Generating a Genome Assembly with PCAP

In recent years, the whole-genome shotgun (WGS) technique has become the method of choice for generating genome sequences. In this technique, the entire genome is randomly sheared and cloned into a small insert library (typically plasmids or fosmids). The plasmid inserts are then sequenced either individually or as pairs from each end of the insert to achieve an average coverage of 5- to 10-fold across each base. These "sequence reads," or "reads" for short, are typically 400 to 800 base pairs long, and, depending on the size and coverage of the WGS, there may be millions, or tens of millions, of them. The challenge is then to assemble these reads together into large contiguous stretches of sequence by a process of finding overlaps among them and piecing them together like a jigsaw puzzle. The Parallel Contig Assembly Program (PCAP) is one of several such software tools that make WGS assembly possible.

The PCAP package is a set of programs for generating a genome assembly from a mixture of paired and unpaired sequence reads. PCAP can handle a genome of 30 Mb on a single-processor computer, a genome of 300 Mb on a multiprocessor computer with 10 processors or more, and a genome of 3 Gb on a computer cluster of 100 or more processors.

This unit presents instructions (see Basic Protocol 1) for using PCAP on a multiprocessor computer in a 300-Mb genome-assembly project typical of an invertebrate or large microbe, using an example data set. The use of other parameter values in PCAP for more advanced performance is also described in that protocol. Using PCAP on a computer cluster to assemble a 3-Gb genome, typical of the size of a vertebrate genome, is described in Basic Protocol 2. Accompanying support protocols describe the downloading and installation of PCAP (Support Protocol 1), the preparation of the input files (Support Protocol 2), and the generation of the `fofn.con` file used by PCAP (Support Protocol 3).

PCAP consists of several main programs for generating an assembly. First, the `pcap` program computes pairwise overlaps between reads. Next, the `bdocs` program uses these overlaps to calculate the coverage depths at each region of the genome. After that, the `bclean` program removes overlaps between reads with extremely high coverage depths, which typically correspond to repetitive regions of the genome that would otherwise confuse the assembly process. In the subsequent step, the `bcontig` program builds the assembly layout, placing each read into an ungapped region of contiguous sequence known as a "contig," and then assembling the contigs into larger gapped structures known as "supercontigs." Finally, the `bconsen` program generates the consensus sequences of the contigs. The consensus is a single sequence that selects the most likely base pair from among the overlapping reads that make up a contig, thereby eliminating many sequencing errors in the raw reads.

The PCAP package also contains a few minor programs for formatting an assembly and collecting statistics on it. The `bform` program combines a number of files of consensus sequences into a single file and compiles lists of all reads that were either used or omitted from the assembly. The `bpair` program reports the status of read pairs at the contig level and at the read level. The `n50` program collects the N_{50} lengths (a standard measure of the distribution of contig length) and counts of contigs and supercontigs. The `xstat` program reports the distribution of the distances of read pairs in supercontigs.

Contributed by Xiaoqiu Huang and Shiaw-Pyng Yang
Current Protocols in Bioinformatics (2005) 11.3.1-11.3.23

In addition, PCAP contains several Perl scripts for automatically running the major and minor programs in the proper order. The `autopcap` script automatically runs the programs to produce a small-scale assembly on a shared-memory computer. The other Perl scripts produce a large-scale assembly on a distributed cluster of computers and a computer with large memory. The `sublapjobs` Perl script generates many job shell scripts for computing overlaps with the `pcap` code and submits the job scripts to the cluster for execution in parallel. The `runtigcode` Perl script runs the `bdocs`, `bclean`, and `bcontig` programs in the proper order on the large-memory computer. The `subsenjobs` Perl script generates many job shell scripts for computing consensus sequences with the `bconsen` code and submits the job scripts to the cluster for execution in parallel. The `runstatcode` Perl script runs the minor programs in the proper order on the large-memory computer.

BASIC PROTOCOL 1

PRODUCING AN ASSEMBLY WITH PCAP USING AN EXAMPLE DATA SET

The `autopcap` Perl script is used to generate automatically an assembly on a small or medium data set on a shared-memory computer with multiple processors. The assembly is distributed in a number of output files. The formats and contents of the output files are specified.

Necessary Resources

Hardware

Unix or Linux computer with 3 Gb of memory and 15 Gb of free hard disk space. PCAP requires 3 Gb of memory and 15 Gb of hard disk space for projects in the 100 to 300 Mb range. Larger genome projects will require more memory and disk space. A rule of thumb for estimating the memory and disk space requirements of PCAP is to let *N* be the total number of raw bases in Mb for a project, whereby the memory and disk space requirements of PCAP for the project are about 15*N* and 75*N* Mb, respectively. For projects that exceed the 3-Gb memory limit, one will have to use 64-bit hardware, such as that provided by the AMD64, Itanium, or Sun platforms.

Software

PCAP (*http://seq.cs.iastate.edu*). For instructions on downloading and installing PCAP, see Support Protocol 1. PCAP is free to academic users, but a licensing agreement is required for commercial users. If PCAP requires more than 3 Gb of memory, then the 64-bit version of PCAP should be obtained.

Files

PCAP takes as input a number of pairs of `gzip`-compressed base and quality files in FASTA format, a file of read pairs, and a file of all base file names without the `gz` suffix. The files for the example used in this unit are included in the PCAP package. See Support Protocols 2 and 3 for generation of input files.

Producing an assembly on an example data set

1. Run PCAP with the default parameter values. From the Unix command line, enter the PCAP distribution directory. Execute the following commands:

```
cd example
../autopcap fofn > auto.log &
```

The `example` *directory contains a small data set. The PCAP code is in the parent directory of* `example`. *The file* `fofn` *contains the base file names (without the* `gz` *suffix that indicates compression) of all the reads that are to be assembled, and the file* `fofn.con`

lists the read pairs among this set. Make sure that the directory contains only files for the data set before running PCAP. The `autopcap` *job should take a minute or two on the small data set. When it is done, the last line in the file* `auto.log` *should read:*

```
The autopcap job is completed.
```

If the job runs successfully, it will leave the assembly results and statistics in the following files:

`contigs.bases`: Contig base sequences in FASTA format.
`contigs.quals`: Contig quality scores in FASTA format.
`supercontigs`: Overview of supercontigs.
`reads.placed`: The positions of reads in the assembly.
`reads.unplaced`: The names of reads that are not in the assembly.
`fofn.pcap.scaffold*.ace`: Ace files of contigs for the Consed assembly viewer and editor program.
`readpairs.contigs`: Major unused read pairs between contigs.
`readpairs.reads`: The positions of read pairs in the assembly.
`fofn.con.pcap.results`: The status of read pairs.
`fofn.con.pcap.sort.stat`: The distribution of read pair distances.
`fofn.pcap.n50`: The length statistics of contigs and supercontigs.
`fofn.pcap.contigs*.snp`: Alignment columns with potential SNPs.

File formats for the first five items were designed by J.C. Mullikin and D.B. Jaffe for use in the mouse whole-genome assembly project.

2. The output files are simple text files, but they can be quite large and readers are not advised to try and view them in a Unix text editor such as `emacs` or `vi`. Instead, view the output files using a command-line pager such as `less` or `more`. Using `less`, one may scroll forward by pressing the letter `f` on the keyboard and backward by pressing the letter `b`, and quit the program by pressing the letter `q`. For example, view the `contigs.bases` (Fig. 11.3.1) file by typing:

```
less contigs.bases
```

3. When reading the output files, it helps to understand the PCAP convention for naming contigs and supercontigs. A supercontig is a list of contigs that are ordered and oriented with respect to the genome; the supercontigs are named `Supercontig0`, `Supercontig1`, `Supercontig2`, and so on, in descending order of size. The

```
>Contig0.1
CGCGGAATTCCTCTCCACCTCTCTTACCCATGTGGCACTTAGCACACACTGCTTTGGATT
CTATTTAAACACTCAACTTGCTTATATCTTGGTTTCTTGGTTGTCCAAGGAGAGCACAGG
CCTCTGGAGGGCAGGAGATGTATAGAAAACGAATTTTTCTGTAACTAAAAGAAATTTTTG
TTTTTAGGCCGAGTCTTGTTCTGTCTTCCAGGCTGGAGTGCAATGGTGCAATTTCTGCTC
ACTGCAACCTCCGCCTCCTGGGTTCAAGCGTTTCTCCTGCCTCAGCCTCCCGAGTAGCTG
GGACTACAGGTGCCCTCCACCATGCCCAGCTAATTTTTGTATTTTCAGTAGAGATCGGGT
TTCATCATGGTGGTCATGCTGGTCTTGAACTCCTG
>Contig0.2
AATAAAACAGCCACTTAGCCACCAGCCTACTTAACAAAACCAGTGTCGTGGAAGTCCTGT
ATGCTGTTCCTCTGATTACAGCCCCTCCTCTATCCTGGAGGCAGCCTCTCTCCTAAATTC
TGGGGTAGTCACTCCCTGACTCTTTTAACAAAACAGCTGTATACCCCAAGCATGTATTCT
GAAACAGTAACAGTATTGCTGGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTT
GGGAGGCCGAGGCGGGCGGATCACGAGGTCAGGAGATCGAGACCACGGTGAAACCCCGTC
TCTACTAAAAATACAAAAAATTAGCCGGGCGCAGTGGTGGGCGCCTGTAGTCCCAGCTAC
```

Figure 11.3.1 The top part of the `contigs.bases` file produced on the example data set.

```
supercontig Supercontig0
contig Contig0.1
gap 835 750 * 2
contig Contig0.2
supercontig Supercontig1
contig Contig1.1
gap -1772 687 * 24
contig Contig1.2
supercontig Supercontig2
contig Contig2.1
supercontig Supercontig3
contig Contig3.1
```

Figure 11.3.2 The entire content of the `supercontigs` file produced on the example data set.

contigs that make up `Supercontig0` are named `Contig0.1`, `Contig0.2`, `Contig0.3`, and so forth. For example, `Contig3.1` corresponds to the first contig in `Supercontig3`.

4. Using the command-line pager, examine the `supercontigs` file (Fig. 11.3.2). This file shows, for each supercontig, the contigs and the gaps in the supercontig. The file consists of lines, each starting with one of the three keywords: `supercontig`, `contig`, or `gap`. Each of these three types of line has a slightly different format (items in italics are supercontig name, contig name, or gap information):

   `supercontig` line format:
   `supercontig` *`supercontig_name`*
   `contig` line format:
   `contig` *`contig_name`*
   `gap` line format:
   `gap` *`gap_length gap_length_deviation * number_of_read_pairs`*

   The contig name on a contig line is the same as the one used in the file `contigs.bases`. *The contigs before and after the gap are linked by a number of read pairs, with a gap-length estimate obtained from each read pair. The gap length and gap length deviation are the average and standard deviation of the gap length estimates from the read pairs.*

5. Next, examine the `reads.placed` file (Fig. 11.3.3). This file has a line for each read that is used in the assembly. The line contains the following items from the left to the right, separated by spaces.

   Column 1: `*`
   Column 2: name of the read
   Column 3: left trimming position of the read
   Column 4: number of bases in the trimmed read
   Column 5: orientation of the read in a contig (`0` = given, `1` = reverse)
   Column 6: name of the contig
   Column 7: name of the supercontig
   Column 8: estimated start position of the trimmed read in the contig
   Column 9: estimated start position of the trimmed read in the supercontig

   The read may be trimmed by PCAP to remove poorly aligning ends. The left trimming position of the read is the position immediately after the left poor end. If the left poor end is empty, then the left trimming position is `1`. *In this file, the value* `0` *indicates that the read is in the same orientation in which it appears in the raw read file (its "given" orientation) and the value* `1` *indicates that the read is in its reverse orientation. Unfortunately, for historical reasons, in all other files produced by PCAP,* `1` *stands for the same orientation*

```
* ISB1287.x1 1 176 0 Contig0.1 Supercontig0 1 1
* ISB943.y2 1 202 0 Contig0.1 Supercontig0 1 1
* ISB545.x1 1 186 1 Contig0.1 Supercontig0 9 9
* FFC97.y1 1 285 0 Contig0.1 Supercontig0 1 1
* FFC463.y1 1 328 0 Contig0.1 Supercontig0 1 1
* FFC13-P21.x1d-01-F 1 252 0 Contig0.1 Supercontig0 94 94
* FFC1336.x1 1 302 0 Contig0.1 Supercontig0 94 94
* FFC139.y1 18 467 0 Contig0.2 Supercontig0 1 1233
* ISB764.x1 42 394 0 Contig0.2 Supercontig0 181 1413
* ISB1666.y1 41 515 0 Contig0.2 Supercontig0 119 1351
```

Figure 11.3.3 The top part of the `reads.placed` file produced on the example data set.

```
* uml57e10.b1  repeat
* umy52e11.g1  repeat
* FFC608.x1  repeat
* ISB770.x1  unused
* ISB724.x1  unused
* umm38b11.g1  unused
* uml16c03.g1  unused
* umz53f08.g1  unused
* unb97b05.g1  unused
```

Figure 11.3.4 The entire content of the `reads.unplaced` file produced on the example data set.

and `0` *stands for the reverse orientation. The start position of the trimmed read in the contig is with respect to the left end of the contig. For column 9, the lengths of the gaps that occur upstream of the read in the supercontig are included to compute the start position of the trimmed read with respect to the left end of the supercontig.*

6. Examine the `reads.unplaced` file (Fig. 11.3.4). This file has a line for each read that is not used in the assembly in the format:

   `* read_name short_explanation`

   Possible short explanations are chimera, repeat, short, and unused, but the explanations are not always accurate. The `repeat` *category of unplaced reads includes two groups of reads: unique reads that have no overlap with other reads and highly repetitive reads that have only repetitive overlaps with other reads. The designation* `short` *is given to an unplaced read if the right clipping position of the read is not sufficiently larger than the left clipping position of the read. The designation* `unused` *is given to an unplaced read if the read is used in construction of the layout of a contig but removed in generation of the consensus sequence of the contig. In general, a read may not be used in the assembly if the read has no overlaps with other reads, is from a highly repetitive region of the genome, or is in a region of the genome that is hard to assemble by PCAP.*

7. *Optional:* Examine the `.ace` files. These files are intended for viewing the assembly with the Consed program (see *UNIT 11.2*). To view the contigs in Consed, start the program with the `-nophd` option and open one of the following `.ace` files produced by PCAP:

   `fofn.pcap.scaffold0.ace`
   `fofn.pcap.scaffold1.ace`

```
C0.1 1 405
C0.2 1 27642 C2.1 1 631 1 0 1 C3.1 0 443 1 0 1
C1.1 1 15491
C1.2 1 12401
C2.1 1 631 C0.2 1 27642 0 0 1
C3.1 1 443 C0.2 0 27642 1 0 1
```

Figure 11.3.5 The entire content of the `readpairs.contigs` file produced on the example data set.

The `scaffold*.ace` *files describe the structure of the supercontigs. PCAP distributes the supercontigs among these files in such a way that each file contains multiple supercontigs. To determine which* `scaffold*.ace` *file contains a particular supercontig, divide the supercontig number by the number of* `.ace` *files and take the remainder. Hence if searching for the file that contains* `Supercontig5` *and there are two* `.ace` *files, the remainder after dividing 5 by 2 is 1, and the desired supercontig will be present in* `fofn.pcap.scaffold1.ace`.

Below are two more `.ace` files produced by PCAP:

`fofn.pcap.singleton0.ace`
`fofn.pcap.singleton1.ace`

The file `fofn.pcap.singleton0.ace` *is an* `.ace` *file of singlet reads that are linked by read pairs to or are associated with supercontigs in* `fofn.pcap.scaffold0.ace`. *The file* `fofn.pcap.singleton1.ace` *is similarly related to* `fofn.pcap.scaffold1.ace`.

8. Examine the `readpairs.contigs` file (Fig. 11.3.5). This file has a line for each contig in the assembly. Each line contains information on the relationship between the two members of a read pair, and can help resolve problems in the assembly. The first three columns of each line are:

   Column 1: name of the current contig
   Column 2: orientation of the contig (`1` = given, `0` = reverse)
   Column 3: length of the contig in bp

   The contig name is in a short form like `C0.1` *for* `Contig0.1`. *If the current contig is not linked by read pairs to contigs in other superconitgs, then there are no additional columns on the line. Otherwise, there are up to four groups of six consecutive columns each on the line. The groups represent the strongest read pair links between the current contig and contigs in other supercontigs. The strength of a link is the number of read pairs that support this link. The groups are in the same format and the format of group 1 is shown below:*

   Column 4: name of a contig in another supercontig
   Column 5: orientation of the contig (`1` = given, `0` = reverse)
   Column 6: length of the contig in bp
   Column 7: direction of the contig (`1` = downstream, `0` = upstream)
   Column 8: distance of the link in bp
   Column 9: number of read pairs

   If the link places the contig in group 1 downstream from the current contig, then the direction of the contig in group 1 is `1`. *Otherwise, the direction is* `0`. *If the contig in group 1 overlaps with the current contig, then the distance of the link is* `0`. *Otherwise, the distance of the link is the distance between the contig in group 1 and the current contig.*

```
ISB1287.x1 1 176 C0.1 1 1 no read pair
ISB943.y2 1 202 C0.1 1 1 S 4000 ISB943.x1 0 638 C0.2 1975 3207
ISB545.x1 0 186 C0.1 9 9 no read pair
FFC97.y1 1 285 C0.1 1 1 S 4000 FFC97.x1 0 583 C0.2 2652 3884
FFC463.y1 1 328 C0.1 1 1 no read pair
FFC13-P21.x1d-01-F 1 252 C0.1 94  94 no read pair
FFC1336.x1 1 302 C0.1 94   94 no read pair
FFC139.y1 1 467 C0.2 1 1233 S 4000 FFC139.x1 0 475 C0.2 3354 4586
ISB764.x1 1 394 C0.2 181 1413 S 4000 ISB764.y1 0 441 C0.2 5751 6983
ISB1666.y1 1 515 C0.2 119 1351 S 4000 ISB1666.x1 0 540 C0.2 4911 6143
```

Figure 11.3.6 The top part of the `readpairs.reads` file produced on the example data set.

This file is useful for finding assembly problems by examining read pair links between contigs in different supercontigs. Weak read pair links may be due to wrong read pairs. However, strong read pair links may indicate problems due to polymorphism, repeats, or duplication.

9. Examine the `readpairs.reads` file (Fig. 11.3.6). This file shows the status of read pairs in terms of the occurrences of reads in contigs and supercontigs. In other words, the file is an extension of the `reads.placed` file by showing on the same line the positions of the two paired reads in the assembly. The file has a line for each read in the assembly. The left part of the line has 6 columns for the current read:

   Column 1: name of the read
   Column 2: orientation of the read (`1` = given, `0` = reverse)
   Column 3: length of the trimmed read in bp
   Column 4: name of the contig where the read occurs
   Column 5: position of the read in the contig
   Column 6: position of the read in the supercontig

   If the current read is not associated with another read by a read pair, then the remaining part of the line is the phrase `no read pair`. *Otherwise, the remaining part of the line contains information about the read pair and the other read in the read pair:*

   Column 7: status of the read pair (`S` = satisfied, `D` = dissatisfied)
   Column 8: the distance of the read pair in bp
   Column 9: name of the other read
   Column 10: orientation of the read (`1` = given, `0` = reverse)
   Column 11: length of the trimmed read in bp
   Column 12: name of the contig where the read occurs
   Column 13: position of the read in the contig
   Column 14: position of the read in the supercontig

   The file is arranged in increasing order of supercontig number. The lines for a supercontig are arranged in the order of the positions of reads in the supercontig. This file is useful for studying a region of the assembly by taking a close look at individual read pairs in the region.

10. Examine the `fofn.con.pcap.results` file (Fig. 11.3.7). This file reports the status of each read pair in the `fofn.con` file. See Support Protocol 2 on the format of the `fofn.con` file. It has a line for each entry in the `fofn.con` file. The first five columns on the line are from the `fofn.con` file. The remaining columns on the line report the status of the read pair. Each of the nine status categories is indicated by a short descriptive phrase, listed below:

```
ISB943.y2    ISB943.x1     1000   7000 ISB943       3718 satisfied in a scaffold
FFC97.y1     FFC97.x1      1000   7000 FFC97        4282 satisfied in a scaffold
FFC139.y1    FFC139.x1     1000   7000 FFC139       3581 satisfied in a   contig
FFC1964.y1   FFC1964.x1    1000   7000 FFC1964      3420 satisfied in a   contig
ISB1666.y1   ISB1666.x1    1000   7000 ISB1666      5020 satisfied in a   contig
ISB757.x1    ISB757.y1     1000   7000 ISB757       3902 satisfied in a   contig
ISB2267.y2   ISB2267.x1    1000   7000 ISB2267      5099 satisfied in a   contig
FFC1023.y1   FFC1023.x1    1000   7000 FFC1023      4916 satisfied in a   contig
ISB764.x1    ISB764.y1     1000   7000 ISB764       5635 satisfied in a   contig
ISB1475.x1   ISB1475.y1    1000   7000 ISB1475      6013 satisfied in a   contig
```

Figure 11.3.7 The top part of the `fofn.con.pcap.results` file produced on the example data set.

Category 1: *satisfied in a contig*. The two reads of the read pair occur in a contig in the expected orientation and distance.

Category 2: *unsatisfied in the distance in a contig*. The reads occur in a contig in the expected orientation but the distance between the reads in the contig is outside the expected range.

Category 3: *satisfied in a scaffold*. The reads occur in different contigs of a supercontig in the expected orientation and distance.

Category 4: *unsatisfied in the distance in a scaffold*. The reads occur in different contigs of a supercontig only in the expected orientation. The distance between the reads in the supercontig is outside the given range.

Category 5: *singlet*. One of the reads is not in any contig.

Category 6: *short*. One of the reads is in a short supercontig of length less than 10,000 bp.

Category 7: *terminal*. Each of the reads is in an end of a supercontig.

Category 8: *redundant*. The read pair is redundant. For a group of redundant read pairs, only one of them is used in construction of supercontigs.

Category 9: *unsatisfied*. The read pair is not in any of the above categories.

For Categories 1 through 4, column 6 of the line is the distance between the reads in a supercontig. The number of read pairs in each category is reported at the end of the file. In particular, the read pairs in Category 9 are likely to be due to assembly problems or wrong read pairs.

11. Examine the `fofn.con.pcap.sort.stat` file (Fig. 11.3.8). This file reports the distribution of distances on column 6 of the `.results` file for read pairs in Categories 1 through 4. For each read pair, the distance is the approximate distance between the reads in a supercontig. The information contained in this file is useful for producing accurate read pair distance ranges for PCAP by revising the initial distance ranges in the `.con` file (see Support Protocol 2).

12. Examine the `fofn.pcap.n50` file (Fig. 11.3.9). This file reports the statistics on the numbers and lengths of contigs and supercontigs. It contains the total length of contigs in bp, the number of contigs, the maximum contig length in bp, the contig N_{50} length, and the contig N_{50} number. The file also contains the statistics for major contigs of length $\geq$1000 bp. The statistics for supercontigs and major supercontigs are also in the file. In addition to the N_{50} statistics, the statistics for N_{10} through N_{100} are also reported. N_{50} is defined such that half of the assembled base pairs are contained in contigs of size N_{50} or larger, and is a better measure of assembly contiguity than average contig length.

```
Total Percentage is the percentage of read pairs
     that are within the distance given at left.
Incremental Percentage is the percentage of read pairs
     that are between the previous and current distances.
A first line in the group:
     FFC1002.x1   FFC1002.y1     1000   7000 FFC1002   4240 satisfied in a contig

A last line in the group:
     ung44g11.b1  ung44g11.g1     500   6000 ung44g11   4683 satisfied in a contig

Distance   Total Percentage  Incremental Percentage
       400     2.19178             2.19178
       800     6.57534          4.38356
      1200    10.41096          3.83562
      1600    13.97260          3.56164
      2000    16.71233          2.73973
      2400    19.72603          3.01370
      2800    21.91781          2.19178
      3200    23.28767          1.36986
      3600    29.31507          6.02740
      4000    34.79452          5.47945
      4400    44.38356          9.58904
      4800    59.72603         15.34247
      5200    79.17808         19.45205
      5600    87.12329          7.94521
      6000    91.23288          4.10959
      6400    92.60274          1.36986
      6800    95.61644          3.01370
      7200    96.71233          1.09589
      7600    98.08219          1.36986
      8400    98.90411          0.82192
      8800    99.17808          0.27397
      9200    99.45205          0.27397
      9600    99.72603          0.27397
     16800   100.00000          0.27397
Number of constraints in the group: 365
```

Figure 11.3.8 The entire content of the `fofn.con.pcap.sort.stat` file produced on the example data set.

```
Total ctg sum: 57003, Number of  contigs : 6, Max ctg length: 27642
Ctg N40 length: 27642,   Ctg N40 number: 1

Total major  ctg sum: 55534, Number of major   Contigs : 3
Major ctg N40 length: 27642, Major   ctg N40 number: 1

Total ctg sum: 57003, Number of  contigs : 6, Max ctg length: 27642
Ctg N50 length: 15491,   Ctg N50 number: 2

Total major  ctg sum: 55534, Number of major   Contigs : 3
Major ctg N50 length: 15491, Major   ctg N50 number: 2

Total ctg sum: 57003, Number of  contigs : 6, Max ctg length: 27642
Ctg N60 length: 15491,   Ctg N60 number: 2

Total major  ctg sum: 55534, Number of major   Contigs : 3
Major ctg N60 length: 15491, Major   ctg N60 number: 2
```

Figure 11.3.9 The middle part of the `fofn.pcap.n50` file produced on the example data set.

13. Examine the `fofn.pcap.contigs*.snp` file (Fig. 11.3.10). This file reports information about potential SNPs in the contig sequences. PCAP computes an alignment of reads for each contig. A column of the alignment is good if the column has exactly one base type with a high quality score. A potential SNP is scored if an alignment column has two or more base types with high quality scores and the column is surrounded by five consecutive good columns on each side. For each potential SNP, the file contains one line that begins with the keyword `SP`, and two or more lines that

```
SP  16  10143  Contig1.2
BS  A  20  1089  umv97a06.b1
BS  A  48  1054  une41d02.g1
BS  A  32  1028  unb01d03.g1
BS  A  50  996  unf45d05.g1
BS  A  49  974  umk08g01.g1
BS  A  53  967  umy41a11.b1
BS  A  50  407  une04d03.b1
BS  A  57  1068  una19h06.b1
BS  A  63  1045  umn13f09.b1
BS  A  63  1117  umv42f07.b1
BS  A  52  638  une09h01.b1
BS  A  53  1074  umw12d02.g1
BS  A  63  1046  una09c06.g1
BS  A  34  1028  umy51f11.b2
BS  T  60  1012  umv99b11.g1
BS  T  35  991  umz90c12.g1
```

Figure 11.3.10 The entire content of the `fofn.pcap.contigs1.snp` file produced on the example data set.

begin with the keyword `BS`. The `SP` line provides information about the SNP, and the `BS` lines provide information about each base in the alignment column in which the SNP is contained. An `SP` line has four columns:

Column 1: `SP` (keyword)
Column 2: number of `BS` lines that follows
Column 3: position of the SNP in the contig sequence
Column 4: name of the contig

A `BS` line consists of five columns:

Column 1: `BS` (keyword)
Column 2: base
Column 3: quality score
Column 4: length of the trimmed read with the base
Column 5: name of the read

Advanced parameters

By changing its parameter values, one can tune PCAP to reduce running time or to tweak the nature of the assembly that is produced.

14. Reduce the time requirement of PCAP. If the computer system is a shared memory system with at least four processors, then one can tell PCAP to take advantage of the four processors by providing the `-y` option:

```
../autopcap fofn -y 4 > auto.log &
```

This tells PCAP to perform the overlap task in parallel by doing the four subtasks (computation of overlaps for each of four subsets of reads) at the same time, one subtask per processor. The consensus generating task is also carried out in parallel. Performing the overlap and consensus tasks in parallel typically reduces the running time of PCAP, provided that the number of subtasks (four) does not exceed the number of processors available on the system. Using a number of subtasks generally reduces the computation time.

15. Run PCAP on one processor. If the computer system has only one processor, then use the one processor with the `-p` option:

```
../autopcap fofn -p 0 > auto.log &
```

This tells PCAP to run one subtask at a time. The default option is `-p 1`, which tells PCAP to run all the subtasks at the same time.

16. If the computer system is short of memory for PCAP, trade time for space with the `-p` and `-y` options:

```
../autopcap fofn -p 0 -y 8 > auto.log &
```

This tells PCAP to partition the task into eight subtasks and run one subtask at a time. The memory requirement of the subtask is about one eighth that of the task. It is important to tell PCAP to use only one processor with the `-p 0` option. If the `-p 1` option is used, then PCAP runs eight subtasks at the same time, with each subtask requiring its own space.

17. Control overlaps with high-quality base differences. If there are no SNPs in the reads, then remove overlaps with the stringent `-d` option:

```
../autopcap fofn -d 90 > auto.log &
```

If there are SNPs in the reads, then tolerate overlaps with the loose `-d` option:

```
../autopcap fofn -d 150 > auto.log &
```

PCAP rejects overlaps with a quality difference score greater than the value for the `-d` parameter. The quality difference score of an overlap is the sum of quality scores of base differences. The quality score of a difference at bases of quality scores q1 and q2 is max(0, min(q1, q2) − 20). For example, an overlap with five differences at bases of quality score 45 has a quality difference score of 125, where the quality score of each base difference is max(0, min(45, 45) − 20) = 25.

18. Handle highly repetitive reads. Reject highly repetitive overlaps with the `-l` and `-s` options:

```
../autopcap fofn -l 50 -s 7000 > auto.log &
```

Tolerate highly repetitive overlaps with the `-l` and `-s` options:

```
../autopcap fofn -l 50 -s 4000 > auto.log &
```

*PCAP finds highly repetitive reads based on the depth of coverage by overlaps. A read with a coverage depth greater than the value (called `rcd`) for the `-l` parameter is highly repetitive. An overlap with an adjusted similarity score less than the value for the `-s` parameter is rejected. The adjusted similarity score of an overlap is obtained by multiplying the similarity score of the overlap by a uniqueness factor. The similarity score of an overlap is the sum of quality-weighted scores of base matches and differences in the overlap. The quality-weighted score of a match at bases of quality values q1 and q2 is 2 * min(q1, q2), whereas the quality-weighted score of a difference at bases of quality values q1 and q2 is -6 * min(q1, q2).*

The uniqueness factor of an overlap depends on the uniqueness of the regions in the overlap. If the regions in the overlap have a coverage depth greater than `rcd`, then the uniqueness factor is 0. Otherwise, the uniqueness factor is between 0 and 4, where 1 corresponds to a coverage depth of rcd/2, 2 corresponds to a coverage depth of rcd/4, 3 corresponds to a coverage depth of rcd/8, and 4 corresponds to a coverage depth of rcd/16. For example, an overlap of 50 matches at bases of quality values 20 with a coverage depth of rcd/8 is 6000.

SUPPORT PROTOCOL 1

DOWNLOADING AND INSTALLING PCAP

This protocol provides instructions for downloading and installing the PCAP program. Although the PCAP program runs on all Unix and Linux computer systems, each type of computer system has its own version of PCAP binary code. If it is necessary to run PCAP on two different types of computer systems, it is then necessary to download both versions of PCAP binary code for the systems.

Necessary Resources

Hardware

Unix or Linux computer with 3 Gb of memory and 15 Gb of free hard disk space. Large assembly projects require more memory and disk space.

Software

Versions of PCAP binary code for different types of Unix and Linux computer systems are available at *http://seq.cs.iastate.edu*. PCAP is free to academic users, but a licensing agreement is required for commercial users. If PCAP requires more than 3 Gb of memory, then the 64-bit version of PCAP should be obtained.

Files

A small example data set is included in the PCAP package. A large example data set and an assembly produced by PCAP on the data set are also available from *http://seq.cs.iastate.edu*.

1. Point the browser at *http://seq.cs.iastate.edu* and click the word Download on the top line to the right of "PCAP and CAP3 code:". This brings up a licensing agreement for academic users. Fill out the form to register, read the agreement, and click the button at the bottom of the page to accept the agreement.
2. Click the PCAP button on the next page. This brings up a list of `.tar` files of PCAP code for a number of computer systems.

   The PCAP code is available for the following types of computer systems.

   64-bit Opteron Linux: a Linux system on 64-bit Opteron processors
   64-bit Itanium Linux: a Linux system on 64-bit Itanium processors
   32-bit Pentium Linux: a Linux system on 32-bit Pentium processors
   64-bit Sparc Solaris: a Sun Solaris system on 64-bit Sparc processors
   32-bit Sparc Solaris: a Sun Solaris system on 32-bit Sparc processors
   64-bit Opteron Solaris: a Sun Solaris system on 64-bit Opteron processors
   64-bit Alpha Tru64: an HP Unix system on 64-bit Alpha processors
   64-bit Itanium Altix: a SGI Linux system on 64-bit Itanium processors
   64-bit MIPS IRIX : a SGI Unix system on 64-bit MIPS processors
   32-bit PowerPC G5 OS X: a 32-bit Mac Unix system on 64-bit PowerPC G5 processors

   The PCAP code for additional types of computer systems will be added to the list in the future. For example, when a 64-bit version named Tiger of Mac OS X is available, the 64-bit PCAP code for OS X will be added to the list.
3. Assuming that one's computer system is a Linux system on 64-bit Opteron processors, click the `.tar` file for 64-bit Opteron Linux to download the PCAP code for that computer system. Save the `.tar` file in a directory on the local computer system and go to the directory. The `.tar` file is named `pcap.linux.opteron64.tar`.

4. Unpack the `.tar` file in the directory by executing the command:

```
tar xvf pcap.linux.opteron64.tar
```

The `pcap` code is in the directory `pcap.linux.opteron64`. Go to the directory and show the main/minor programs in the directory by executing the commands:

```
cd pcap.linux.opteron64
ls
```

PREPARATION OF INPUT FILES

This protocol provides instructions for preparing one's data set for PCAP. PCAP requires a number of `gzip`-compressed base and quality files in FASTA format, a file of read pairs, and a file of all base file names without the `gz` suffix. The contents and formats of the input files are specified.

Necessary Resources

Hardware

Unix or Linux computer with 3 Gb of memory and 15 Gb of free hard disk space. Large assembly projects require more memory and disk space.

Software

Versions of PCAP binary code for different types of Unix and Linux computer systems are available at *http://seq.cs.iastate.edu*. PCAP is free to academic users, but a licensing agreement is required for commercial users. If PCAP requires more than 3 Gb of memory, then the 64-bit version of PCAP should be obtained.

Files

A small example data set is included in the PCAP package. A data set for input to PCAP consists of base and quality files in FASTA format. The files can be generated by the `Phred` program and screened for sequencing vectors by the `Cross_Match` program. Both `Phred` and `Cross_Match` are from the `Phred/Phrap/Consed` package (see *UNITS 11.1 & 11.2*).

1. Distribute the entire set of reads in multiple pairs of base and quality files in FASTA format. For a 300-Mb assembly project, produce tens of pairs of base and quality files such that no file is larger than 30 Mb. For a 3-Gb assembly project, produce hundreds of pairs of base and quality files such that no file is larger than 30 Mb.

   Each read name line begins with the character > followed by the name of the read. There is no space between the character > and the read name. The read name and other information must be separated by at least one space character (not a tab). Base and quality files are named using the PHRAP/CAP3 convention under which, if a base file is named `xyz`*, its quality file is named* `xyz.qual`*. Base and quality files that follow this convention are produced by programs in the popular* `Phred/Phrap/Consed` *package (see UNITS 11.1 & 11.2), in which read sequences are screened for sequencing vectors with* `Cross_Match`*.*

   For ease of use when creating the `.con` *file of read-pair constraints, it is suggested that all reads from a particular clone library be placed into a single pair of base and quality files. This ensures that all read pairs in a file will be the same expected distance apart. Do not mix reads from different libraries.*

2. Compress all base and quality files with the `gzip` utility before feeding them to `PCAP`.

   `gzip` is almost always preinstalled on Unix or Linux systems, but, if necessary, one can obtain it from http://www.gzip.org. It is necessary to place the compressed base and quality files into a single subdirectory where they can be read by `PCAP`.

3. After having prepared the directory of compressed base and quality files, create a "base name" file (equivalent to the `fofn` file of Basic Protocol 1) that contains the names of each of the read files without their `.qual` or `.gz` extensions.

   For example, if there are FASTA and quality files named `abc.gz`, `abc.qual.gz`, `def.gz`, `def.qual.gz`, `xyz.gz`, *and* `xyz.qual.gz`, *and these files all reside in the current directory, the base name file will contain the lines:*

```
abc
def
xyz
```

   Remember that each base and quality file may contain multiple reads, so this file does not contain one line for each read. The name of this base name file becomes the first command-line argument passed to `autopcap`.

4. Create a file that describes the read pairs using the name of the base name file plus the extension `.con` (for "constraint"). If the base name file is named `fofn`, then the file of read pairs has to be named `fofn.con`. Each line of the `.con` file (constraint file) specifies one forward-reverse read pair of the form:

   Column1: `ReadA`
   Column2: `ReadB`
   Column3: `MinDistance`
   Column4: `MaxDistance`
   Column5: `Template`

   where `ReadA` and `ReadB` are the names of the two reads in the pair, `MinDistance` and `MaxDistance` are the estimated minimum and maximum distances between the reads in bp, and `Template` is the name of the subclone from which `ReadA` and `ReadB` were derived. The names and distances on each line are separated by white space (blanks or tabs). All distances are positive, and `MinDistance` is less than `MaxDistance`. The `MinDistance` and `MaxDistance` values should be derived from knowledge of the average insert size of the plasmids from which the pairs are derived; one should be liberal in one's estimates, as a range that is too narrow will adversely affect the results.

```
SubE11.b1  SubE11.g1  1000  6000  SubE11
SubE11.b2  SubE11.g1  1000  6000  SubE11
SubE11.b1  SubE11.g2  1000  6000  SubE11
SubE11.b2  SubE11.g2  1000  6000  SubE11
SubE11.b1  SubE11.g3  1000  6000  SubE11
SubE11.b2  SubE11.g3  1000  6000  SubE11
SubA23.b   SubA23.g   30000 50000 SubA23
```

Figure 11.3.11 Specification of read pairs in the `.con` file when the same subclone is sequenced multiple times.

Note that if the same subclone is sequenced multiple times, each possible combination of right and left end reads must be present in `.con` file. For example, consider a subclone named SubE11 of size between 1,000 to 6,000 bp and a subclone named SubA23 of size between 30,000 to 50,000 bp. If one end of SubE11 is sequenced twice to produce two reads named SubE11.b1 and SubE11.b2, and the other end of SubE11 is sequenced three times to produce three reads SubE11.g1, SubE11.g2, and SubE11.g3, then corresponding read pairs in the `.con` file must be specified as shown in Figure 11.3.11.

GENERATING THE `fofn.con` FILE

SUPPORT PROTOCOL 3

The protocol below provides instructions for generating the `fofn.con` file with the `formcon2` program in the PCAP package. The reads in the example included in the PCAP package are named using the WashU GSC naming convention in which `b` and `g` indicate forward and reverse end reads respectively and the subsequent numbers indicate multiple reads from the same subclone The use of the WashU naming scheme is not mandatory, but if the naming convention is used, then the `formcon2` utility in the PCAP package can be used to create a properly formatted `.con` file.

Necessary Resources

Hardware

Unix or Linux computer with 3 Gb of memory and 15 Gb of free hard disk space. Large assembly projects require more memory and disk space.

Software

Versions of PCAP binary code for different types of Unix and Linux computer systems are available at *http://seq.cs.iastate.edu*. PCAP is free to academic users, but a licensing agreement is required for commercial users. If PCAP requires more than 3 Gb of memory, then the 64-bit version of PCAP should be obtained.

Files

A small example data set is included in the PCAP package. A data set for input to PCAP consists of base and quality files in FASTA format. The files can be generated by the `Phred` program and screened for sequencing vectors by the `Cross_Match` program. Both `Phred` and `Cross_Match` are from the `Phred/Phrap/Consed` package (see *UNITS 11.1 &11.2*). If the reads in the data set follow the WashU GSC naming convention, one can use the `formcon2` program to generate a properly formated `.con` file for one's data set.

1. Enter the PCAP distribution directory, and make a copy of the example data set by typing:

```
cp -r example test
```

2. Enter the newly-created `test` directory, and remove the `fofn.con` file by typing:

```
cd test
rm fofn.con
```

3. Uncompress the two base files by typing:

```
gunzip others.fasta.screen.gz
gunzip plasmid.fasta.screen.gz
```

```
ISB943.y2       ISB943.x1       1000      7000  ISB943
FFC97.y1        FFC97.x1        1000      7000  FFC97
FFC139.y1       FFC139.x1       1000      7000  FFC139
FFC1964.y1      FFC1964.x1      1000      7000  FFC1964
ISB1666.y1      ISB1666.x1      1000      7000  ISB1666
ISB757.x1       ISB757.y1       1000      7000  ISB757
ISB2267.y2      ISB2267.x1      1000      7000  ISB2267
FFC1023.y1      FFC1023.x1      1000      7000  FFC1023
ISB764.x1       ISB764.y1       1000      7000  ISB764
ISB1475.x1      ISB1475.y1      1000      7000  ISB1475
```

Figure 11.3.12 The top part of the `fofn.con` file for the example data set.

4. Generate a `.con` file from each base file by using the `formcon2` program:

   ```
   ../formcon2 others.fasta.screen 1000 7000
   ../formcon2 plasmid.fasta.screen 500 6000
   ```

   For the first base file, a minimum distance of 1000 and a maximum distance of 7000 are used. For the second file, 500 and 6000 are used. The `formcon2` *program produces a file of read pairs named* `abc.con` *on a base file named* `abc`*. As described earlier, it is recommended that all reads in a base file come from the same library so that all paired reads in the file will be the same expected distance from apart. A good rule of thumb is that given an average library insert size of X, the minimum distance between read pairs will be 0.7X and the maximum distance will be 1.3X.*

5. Concatenate the two `.con` files and place the results in the file `fofn.con` (Fig. 11.3.12):

   ```
   cat  others.fasta.screen.con plasmid.fasta.screen.con  >
   fofn.con
   ```

6. Compress the two base files by typing:

   ```
   gzip others.fasta.screen
   gzip plasmid.fasta.screen
   ```

   All files must be in the directory from which PCAP will be run and be in the same location as the base name file.

BASIC PROTOCOL 2

GENERATING A LARGE-SCALE ASSEMBLY WITH PCAP USING DISTRIBUTED COMPUTING

This protocol describes how PCAP can be used to generate a genome assembly of gigabasepair (Gb) size, using distributed computing.

The `autopcap` script (Basic Protocol 1) was designed for <300-Mb genome-assembly projects on a computer system with ten processors. The input files are in a common file system accessible by all the processors. While this arrangement is acceptable for running ten PCAP jobs at the same time, an input/output bottleneck would occur if 100 PCAP jobs tried access to the common file system at the same time. To handle a genome-assembly project of Gb size, PCAP requires a distributed cluster of 100 or more computers; the procedure for achieving this is described in this protocol. In addition, PCAP requires a computer with large memory for memory-intensive sequential computations.

Four Perl scripts are used to coordinate and launch computational jobs to the distributed cluster and the large-memory computer. A job for the cluster is defined by a Unix shell script. The script moves input files from the common file system to the local disk of a computational node on the cluster, runs the `pcap` or `bconsen` program on a processor of the node, and moves the output files from the local disk to the common file system. The `sublapjobs` Perl script generates many shell scripts for computing overlaps with the `pcap` code and submits the shell scripts to the cluster for parallel execution of the scripts. The `runtigcode` Perl script runs the `bdocs`, `bclean`, and `bcontig` programs in the proper order on the large-memory computer. The `subsenjobs` Perl script generates many shell scripts for computing consensus sequences with the `bconsen` code and submits the shell scripts to the cluster for parallel execution of the scripts. The `runstatcode` Perl script runs the minor programs in the proper order.

Necessary Resources

Hardware

PCAP requires a distributed cluster of 100 Unix or Linux computers called computational nodes. Each node has one or more 64-bit processors that share 8 Gb of memory and 20 Gb of free local disk space. Computational jobs on the cluster are controlled by a batch job scheduler. Each computational job is defined by a shell script, which is submitted to the scheduler with the `qsub` command. In addition, PCAP requires a 64-bit Unix or Linux computer with 32 Gb of memory. The nodes on the distributed cluster and the large-memory computer have access to a common file system with 1500 Gb of free disk space. The common file system can be mounted by NFS or a similar shared file system.

Software

See Support Protocol 1 for instructions on downloading and installing PCAP. If the distributed cluster and the large-memory computer are different types of computer systems, which require different versions of PCAP binary code, download the `.tar` file of PCAP code for the distributed cluster and the `.tar` file of PCAP code for the large-memory computer. Install the PCAP code for the distributed cluster in a directory accessible to the nodes on the cluster, and install the PCAP code for the large-memory computer in a directory accessible to the computer.

Files

PCAP takes as input a number of pairs of `gzip`-compressed base and quality files in FASTA format, a file of read pairs, and a file of all base file names without the `gz` suffix. See Support Protocol 2 for generation of input files. As explained below, the compressed base/quality files are copied by computational jobs from the common file system to the local disk of computational nodes. It is a good idea to avoid large base/quality files, which take too long to be copied and hence cause jobs to read them at the same time. Use small base/quality files that can be coped to a local directory in a fraction of a second. Create a directory on the common file system. Create a child directory in the directory and place all the input files in the child directory.

1. Place the four Perl scripts (`sublapjobs`, `runtigcode`, `subsenjobs`, and `runstatcode`) of the PCAP code in the parent directory of the common directory containing the input files.

2. Customize the four Perl scripts to the distributed cluster and large-memory computer being used. Each Perl script has an initial section of variable definitions that need to be modified so that the script knows the special arrangements on the particular computer

system being used. The variables and their initial definitions in the `sublapjobs` and `subsenjobs` Perl scripts are explained below. Replace each initial definition in double quotes with the correct definition for the computer system being used.

```
$CodeDirPath = "/home/xqhuang/PCAP";
```

The variable `$CodeDirPath` *holds the path of the directory containing the PCAP code for the distributed cluster. This directory must be accessible to the nodes on the distributed cluster.*

```
$LocDiskDirPath = "/opt/scratch";
```

The variable `$LocDiskDirPath` *holds the path of a directory on the local disk of each node on the distributed cluster. The local directory is used by the* `pcap/bconsen` *program to store temporarily input and output files. As to be explained later, using the local disk on each node reduces an input/output bottleneck on the common file system.*

```
$JobQueue = "qblade";
```

The variable `$JobQueue` *holds the name of a job queue on the distributed cluster. The queue name is to be used by the Perl script in the* `-q` *option of the* `qsub` *command. All shell script jobs are submitted to the queue for execution. To select a proper job queue, see the manual for* `qsub` *on the distributed cluster.*

```
$NodeNoProcNo = "nodes=1:ppn=1";
```

The variable `$NodeNoProcNo` *holds the number of nodes requested by the job and the number of processors requested on each node. The initial definition indicates that one node is requested and one processor on the node is requested. The number of nodes requested by each assembly job should always be one. If there is enough memory on the node for each processor, one processor on the node is requested. Otherwise, all processors on the node should be requested to keep the memory on the node from being allocated to other jobs. The value of* `$NodeNoProcNo` *is to be used by the Perl script in the* `-l` *option of the* `qsub` *command. For more information on the* `-l` *option, see the manual for* `qsub` *on the distributed cluster.*

The initial definition of one variable in the `runtigcode` and `runstatcode` Perl scripts needs to be modified. Replace the initial definition in double quotes with the correct definition for the computer system.

```
$CodeDirPath = "/home/xqhuang/PCAP";
```

The variable `$CodeDirPath` *holds the path of the directory containing the PCAP code for the large-memory computer. This directory must be accessible to the large-memory computer.*

3. Log on to the front end node of the distributed cluster, go to the common input directory, and run the `sublapjobs` script with the base name file (say `largefofn`) as an argument.

```
../sublapjobs largefofn -y 100 > sublap.log &
```

The `-y 100` *option tells the Perl script to submit 100 jobs to the distributed cluster. Each job runs the* `pcap` *program once and performs other operations that the* `pcap` *program needs to run efficiently and successfully. The* `pcap` *program computes overlaps for a subset of reads. The number of jobs should be large enough such that there is a sufficient amount of memory on the node for the* `pcap` *program. The amount of memory required by the* `pcap` *program on a subset of reads is 15 times the size of the subset, where the size of the subset is the total number of raw bases in the whole set divided by the number of jobs. Note that the same number of jobs (*`-y 100`*) must be used for the four Perl scripts.*

The `sublapjobs` *Perl script works in 100 iterations, generating a shell script in a file uniquely named* `lap.$$.*` *in the directory* `/tmp`*, making the shell script executable, and submitting the shell script to the cluster with the* `qsub` *command. The double dollar*

sign (`$$`) denotes the I.D. of the process that executes the `sublapjobs` *Perl script. The number of a shell script file* `lap.$$.*` *is the iteration number minus 1. For example, shell script file* `lap.$$.5` *is generated in iteration 6. Each job is defined by a shell script with the script number as the I.D. of the job.*

The shell script performs the following tasks in order. It goes to the local directory and creates a working directory with a unique name in the local directory. It goes to the working directory and copies the file `largefofn` *from the common input directory to the working directory. It runs the* `pcap` *program in the working directory with the file* `largefofn` *given to* `pcap` *as an argument, the path of the common input directory given to pcap in the* `-r` *option, the number of jobs (100) given to* `pcap` *in the* `-y` *option, and the job I.D. given to* `pcap` *in the* `-z` *option. It copies the output files produced by* `pcap` *to the common input directory. It removes the working directory.*

A `pcap` *job refers to the* `pcap` *program in execution. Since each shell script job runs the* `pcap` *program once, the number of shell script jobs is equal to the number of* `pcap` *jobs and the I.D. of the shell script job is the I.D. of the* `pcap` *job.*

The `pcap` *program uses the number of jobs given in the* `-y` *option and the I.D. of the current job given in the* `-z` *option to determine a unique subset of reads and compute overlaps for the subset of reads. To reduce the input/output bottleneck to the common input directory, the* `pcap` *program copies the pairs of compressed base and quality files one pair at a time from the common input directory to the working directory. The* `pcap` *program uncompresses the current pair of files in the working directory, reads the uncompressed files in the working directory, and removes them from the working directory when done. At any time, only one pair of base/quality files are kept in the working directory on the local disk, so the availability of local disk space should not be an issue.*

4. Wait until all the shell script jobs submitted by the Perl script in the last step are successfully completed. For example, when shell script job 3 is successfully completed, the file `largefofn.pcap.info3` should be in the common input directory and the last line in the file should read:

   ```
   The pcap job is completed.
   ```

   One can proceed to the next step only after all the jobs are successfully completed. This is because the script in the next step uses the output files from each job. Depending on the size of the input files and the speed of the distributed cluster, the jobs may take 10 to 50 hr to complete.

5. Log on to the large-memory computer and go to the common input directory. Run the `runtigcode` Perl script with the file `largefofn` as an argument.

   ```
   ../runtigcode largefofn -y 100 > runtig.log &
   ```

   The `runtigcode` *Perl script runs, on the large-memory computer, the* `bdocs` *and* `bclean` *programs to remove low-quality overlaps, and the* `bcontig` *program to produce the layouts of contigs and supercontigs. The supercontigs are distributed in 100 files named* `largefofn.pcap.scaffold*` *. The supercontigs in a file* `largefofn.pcap.scaffold*` *are to be used by the* `bconsen` *job of the same number. For example, the file* `largefofn.pcap.scaffold5` *corresponds to* `bconsen` *job 5.*

6. Wait until the script is completed. When it is done, the last line in the file `largefofn.pcap.scaffold.info` should read:

   ```
   The bcontig job is completed.
   ```

   The script may take 5 to 20 hr to complete.

7. Log on to the front-end node of the distributed cluster, go to the common input directory, and run the `subsenjobs` script with the file `largefofn` as an argument.

   ```
   ../subsenjobs largefofn -y 100 > subsen.log &
   ```

The `-y 100` *option tells the Perl script to submit 100 jobs to the distributed cluster. Each job runs the* `bconsen` *program once and performs other operations that the* `bconsen` *program needs to run efficiently and successfully. In addition to the file* `largefofn`*,* `bconsen` *job* `$k` *needs to use the file* `largefofn.con` *and the file* `largefofn.pcap.scaffold$k` *for any* `$k` *between 0 and 99. Thus shell script* `$k` *produced by* `subsenjobs` *copies the file* `largefofn.con` *and the file* `largefofn.pcap.scaffold$k` *to the working directory for* `bconsen` *job* `$k`*. The rest of the* `subsenjobs` *Perl script is similar in structure to the* `sublapjobs` *Perl script.*

8. Wait until all the jobs submitted by the Perl script above are successfully completed. For example, when shell script job 5 is successfully completed, the file `largefofn.pcap.consen.pros5` should be in the common input directory and the last line in the file should read:

```
The bconsen job is completed.
```

The jobs may take 5 to 30 hr to complete.

9. Log on to the large-memory computer and go to the common input directory. Run the `runstatcode` Perl script with the file `largefofn` as an argument.

```
../runstatcode largefofn -y 100 > runstat.log &
```

The `runstatcode` *Perl script runs on the large-memory computer; the minor programs to format the assembly and collect statistics on it. When the script is successfully completed, the last line in the file* `runstat.log` *should read:*

```
The runstatcode task is completed
```

GUIDELINES FOR UNDERSTANDING RESULTS

An assembly is evaluated in terms of continuity and accuracy. The continuity level of the assembly is indicated by the N_{50} contig and supercontig lengths in the `.pcap.n50` file, and the percentage of reads in the assembly. The percentage of reads in the assembly is computed by counting the number of reads in the `reads.placed` file and the number of reads in the `reads.unplaced` file. For a normal assembly produced on a data set of five-fold coverage, the N_{50} contig length is at least 10 kb, the N_{50} supercontig length is at least 1 Mb, and the percentage of reads in the assembly is at least 70%.

The accuracy of the assembly at the read level may be indicated by the percentage of read pairs that are satisfied in both orientation and distance, which is reported at the end of the `.pcap.results` file. For a normal assembly produced on a data set of five-fold coverage, the percentage of read pairs that are satisfied in orientation and distance is at least 40%. The accuracy of the assembly at the base level may be indicated by the consensus base quality scores in the `contigs.quals` file. The accuracy of the assembly at the read and base levels may be evaluated by comparing the consensus sequences in the `contigs.bases` file with the finished sequence of a closely related genome if it is available. The comparison can be done with the BLAT program (Kent, 2002; also see *UNIT 1.4*).

If contigs from different supercontigs are linked by a sufficient number of read pairs (say at least 5), then those contigs may be misassembled. Those contigs are reported in the `readpairs.contigs` file. The read pairs are reported in the `readpairs.reads` file.

The polymorphism rate of the genome is estimated by dividing the total number of consensus bases in the `.pcap.n50` file by the total number of SP lines in the `.contigs*.snp` files. For an inbred strain, a rate less than one SP line in 1000 bp is common.

The alignment of reads in every contig can be viewed in Consed with the `-nophd` option (see *UNIT 11.2*). The reads in the contig should be well aligned. Consed has features to locate the regions of contigs with low quality scores.

The quality of the assembly is affected by a number of factors. If the genome has a very high percentage of highly repetitive elements, a high polymorphism rate, whole-chromosome duplications, or a very low coverage by the data set, then the genome is difficult to assemble and the N_{50} contig and supercontig lengths on the assembly are very low. If a serious error occurs in the assembly process, then the N_{50} contig and supercontig lengths of the assembly are also very low.

Very low N_{50} contig and supercontig lengths indicate that the assembly as a whole has gone poorly. The following steps may be useful for finding the cause for the poor assembly.

First, examine the second last line of the `.pcap.bform.info` file with the `tail` command. The total number of reads in the input data set is reported on the line. If the total number of reads is much lower than expected, then many reads may have been left out of the input files in the common input directory. Otherwise, examine the last line of the `.pcap.docs.info0` file with the `tail` command. The total number of overlaps computed by all the `pcap` jobs is reported on the first part of the line. If the total number of overlaps is much lower than the average depth of coverage times the total number of reads in the input data set, then an error may have occurred in the overlap computation by the `pcap` jobs. Otherwise, examine the last line of the `pcap.clean.info` file with the `tail` command. The total number of unique overlaps, which are not over any repetitive reads, is reported on this line. If the number of unique overlaps is much lower than the total number of overlaps, then the cause for the poor assembly may be that the genome has a very high percentage of highly repetitive elements. Otherwise, use the `grep` command on the `.pcap.scaffold.info` file to locate a line containing the phrase `No. of good overlaps.` The number on this line is the number of good overlaps that remain after low-quality unique overlaps are removed. If the number of good overlaps is much lower, then the genome may be difficult to assemble.

COMMENTARY

Background Information

The PCAP program and its predecessor CAP3 (Huang and Madan, 1999) were developed and improved over a cumulative period of 10 years. The programs are based on three computational techniques. A fast comparison method (Pearson and Lipman, 1988; Altschul et al., 1990) is used to quickly find pairs of reads with a potential overlap. Dynamic programming methods (Needleman and Wunsch, 1970; Smith and Waterman, 1981) are used to compute overlaps between reads and to construct alignments of reads in contigs. A maximum-weight spanning tree method (Kruskal, 1956) is used to construct contigs and supercontigs. The PCAP program has undergone many iterations of improvements based on feedback from its users. The PCAP program has a number of features to address common problems in reads. The PCAP program is easy to run and portable to Unix and Linux computers.

The PCAP program works in three major phases. In phase 1, the whole set of reads is partitioned into subsets of similar sizes. Each subset is compared with the whole set to compute overlaps between reads in the subset and reads in the whole set. The comparisons for the subsets are performed in parallel. The pairs of reads with two close word matches of 12 bp are quickly located. For each pair of reads, an overlap between the reads is computed by a banded dynamic programming algorithm. A region of a read is identified to be highly repetitive if it occurs in many overlaps. Overlaps involving only highly repetitive regions are removed. The remaining overlaps are called unique overlaps.

In phase 2, poor ends of each read are determined and removed based on unique overlaps.

Unique overlaps are ranked in decreasing order of overlap strength. The strength of an overlap depends on the similarity level of the overlap and the depths of coverage for the read positions in the overlap. The overlaps of strength greater than a cutoff are called good overlaps. Reads are assembled into contigs by processing the good overlaps in the decreasing order. Corrections to contigs are made based on read pairs. Corrections include breaking a contig in the middle and joining broken contig pieces.

Links of read pairs between contigs are ranked in decreasing order of link strength. Contigs are connected into supercontigs by processing the links in the decreasing order. Corrections to supercontigs are made based on read pairs. The supercontigs are arranged in decreasing order of size, which are named `Supercontig0`, `Supercontig1`, `Supercontig2`, and so forth. Then the supercontigs are partitioned into groups, where a group consists of supercontigs with the same remainder after dividing their supercontig numbers by the number of groups. This partition ensures that the groups are balanced in supercontig size.

In phase 3, consensus sequences for the supercontigs in each group are computed. The computations for the groups of supercontigs are performed in parallel with each group of supercontigs assigned to one processor. The supercontigs in a group are considered one at a time. For the current supercontig, attempts are made to close gaps between contigs in the supercontig, with repetitive reads that are linked by read pairs to the supercontig. The resulting contigs in the supercontig are considered one at a time. For the current contig, a multiple alignment of reads in the contig is constructed and a consensus sequence is generated from the alignment. Read base quality scores are used in the computation of multiple alignments and generation of consensus sequences.

Critical Parameters and Troubleshooting

If the `autopcap` job fails to produce an assembly on the example data set, double check to make that the right version of PCAP for the particular computer system is installed in the parent directory of the directory containing the example data set and that the example data set is there. In general, the `autopcap` job may fail for a number of reasons: missing input files, input files in incorrect format, insufficient memory, insufficient disk space, incorrect use of PCAP, and unknown bugs in PCAP.

Most common problems occur in the input files, e.g., the base file and quality file are not consistent, a read occurs in one file, but not in the other file, the reads in the two files do not occur in the same order, The name and the description of a read are separated by other white spaces instead of blank spaces. The base and quality files are not compressed, or compressed with another program instead of `gzip`.

If a very poor assembly is produced by PCAP on one's data set, one may run PCAP on an example data set of size similar to the size of the data set. If the assembly on the example data set is much better than the assembly on one's data set, then the poor assembly may be due to that data set.

The number of subtasks given in the `-y` option of each of the five Perl scripts must not exceed the number of supercontigs in the resulting assembly. Otherwise, some group of supercontigs would be empty and the `bconsen` program would complain that it can not open `.pcap.scaffold*` files.

If an error occurs during execution of a shell script that is submitted by `sublapjobs` to the distributed cluster. The error message from the cluster is reported in a file named `lap.$$.$sid.err` in the `/tmp` directory, where the notation `$$` denotes the I.D. of the `sublapjobs` process and `$sid` is the I.D. of the shell script job. The error message from the cluster for `subsenjobs` is reported in a file named `sen.$$.$sid.err`. The error message may be useful for finding the cause of the error.

The `-r` option is used only for running `pcap` and `bconsen` programs on a small local disk, where the current directory does not have any base/quality files. The current directory is the one with the file of base file names that is given as an argument to the `pcap/bconsen` program. If the current directory has all the input files, then the `-r` option must not be used. Otherwise, the input files in the current directory would be removed by the jobs.

A `pcap` job is the `pcap` program in execution. If many jobs want to copy a file from a global directory to a local directory at the same time, then the system may not be able to execute all the copy operations successfully. As a result, the file is not copied to the local directory for some jobs. A change is made to `pcap` and `bconsen` to deal with this problem. If a `cp` command is not successful for a `pcap/bconsen` job, then the job will sleep for up to 16 sec and then attempt `cp` again. A

`pcap/bconsen` job will attempt `cp` at most 11 times. The job will quit if all 11 attempts fail.

Suggestions for Further Analysis

Other Assembly Programs

A number of whole-genome assembly programs have been developed: Celera Assembler (Myers et al., 2000), JAZZ (Aparicio et al., 2002), Arachne (Jaffe et al., 2003), PHUSION (Mullikin and Ning, 2003), PCAP (Huang et al., 2003), and Atlas (Havlak et al., 2004). Those programs have been used in a number of genome assembly projects. Efforts are under way to determine the strengths and weaknesses of the programs. It is likely that some programs are better than other programs for certain assembly projects. However, no program is perfect. Continued improvements to the programs are necessary to meet the needs of genome assembly projects.

The authors of this unit are developing Perl scripts for generating input files for PCAP from files in the NCBI trace archive.

Acknowledgements

The authors of this unit wish to thank Asif Chinwalla, LaDeana Hillier, Pat Minx, and Rick Wilson of Genome Sequencing Center, Washington University Medical School, for suggestions. The authors also thank Liang Ye for the initial typesetting of this unit.

Literature Cited

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1990. Basic local alignment search tool. *J. Mol. Biol.* 215:403-410.

Aparicio, S., Chapman, J., Stupka, E., Putnam, N., Chia, J.M., Dehal, P., Christoffels, A., Rash, S., Hoon, S., Smit, A.F., Gelpke, M.D., Roach, J., Oh, T., Ho, I.Y., Wong, M., Detter, C., Verhoef, F., Predki, P., Tay, A., Lucas, S., Richardson, P., Smith, S.F., Clark, M.S., Edwards, Y.J., Doggett, N., Zharkikh, A., Tavtigian, S.V., Pruss, D., Barnstead, M., Evans, C., Baden, H., Powell, J., Glusman, G., Rowen, L., Hood, L., Tan, Y.H., Elgar, G., Hawkins, T., Venkatesh, B., Rokhsar, D., and Brenner, S. 2002. Whole-genome shotgun assembly and analysis of the genome of *Fugu rubripes*. *Science* 297:1301-1310.

Havlak, P., Chen, R., Durbin, K.J., Egan, A., Ren, Y., Song, X.-Z., Weinstock, G.M., and Gibbs, R. 2004. The Atlas genome assembly system. *Genome Res.* 14:721-732.

Huang, X. and Madan, A. 1999. CAP3: A DNA sequence assembly program. *Genome Res.* 9:868-877.

Huang, X., Wang, J., Aluru, S., Yang, S.-P., and Hillier, L. 2003. PCAP: A whole-genome assembly program. *Genome Res.* 13:2164-2170.

Jaffe, D.B., Butler, J., Gnerre, S., Mauceli, E., Lindblad-Toh, K., Mesirov, J.P., Zody, M.C. and Lander, E.S. 2003. Whole-genome sequence assembly for mammalian genomes: ARACHNE 2. *Genome Res.* 13:91-96.

Kent, W.J. 2002. BLAT: The BLAST-like alignment tool. *Genome Res.* 12:656-664.

Kruskal, J.B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.* 7:48-50.

Mullikin, J.C. and Ning, Z. 2003. The Phusion assembler. *Genome Res.* 13:81-90.

Myers, E.W., Sutton, G.G., Delcher, A.L., Dew, I.M., Fasulo, D.P., Flanigan, M.J., Kravitz, S.A., Mobarry, C.M., Reinert, K.H., Remington, K.A., Anson, E.L., Bolanos, R.A., Chou, H.H., Jordan, C.M., Halpern, A.L., Lonardi, S., Beasley, E.M., Brandon, R.C., Chen, L., Dunn, P.J., Lai, Z., Liang, Y., Nusskern, D.R., Zhan, M., Zhang, Q., Zheng, X., Rubin, G.M., Adams, M.D., and Venter, J.C. 2000. A whole-genome assembly of *Drosophila*. *Science* 287:2196-2204.

Needleman, S.B. and Wunsch, C.D. 1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.* 48:443-453.

Pearson, W.R. and Lipman, D. 1988. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. U.S.A.* 85:2444-2448.

Smith, T.F. and Waterman, M.S. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* 147:195-197.

Key References

Huang et al., 2003. See above.

This article describes the methods used in PCAP in detail.

Internet Resources

http://seq.cs.iastate.edu

This site contains documentation on PCAP and example test data sets.

Contributed by Xiaoqiu Huang
Iowa State University
Ames, Iowa

Shiaw-Pyng Yang
Washington University Medical School
St. Louis, Missouri