a file depending on evalue
#LOOP end

#close files
```

script1.pl (1)

```
#!/usr/local/bin/perl
#make sure we have proper command line parameters
if($#ARGV != 2)
{
   print "USAGE: script1.pl fastafile balstresults binfile\n";
   exit;
}
#get input fasta file name, check if the file exists
my $input = $ARGV[0];
if(! -e $input)
{
   print "ERROR: file $input does not exist\n";
   exit;
}
#get blast results file name, check if the file exists
my $blastresults = $ARGV[1];
if(! -e $blastresults)
{
   print "ERROR: file $blastresults does not exist\n";
   exit;
}
#get evalues file name, check if the file exists
my $binfile = $ARGV[2];
if(! -e $binfile)
{
   print "ERROR: file $binfile does not exist\n";
   exit;
}
```

```
#open evalues file, read evalues and store in an array
open IN, $binfile or die "ERROR: $!\n";
my @bins;
while(my $line = <IN>)
{
   chomp $line;
   push @bins, $line*1.0;
}
close(IN);
print $#bins . " bin delimiters read\n";

#order the evalues array ascending
@bins = sort {$a <=> $b} @bins;

#open blast results
open IN, $blastresults or die "ERROR: $!\n";
```

script1.pl (3)

```
my %id2evalue;
#LOOP start
while($line = <IN>)
{
#read results, store evalues in hash with seq id as key
#split each line by \t, id->[0], evalue->10
   chomp $line;
   my @data = split/\t/, $line;
   my $id = $data[0];
   my $evalue = 1.0*$data[10];
   if(defined $id2evalue{$id})
   {
#if multiple hits, keep the best evalue
      my $oldevalue = $id2evalue{$id};
      if($oldvalue > $evalue)
      {
         $id2evalue{$id} = $evalue;
      }
   }
   else
   {
      $id2evalue{$id} = $evalue;
   }
#LOOP end
}
close(IN);
print 1*(keys %id2evalue) . " results read\n";
```

```
use Bio::SeqIO;
#using Bio::Perl open output files for sequences
my @out;
for(my $i=0; $i<=$#bins+1; $i++)
{
   $out[$i] = Bio::SeqIO->new(-file => ">$input.bin.$i" , -format => 'Fasta');
}

#using Bio::Perl read input fasta file
my $in  = Bio::SeqIO->new(-file => $input , -format => 'Fasta');
```

script1.pl (5)

```
#LOOP start
while (my $seq = $in->next_seq() )
{
#print sequence to a file depending on evalue
   my $n = 0;
   my $id = $seq->display_id;
   for(my $i=0; $i<=$#bins; $i++)
   {
      if($id2evalue{$id} < $bins[$i])
      {
         $out[$i]->write_seq($seq);
         $n = 1;
      }
   }
   if($n == 0)
   {
      $out[$#bins+1]->write_seq($seq);
   }
#LOOP end
}

#close files
for(my $i=0; $i<=$#bins+1; $i++)
{
   $out[$i]->close();
}
$in->close();
```

Joining several output files (tables) by column

There are several files with expression data for various individuals.

Each of the files looks like this:

```
#Gene   Transcript      A_depth B_depth Tot_depth       No_SNPs Depth_per_SNP   Av_par1_ratio
Stdev_par1_ratio
AC148152.3_FG005        AC148152.3_FGT005       1       1       2       2       1.00    0.50    0.50
AC148152.3_FG008        AC148152.3_FGT008       0       2       2       2       1.00    0.00    0.00
AC148167.6_FG001        AC148167.6_FGT001       43      26      69      2       34.50   0.62    0.01
AC149475.2_FG003        AC149475.2_FGT003       23      11      34      1       34.00   0.68    0.00
```

Task: merge the files to create one table

Joining several output files (tables) by column

What is the unique element in each row?

It is Transcript ID.

We will join tables using Transcript ID as a key.

```
#Gene   Transcript      A_depth B_depth Tot_depth       No_SNPs Depth_per_SNP   Av_par1_ratio
Stdev_par1_ratio
AC148152.3_FG005        AC148152.3_FGT005       1       1       2       2       1.00    0.50    0.50
AC148152.3_FG008        AC148152.3_FGT008       0       2       2       2       1.00    0.00    0.00
AC148167.6_FG001        AC148167.6_FGT001       43      26      69      2       34.50   0.62    0.01
AC149475.2_FG003        AC149475.2_FGT003       23      11      34      1       34.00   0.68    0.00
```

Joining several output files (tables) by column

In the new file column Transcript ID and Gene ID will stay the same as in the component files,

-> followed by data columns form file 1

-> followed by data columns from file.2

Gene<TAB>Transcript<TAB>Data1_file1<TAB>Data2_file1<TAB>[…]<NL>
+
Gene<TAB>Transcript<TAB>Data1_file2<TAB>Data2_file2<TAB>[…]<NL>
=
Gene<TAB>Transcript<TAB>Data1_file1<TAB>Data2_file1<TAB>[…]<TAB>Data1_file2<TAB>Data2_file2<TAB>[…]<NL>

script2.pl

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

#get the file names

#first file will be appended with data from second

#need to know number of columns in each file when padding later
#declare hash, which will contain whole lines, key: transcript ID.

#read the first file into hash
#close file1
```

script2.pl

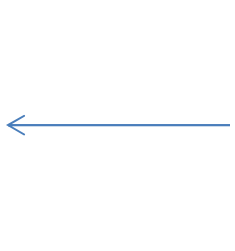
```
#!/usr/local/bin/perl

#get the file names

#first file will be appended with data from second

#need to know number of columns in each file when padding later
#declare hash, which will contain whole lines, key: transcript ID.

#read the first file into hash
#close file1

#open file2
#create new header for the output file
#skip first two columns, append rest

#loop over the lines in file2
        #split line in file2
        #if the line exists in file1 append new data to this line
        #if the line does not exist in file1 create a new entry with zeros for file1
        #flag the key as processed, need to add zeros to not processed when printing

#close file2
```

script2.pl

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

#get the file names

#first file will be appended with data from second

#need to know number of columns in each file when padding later
#declare hash, which will contain whole lines, key: transcript ID.

#read the first file into hash
#close file1

#open file2
#create new header for the output file
#skip first two columns, append rest

#loop over the lines in file2
        #split line in file2
        #if the line exists in file1 append new data to this line
        #if the line does not exist in file1 create a new entry with zeros for file1
        #flag the key as processed, need to add zeros to not processed when printing

#close file2

#open output file
#print header
#Print out all records from hash:
#any record not appended need to be padded with zeros

#close output file
```

script2.pl (1)

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

#get the file names
my $masterfile = $ARGV[0];
my $newfile = $ARGV[1];
#first file will be appended with data from second

#need to know number of columns in each file when padding
my $masterlength = 0;
my $newlength = 0;

my $nfile1 = 0;
my $nfile2 = 0;
my $nfile3 = 0;
my $nfile1new = 0;
my $nfile2new = 0;
my $appended = 0;

#declare hash, which will contain whole lines, key: transcript ID.
my %master = {};

#read the first file into hash
open in, $masterfile  or die "Cannot open file1 '$masterfile'\n";
my $mastheader = <in>;
chomp $mastheader;
while(my $line=<in>)
{
        chomp $line;
        my @aux = split /\t/, $line;
        $master{$aux[1]} = $line;
        $masterlength = $#aux - 1;
        $nfile1++;
}
#close file1
close in;
```



variables to collect statistics

```
#open file2
open in, $newfile  or die "Cannot open file2 '$newfile'\n";

#create new header for the output file
#skip first two columns, append rest
my $file2header = <in>;
chomp $file2header;
my @file2headarr = split /\t/, $file2header;
$mastheader .= "\t" . join("\t", @file2headarr[2..$#file2headarr]);

my %donetrn;
#loop over the lines in file2
while(my $line=<in>)
{
        chomp $line;
        #split line in file2
        my @aux = split "\t", $line;
        $newlength = $#aux - 1;
        #if the line exists in file1 append new data to this line
        if($master{$aux[1]} ne "")
        {
                $master{$aux[1]} = $master{$aux[1]} . "\t" . join("\t",@aux[2..$#aux]);
                $appended++;
        }
```

```
        #if the line does not exist in file1 create a new entry with zeros for file1
        else
        {
                $nfile2new++;
                my $val = join("\t",@aux[0..1]);
                for(my $i=1; $i<=$masterlength; $i++)
                {
                        $val = $val . "\t0";
                }
                $master{$aux[1]} = join("\t",$val,@aux[2..$#aux]);
                $newlines++;
        }
        #flag the key as processed, need to add zeros to not processed when printing
        $donetrn{$aux[1]} = 1;
        $nfile2++;
}
#close file2
close in;
```

```
#open output file
open out, ">${masterfile}_${newfile}" or die "Cannot open output file
                                                '${masterfile}_${newfile}'\n";
#print header
print out "$mastheader\n";

#Print out all records from hash:
#any record not appended need to be padded with zeros
foreach $key (keys %master)
{
        if(!defined $donetrn{$key})
        {
                $nfile1new++;
                for(my $i=1; $i<=$newlength; $i++)
                {
                        $master{$key} .= "\t0";
                }
        }
        print out "$master{$key}\n";
        $nfile3++;
}
#close output file
close(out);
print "$nfile1 lines read from $masterfile\n";
print "$nfile2 lines read from $newfile\n";
print "$nfile3 lines outputted\n";
print "$appended lines merged from two files\n";
print "$nfile1new lines came only from $masterfile\n";
print "$nfile2new lines came only from $newfile\n";
```