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 ;
```

exercise1.pl (4)

```
        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.

script1.pl

We will develop the script together during the session

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

We will develop the script together during the session