Running SNAP

The SNAP Team

February 2012

1 Introduction

SNAP is a tool that is intended to serve as the read aligner in a gene sequencing pipeline. Its theory of operation is described in Faster and More Accurate Sequence Alignment with SNAP. This document is intended to describe how to run the tool, rather than being an in-depth description of how it works.

SNAP is a short read aligner. Its task is to take reads generated by a shotgun sequencer and a reference genome and determine where the reads best match against the genome. SNAP works by building an index of the reference genome, and then using that index in the sequencing process. Typically, you'll build the index once and then read it in each time you want to align a set of reads. Generating the index is relatively expensive, and the content of the index varies only with the set of reads and some parameters related to the index not with the reference genome.

SNAP requires a capable machine in order to run alignments quickly. If you're sequencing reads from a full human genome it requires at least 48GB of memory to hold the index, but 64GB works better. It runs alignments in parallel, so having more cores will reduce the time to complete an alignment. It does not speed up perfectly due to contention for memory and IO resources, but it still benefits greatly. We measured a speedup of 20x on a 32 core AMD-based machine.

2 Building an Index

The first task you'll have to do when you get SNAP is to build an index of your reference genome. This section describes that process and gives guidance as to how to select the parameters.

SNAP builds indices with the `snap index` command, which takes a FASTA file for the reference genome (e.g. `hg19.fa`). SNAP stores its index in a directory (which you need to create manually before running the tool), and when you're using it to align reads it starts by reading the index. Indices run from 35-50 gigabytes in size for the full human genome depending on the parameters you select. Indices will be correspondingly smaller for subsets of the genome.

The index build process uses more memory than running an alignment. If you're running on a machine with constrained memory (say 48GB rather than 64GB or more) you may find that index building forces the machine to page its virtual memory. This will result in an enormous slowdown, especially if your machine uses a hard disk (as opposed to a solid state drive) for its pagefile. It may finish eventually, or you may do better to find a bigger machine for the index build and then just copy the resulting index to the machine where you run your alignments.

Building an index for `hg19` took a little less than half an hour on our machine. The index build process is serial, so unlike alignment it does not benefit from multi-processor machines.

2.1 Seed Length

You need to understand a little about how SNAP works in order to grasp the meanings of the parameters used in building the index. SNAP's basic data structure is a hash table, which allows very quickly looking up sequences of bases that exactly match a given fragment of a read. SNAP builds its index by taking every location in the reference genome and making an entry in the hash table for the sequence of bases at each location in the reference genome. The length of the sequence it uses is called the *seed length*, and is a parameter that you must specify when building the index. For example, chromosome 17 in the hg19 reference genome begins with:

```
          1         2
012345678901234567890123
AAGCTTCTCACCCTGTTCCTGCATA
```

If you build an index with seed size 20, then the index will contain:

```
          1         2
012345678901234567890123
AAGCTTCTCACCCTGTTCCT         - chromosome 17, location 0
 AGCTTCTCACCCTGTTCCTG        - chromosome 17, location 1
  GCTTCTCACCCTGTTCCTGC       - chromosome 17, location 2
   CTTCTCACCCTGTTCCTGCA      - chromosome 17, location 3
    TTCTCACCCTGTTCCTGCAT     - chromosome 17, location 4
     TCTCACCCTGTTCCTGCATA    - chromosome 17, location 5
```

This will allow SNAP to look up `CTTCTCACCCTGTTCCTGCA` and see that it's chromosome 17, location 3. However, if you were to look for `CTTCTCAACCTGTTCCTGCA` (the same sequence with one base changed) the index would find nothing; it has no way of telling that it's close. In order for SNAP to align a read, the read has to contain a sequence as long as the seed size that matches exactly in the reference genome. If a read has some differences from the reference, either because of variants in the sampled genome or errors in the read generation process, larger seed sizes decrease the probability that there will be any perfectly matching places in the index.

On the other hand, short seed sizes result in more seeds that matching multiple places in the reference. Imagine that in our example we had chosen a seed size of 3. The seed TTC occurs at locations 4 and 15. Seeds with multiple hits cause SNAP to have to explore more candidate locations before deciding the best match for a read, and if there are too many hits on a seed SNAP will simply ignore it. So, shorter seeds can both reduce performance and increase the likelihood of missing locations in the genome because of the too-common cutoff.

The best choice of seed size depends on the characteristics of the reads you're using. Shorter, lower quality reads need shorter seed sizes, while longer, more accurate reads perform better (in speed, coverage and error rate) with longer seeds. SNAP can handle seeds up to size 23, with 20 being a good default for ~100 base reads with error rates around 2%. Probably the best idea is to experiment with different seeds sizes for your application and see what works best; the cost to generate a new index or to run test alignments is fairly low. If you're using only part of the human genome, or the genome of a smaller organism, you may want to use smaller seed lengths because there may be fewer seeds that occur too many times in the genome.

SNAP's hash table uses 32-bit values to represent the location in the genome that a seed occurs. It stores values larger than the size of the reference genome in the hash table's location field to indicate that a seed occurs more than one time in the reference. The human genome is about 3 billion bases in length, and it's possible to represent 2^{32}, or just fewer than 4.3 billion different values in 32 bits. This means that if more than 1.3 billion seeds occur in more than one place in the genome SNAP will not be able to represent it in its index. For the `hg19` reference human genome, this means that seed sizes less than 17 will not work; trying to generate an index with too small a seed size will result in an error.

SNAP will never be able to build an index for an organism with a genome with more than 2^{32} bases, and in practice it will have trouble with genomes much bigger than 3 billion because of hash table collisions, though you may be able to get it to work by trying large seed sizes.

2.2 Table Size Bias

A hash table needs to contain a key and a value in its entries. For SNAP, the key is the seed. Since there are four bases in DNA, representing a single base takes two bits. So, representing a seed takes two times the seed length bits. In order to save space, SNAP only stores the last 16 bases (= 32 bits) of the seed in the hash table. It deals with seeds longer than 16 bases by having multiple hash tables, one for each possible prefix of bases. So, if you build an index for length 17 seeds, internally SNAP builds four hash tables: one for seeds starting with A, and also ones for T, C, and G. For length 17 seeds, it builds 16 tables, one for AA, AC, AG, AT, TA, etc.

The hash table design that SNAP uses is very efficient, both in space and access time. As a consequence (and unlike most modern hash tables) it needs to know the table's size when it's first created. Because not all sequences of bases occur equally frequently in the genome, the hash tables need to differ in size depending on the relative frequency of the seed prefix they represent. In the case of the `hg19` reference genome, the amount of bias needed is built directly into SNAP. For other reference genomes (or even for individual chromosomes within `hg19`) the built-in tables may not be correct. You can compensate for this by increasing the hash table slack (see the next section) or simply by having SNAP compute the correct set of biases by making a pass through the reference genome (the better solution). To tell SNAP to compute the correct bias table, include the `-c` command line option.

2.3 Slack

The only other parameter necessary for building the index is the hash table slack. In almost all cases, you will not need to change this from its default value, so you can skip this section.

The size of the SNAP hash tables is determined by the number of entries in the table (which in turn depends on the size and content of the reference genome and the table bias, see the previous section) and also the slack. Slack represents entries in the table that remain empty after the full index is loaded. Increasing slack increases the size of the table, and so the memory footprint of SNAP. To a point, increasing slack can improve performance a small amount, but this quickly reaches diminishing returns, and actually has a small negative effect after a while (which is due to memory cache issues that are beyond the scope of this document).

SNAP's default slack parameter is 0.3, giving .3 of an empty entry for every used one (or equivalently about a 77% table loading). If you're very short on memory you might reduce the slack value. There's probably little reason to increase it. If you get a message saying that the index build failed and you

should increase slack, it's nearly always because of incorrect table sizing bias and the better solution is to specify `-c` rather than changing the slack. See the previous section for a discussion of table size bias.

3 Aligning Reads

Once you have built an index, you will want to use SNAP to align reads. Doing this is fairly simple: SNAP consumes FASTQ files and produces its results in a set of SAM files (one for each processor used by SNAP). In its simplest version, for single-end (unpaired) alignment, you run `snap single` with the name of the index directory you've built and the input FASTQ file, and specify the output SAM file with `-o`. SNAP will load the index and then run alignment and put the result in a set of SAM files, one for each processor. For paired-end alignment, there's a similar command, `snap paired`, that takes two FASTQ files.

SNAP runs on only one processor core by default. While this is friendly for other users of a shared machine, it's bad for performance. See section 3.2 for instructions on how to run on more cores.

3.1 Alignment Parameters

SNAP has a set of parameters that control its alignment process that may be specified on the command line. These control how the alignment works along with the seed size chosen at index build time.

The most important by far of the parameters is MaxHits, both in terms of effect on the quality of the result and also on the performance. You'll probably need to adjust MaxDist and MaxSeeds if you use reads much longer than 125-150 bases. The default parameters are chosen for 100 bp reads assuming a 2% sequencing error rate.

3.1.1 ConfDiff

SNAP works by finding candidate locations in the genome for each read, computing the distance between the read and the candidate location, and then picking the best location that it found for the read. However, there are many parts of the genome that differ by very little and in order to avoid picking the wrong one due to an error in read generation moving the read closer to the wrong place in the genome, SNAP supports the idea of a *confident difference*, represented by the ConfDiff parameter. Simply stated, for SNAP to believe it has correctly found an alignment it must not have found any other alignment with an edit distance less than ConfDiff more than the candidate alignment. Increasing ConfDiff reduces the fraction of reads that are confidently aligned, but also reduces the rate of incorrectly aligned reads (as measured by using simulated reads for which we know the correct alignment).

Consider the following example. Say SNAP's trying to align the read `AAGCT` (the first five bases of chromosome 17) and SNAP finds an exact match for it (at chromosome 17, offset 0) but also finds `AATCT` at some other location. The correct match has edit distance 0, and the second one has edit distance 1 (because it would take one single base change, insertion or deletion to change the read into the second match). If ConfDiff is 1, then SNAP is confident of differences of size 1 and will say that it's matched on chromosome 17, offset 0. If ConfDiff is 2 or greater, then it will say that the read is ambiguous.

For obvious reasons, ConfDiff must be at least 1. The default is 2, and you can change it with the `-c` flag.

3.1.2 MaxHits

There are patterns that occur many times in the human genome, some as many as tens of thousands of times. Searching for hits with seeds that match one of these patterns is often very wasteful of time, and often a read will contain some seeds that hits in these repetitive regions while other parts of the read are less ambiguous, so it makes sense to narrow down the search based on the more rare part of the read.

SNAP uses MaxHits to determine how many hits are too many. SNAP will (mostly) ignore seeds that more hits than this (but see the section on adaptive ConfDiff), and pretend that it never used that seed.

Lower MaxHits results in better performance, but also misses some possible alignments, and also misses some cases where there an alignment is ambiguous (because of the possible candidates was never considered because all of the seed matches that pointed to it were excluded as being too popular). Thus, decreasing MaxHits can increase the error rate.

The default MaxHits works well for 100 bp reads, but you can make it significantly lower (e.g. as low as 100) for longer reads and longer seed lengths. The command-line parameter for it is `-h`.

3.1.3 MaxDist

MaxDist is the maximum edit distance that SNAP will ever tolerate between a read and a candidate match in the reference genome. Increasing MaxDist generally decreases performance, because SNAP may have to do more work to compute the precise distance to each match candidate it finds, but increases coverage. Its default is 8, which works fine for reads around 100 bases in length and an error rate around .02. If you've got longer reads or a higher error rate (or both), you'll have to increase MaxDist accordingly. If you've got short, accurate reads you may increase performance somewhat by decreasing MaxDist, but it probably won't help that much. The command line flag for setting MaxDist is `-d`.

3.1.4 MaxSeeds

SNAP selects only certain seeds from a read to look up in the index. Often, when there's a good match there's no chance that it will find a better match no matter how long it looks, and so it will quit. In other cases, however, it continues to try various seeds without reaching a conclusive result. In these cases, it tries at most MaxSeeds seeds. Increasing MaxSeeds improves coverage and error rate while reducing performance. You should increase it for higher-error or longer reads. The command line flag for MaxSeeds is `-n`.

3.1.5 Adaptive ConfDiff

We found that in cases where reads have matches with repetitive portions of the genome that they are more likely to be aligned incorrectly. The adaptive ConfDiff parameter takes advantage of this observation by increasing ConfDiff by one for reads that have too many seeds that have more than MaxHits occurrences in the reference genome. The `-a` flag specifies how many seeds have to be ignore before increasing ConfDiff for a read. By default it is infinite (i.e., ConfDiff is static).

Decreasing the adaptive ConfDiff threshold will reduce percent of incorrectly aligned reads while also lowering the overall percent of reads aligned.

3.1.6 Paired-End Read Spacing

For the paired-end version of SNAP (`snap paired` command), the `-s` parameter specifies the minimum and maximum spacing to allow between the two reads in a pair. Setting this closer to the actual lengths

produced during your sequencing process will slightly improve accuracy and speed. The default allows between 100 and 1000 base pairs spacing. Note that this parameter sets the expected spacing between the *start positions* of the reads, **not** the expected length of the whole paired-end string. For example, for a 500 bp fragment of DNA where 100 bp have been read from each end, the spacing is 400, not 500.

3.1.7 Exploring the Parameter Space

To quickly explore the impact of the alignment parameters on SNAP, you can specify a range for any of them. This is done using the syntax `start:end`, in which case SNAP tries all values from start to end, or `start:step:end`, in which case it advances by step instead of by 1. For example, passing `-d 8:12` will try all MaxDist values from 8 to 12, while passing `-d 8:2:12` will try only 8, 10 and 12. The only alignment parameter that does not support ranges is `-s` (paired-end spacing).

3.2 Performance Parameters

SNAP has several settings that do not change its alignment decisions, but that may affect performance.

The `-t` option tells SNAP how many threads (processors) to use. To make it friendlier in a shared machine environment by default SNAP only uses one processor. For high-throughput sequencing (or if you're the only user of a machine) you will want to set this to the number of cores on your machine. It is inadvisable to set –t to more than the number of processors available.

The `-b` option tells SNAP to bind each thread to its corresponding processor. This improves performance by around 20%, but works best on a dedicated machine where no other users contend for the processors. Using -b makes most sense when –t is set at or near the number of processors on the machine.

Finally, on Linux systems, you can improve performance by 20-30% by using a kernel with huge memory page support (`MADV_HUGEPAGE`) enabled. Recent CentOS and Ubuntu kernels, among others, support this. SNAP will print a warning if it could *not* allocate huge pages. It uses them by default on Windows, but requires you to take some steps to enable the necessary privileges, and prints instructions about how to do so if necessary (and you need to run with admin privileges enabled as well). On machines with memory that's barely sufficient to hold the index you may see a warning that it is "falling back to VirtualAlloc." This just means that your machine couldn't find enough physically contiguous free page frames to allocate more big pages[1].

3.3 Error Reporting for Simulated Reads

Sometimes it is useful to generate simulated reads to evaluate an aligner. In that case, it's possible to annotate the reads with the location in the genome from which they're drawn, to check whether the aligner got the correct result. SNAP understands the format in the FASTQ id string generated by the SAMtools `wgsim` program. If you give SNAP the `-e` flag it will compute number of errors in the reads it processes and report that at program completion.

3.4 Aligned Output

The `-o` parameter tells SNAP the name of a SAM-format output file. If you do not specify it, SNAP discards its output, and only reports performance results.

[1] Don't worry if you didn't understand that. It just means things will be a bit slower than if you bought more memory.

Note that when running in multi-threaded mode, SNAP creates multiple output files, one per thread. These are given names based on the thread number. For example, if you specify `-o output.sam` with four threads, you will get files `output_00.sam`, `output_01.sam`, `output_02.sam`, and `output_03.sam`.

3.5 Alignment Statistics

When SNAP completes running, it prints some statistics about the alignments. The output looks like this:

```
This is a beta version of SNAP and is provided for non-commercial evaluation purposes only.
It is licensed for use only before January 1, 2013 and only for work that is not related to
any contest or prize.

Loading index from directory... 3s.  51304566 bases, seed size 20
ConfDif MaxHits MaxDist MaxSeed ConfAd  %Used   %Unique %Multi  %!Found %Error  Reads/s
2       200     8       35      4       100.00% 95.99%  3.82%   0.19%   0.017%  20016
```

After the license disclaimer, the next line shows the load time and rate of the index and the seed size of the index. After that line prints, SNAP aligns the reads, and then prints the following statistics:

- The *%Used* column shows how many of the reads passed SNAP's filtering test for low-quality bases. If a read has more than MaxDist 'N' bases, it is ignored because it cannot match any location with fewer than MaxDist errors.
- A *unique* alignment is one in which SNAP found exactly one location in the genome that matched the read at least ConfDiff better than every other read. The %Unique column shows what percentage of the *used* reads were uniquely aligned.
- A *multi* alignment is a case where SNAP found matches, but where there the second best match was less than ConfDiff from the best.
- *Not found* means that SNAP found no matches with an edit distance less than MaxDist.
- *%Error* is computed only for `wgsim`-generated simulated reads if the `-e` flag is set.