Running SNAP

The SNAP Team

September, 2013

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 28x 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, and when you're using it to align reads it starts by reading the index. Indices run from 40-57 gigabytes in size for the full human genome depending on the parameters you select. Indices will be correspondingly smaller for subsets of the genome, or for organisms with smaller genomes.

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. It runs in parallel when possible, but depending on the parameters for the index there may or not be much parallelism available. In any case, it does not get as good scaling as running alignments.

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
01234567890123456789012
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 32, with 23 being a good default for ~100 base reads with error rates around 2%. For longer reads, longer seeds make sense. A good rule of thumb is that the seed size should be less than a quarter of the read size (or smaller for lower quality reads). Our experiments show that for seeds smaller than a quarter of the read size there is little if any effect on alignment quality, but that there can be a noticeable performance improvement for larger seeds.

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.

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 19 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 Key Size

Since there are four DNA bases, it takes two bits to represent each possible base. The SNAP hash table includes a key field that stores the seed in binary form. Therefore 32 bits (4 bytes) of hash table key can represent a seed size of 16 bases. Since 16 bases is too small to represent the human genome (because of the number of duplicates), SNAP compensates for this by using more than one hash table. A seed consists of two portions: some number of bases that specify which hash table to use, and then the remainder that are stored in the hash table key. So, for example, if we had a seed size of 17 and a hash table key size of 32 bits, SNAP would actually keep four separate hash tables, one each for seeds starting with A, C, T and G. As the seed size grows, the number of hash tables also grows with the fourth power of the number of bases beyond the size of the hash table key. After a while, this becomes absurd. For example, for a seed size of 32 bases and a key size of 32 bits this would result in 4 billion hash tables (more than there are bases in the human genome, so many of them would be empty); just the array of pointers to these hash tables would take 32GB.

SNAP handles this problem by allowing you to decide the key size of the hash table key at index generation time. The key size can be any number of bytes from 4 to 8, inclusive. SNAP requires that the seed size be large enough that it uses at least all of the key bits, and small enough that it not result in more than 256K hash tables. In practice, this means that the seed size must be between 4x the key size in bytes and 4x the key size plus 9. So for a 4 byte key size the seed size can be from 16 to 25 (assuming that it doesn't run out of genome address space; key sizes lower than 19 don't work with hg19). For a key size of 5 bytes you can get to 29 bases, and 6 is enough for 32.

Increasing the key size increases the size of the hash table, but does not in any affect the results of an alignment. Therefore, it's generally a good idea to use the smallest key size that works for the seed size you've selected.

2.3 Other Index Parameters

Because many seeds occur more than once in a genome, and because SNAP may use more than one hash table (see section 2.2), it is necessary for SNAP to compute the size of each of its hash tables before loading them. There are three ways to do this. The simplest way is to let SNAP estimate the size of each of its tables. It does this by making a pass through the reference genome and using an approximate

counter algorithm. This counter tends to be a little conservative, so this will sometimes result in larger hash tables than are strictly necessary. If you're building an index for the hg19 reference genome, SNAP comes preloaded with the table size estimates (for all seed size/key size pairs that result in 16K hash tables or fewer, and also for key size 4 bytes and seed size 22 and 23 bases). To use these preloaded estimates, use the –hg19 switch. This will result in both a faster index build and a smaller index. The third method to compute the table sizes is to specify the –exact switch, which will result in a slower, more thorough computation of the number of seeds in each table. This gives the benefits of the –hg19 switch for other reference genomes, but at the cost of some speed and memory footprint (during index build time).

Another parameter used 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 tuning this. 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). 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.

The –O switch controls the size of a data structure used during index construction. If index build fails and tells you to use a larger value for –O, then you should. Otherwise, there is no reason to change it. If an index build succeeds, the resulting index doesn't depend on the value of –O. All that it affects is the memory footprint during index build.

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, SAM, BAM and gzipped FASTQ and SAM files and produces its results in a SAM or BAM file. 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 file, and specify the output file with `-o`. SNAP determines both input and output file types by looking at the end of the filename, so files that end in .BAM are BAM files, etc. SNAP will load the index and then run alignment and put the result in the specified output file. For paired-end alignment, there's a similar command, `snap paired`, that takes two input files.

SNAP runs on all of the processor cores in your machine by default. See section 3.2 for instructions on how to run on fewer 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. For paired-end alignment maxSeeds also has a large effect on performance, but with a smaller effect on accuracy than maxHits.

3.1.1 MaxHits

There are patterns that occur many times in the human genome, some as many as hundreds 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 ignore seeds that have more hits than this, 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.

To change maxHits for single-end alignment, use the –h parameter. Very good quality uses values around 2000, while 300 is the default and as low as 10 can be useful for a quick pass over the data.

For paired-end alignment, there is a similar parameter, -H (with a capital H). The paired-end aligner uses a different algorithm than the single-end aligner. Before computing any scores, it intersects the set of seed hits with one another to find candidates to score. This intersection happens in time that's proportional to the log of the number of seed hits, and is linear in the number of places where the seeds hit next to one another. Because of the log factor, there is less of a performance hit to use much larger values of maxHits. The SNAP default for –H is 16000, and there is a small benefit in turning it up to 300000. It runs very quickly with very small values like 10 or 20.

When the paired-end aligner can't find a place in the genome where both ends of a read align near one another, it tries to align each end independently using the single-end aligner (and applies a penalty to the resulting mapping quality). Therefore, the –h parameter is still meaningful for paired-end alignments, which also explains why there's a different flag value for maxHits for single- and paired- end alignments.

3.1.2 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 (though not all that much), 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`.

3.1.3 MaxSeeds

SNAP selects only certain seeds from a read to look up in the index. Increasing the number of seeds may slightly increase mapping accuracy, particularly in the single-end aligner. In the paired-end aligner, performance is very sensitive to the number of seeds, because the number of sets it needs to intersect depends on the number of seeds.

3.1.4 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.

3.1.5 Multiple Runs

You can make multiple runs of SNAP without exiting the program, and without reloading the index if consecutive runs use the same index. To do this, just separate the command parameters for separate runs by a comma (which itself must have spaces on either size of it, not like you'd use in written text). So, for example, you could do:

```
snap single /indices/hg19-23 InputFile.fq –o OutputFile.bam , paired
/indices/hg19-23 InputFile1.fq InputFile2.fq –o OutputFilePaired.bam
```

You can string together as many alignments as you'd like that way, subject to how long of a command line your operating system is willing to tolerate.

One warning: SNAP doesn't even parse the command line after the comma until it's finished the earlier run(s), so if you make a syntax error you won't find out about it until after the earlier run(s) complete. It can be somewhat depressing to come back in the morning and find out that SNAP aborted with a mistyped parameter after the first of 10 alignments finished.

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 you might want to have it use fewer than the number of available processor cores. It is inadvisable to set –t to more than the number of processors available, though it will still work.

The `-b` option tells SNAP to bind each thread to its corresponding processor. This improves performance somewhat, but works best on a dedicated machine where no other users contend for the processors.

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].

[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.

On Windows machines, allocating huge page memory can be very slow (particularly if the machine hasn't been recently booted). If you specify --hp (note two dashes, think of it as "dash minus hp") it will not use huge pages, which increases index load speed and decreases alignment speed.

3.3 Output

The `-o` parameter tells SNAP the name of a SAM- or BAM- format output file. If you do not specify it, SNAP discards its output, and only reports performance results. SNAP determines whether to generate SAM or BAM format by looking at the filename. If it ends in ".bam" then it makes BAM, otherwise, SAM.

To sort the output, specify –so. –sm tells SNAP how much memory (in gigabytes) to allocate for sorting. Since the output file is likely to be larger than the machine's memory, SNAP uses an offline mergesort. Giving it more memory increases the size of the chunks that get sorted, which reduces the number of chunks that need to be merged and so increases speed.

-M tells snap to use the old form CIGAR strings that use "M" to indicate match. By default, SNAP uses = to indicate an exact sequence match, and X to indicate different bases without an indel.

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.