Applying Cortex to Phase 3 1000Genomes data - the recipe

Zamin Iqbal (zam@well.ox.ac.uk)

21 June 2013 - version 1

Contents

1 Overview **1**

2 People **1**

3 What has changed since version 0 of this document? **1**

4 Running the same pipeline at multiple centres **2**

5 Pipeline Outline **2**

6 Compiling and installing the latest release of Cortex **3**

7 Details of each of the above pipeline steps **3**
7.1 Files you will need to download 3
7.2 Making a list of samples 3
7.3 File naming convention 4
7.4 Building per-sample uncleaned graphs 4
7.5 Building a pool and error-cleaning it 4
7.5.1 In case you completely run out of memory 5
7.5.2 For Phase3 we all apply the same formula to get error cleaning threshold: . 6

1 Overview

This is the specification for how to apply Cortex to build samples and population pools for the 1000 Genomes populations. Since for Phase3 of 1000 Genomes we are not doing variant calling with Cortex, buy instead building per-sample and per-population graphs, this is a much shorter document than the Phase 2 Document/

2 People

We are currently running this collaboration between EBI, NCBI, Einstein and Oxford, and so it is important we all apply exactly the same methodology/pipeline to the data. =

3 What has changed since version 0 of this document?

N/A

4 Running the same pipeline at multiple centres

If we are to generate comparable callsets which really are products of the same pipeline, there are a few things we need to do.

1. Please use a clean install of Cortex **v1.0.5.1**5, now available at

   http://sourceforge.net/projects/cortexassembler/files/cortex_var/previous_releases/CORTEX_release_v1.0.5.15.tgz

Either use the wrapper-scripts I provide, or use exactly the command-lines documented below .

5 Pipeline Outline

The pipeline is as outlined in this section**. I have provided scripts to run all of these steps now**, available in the Cortex release under scripts/1000genomes

Each centre is allocated a set of populations to own. Then, for each population

1. **Build a binary graph file for each sample** (one command-line instruction). This requires ~20-50 GB of RAM (afraid it varies depending on how "dirty" the data is, 30Gb of disk, ~3 hours per sample. At Oxford I ran this on multiple processors on a single 512Gb RAM machine, but you could accelerate it further if you have a cluster with nodes with enough memory. Exact memory requirement per sample depends on depth of coverage; a high coverage (Illumina) sample would top out around 70GB of RAM. These uncleaned graphs are a final product, which will be distributed to the 1000 Genomes project (and public).

2. **Pool the samples**. All of the sample graphs are merged into one pool, which is then error-cleaned, and dumped as a file. For a high diversity (or low quality sequence) population, this would squeeze up against a 256Gb RAM limit. A pragmatic approach is to split the population into two halves, pool and error-clean these, and then merge those cleaned pools. This will halve the memory requirement and avoid going just over the 256 GB RAM limit, although it would mean we could lose some doubletons which had one sample in each half. This is what I did with the Luhya, as 40 samples contained 15 billion kmers (I was using k=55, so needed almost double the memory we need for this pipeline). At the end of this we have an un-error-cleaned pool graph.

3. **Error cleaning** is now done on the pool. I give a formula for how to calculate the cleaning threshold.

In commands, this looks like:

1. perl script1_make_filelist_from_sequence_index.pl –sample NAxxxx (one process per sample)

2. perl script2_build_uncleaned_sample_graph.pl –sample NAxxxx (one process per sample

3. perl script3_build_unclean_pool.pl –list LIST –outdir OUTDIR –pop XYZ –mem_height 28 –mem_width 100 (just one process only, but this is the highest memory step, Ideally, alloc 430Gb RAM. (On a 256 Gb RAM server, you will have to split into two pools, then error clean each one with threshold 1, then combine - details below)

4. Cleaning threshold T = int((total depth of coverage/300) +0.5)

5. cortex_var_31_c1 –kmer_size 31 –mem_height 28 –mem_width 100 –multicolour_bin XYZ_uncleaned_pool.q10.k31.ctx –remove_low_coverage_supernodes T –dump_binary XYZ_cleaned_pool.threshT.k31.ctx >& output_clean_pool (this also requires high memory, but is independent of script4 (could be run at the same time)

6 Compiling and installing the latest release of Cortex

Use a clean install of Cortex version 1.0.5.15. Follow the instructions in the INSTALL file. In essence, you need to

1. Run a bash script, "bash install.sh", which compiles all the auxiliary libraries (only needs to be done once)
2. Compile Cortex itself

Then there are a few things you need to do for the VCF-dumping tools

1. Obtain Stampy from http://www.well.ox.ac.uk/project-stampy. Unzip it somewhere, cd into it and type make. Stampy needs Python version 2.6 or 2.7. Only supports x86_64 and (experimental) Mac. The process_calls.pl script for dumping VCF will need as an argument the path to your Stampy.py.
2. Download this version of VCFTools https://sourceforge.net/projects/vcftools/files/vcftools_0.1.9.tar.gz

7 Details of each of the above pipeline steps

In the following steps, I always use XYZ to refer to the population abbreviation (eg LWK for Luhya).

7.1 Files you will need to download

The following files are all available from here:

ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/working/20120814_cortex_resources/

1. A standard reference genome binary human_g1k_v37.proper_chroms.k31.ctx
2. A Stampy hash of this reference binary, this is two files: human_g1k_v37.sthash, and human_g1k_v37.stidx
3. A common installation of Stampy to avoid version nonsense: stampy-1.0.13.tgz

7.2 Making a list of samples

Parse this 1000 Genomes sequence.index file:

ftp.1000genomes.ebi.ac.uk:/vol1/ftp/sequence_indices/20120522.sequence.index

Pull out those lines with your population identifier in column 10 (counting from 0 - has POPULATION as header). Exclude lines with 1 in column 20 (WITHDRAWN).

Script provided: I have provided a sample script to do just this: it reads the sequence index, and given a sample-id, gets the appropriate files from the index, checks they are present on the filesystem (unzipping them if necessary) and makes the filelists in the format Cortex expects them. Usage

perl script1_make_filelist_from_sequence_index.pl –sample NA19316

7.3 File naming convention

Please can we all use the same convention for file-naming. Use .ctx at the end of a filename to signify a Cortex binary graph file. I suggest, for uncleaned binaries, a format: NAxxxx.k31.uncleaned.q10.ctx, and for cleaned NAxxxx.k31.q10.cleanedT.ctx, where T is the threshold used for cleaning the **pool (**see below**),** and 5 is the quality score threshold used**.**

7.4 Building per-sample uncleaned graphs

The command-line is

```
cortex_var_31_c1 –sample_id NAxxxx –kmer_size 31 –mem_height 25 –
mem_width 150 –se_list FILE_SE –pe_list FILE_PE1,FILE_PE2 –quality_score_threshold
10 –dump_binary sample_id.uncleaned.q10.k31.ctx –remove_pcd_duplicates >&
logfile_sampleid_build_uncleaned_q10
```

FILE_SE1 lists all the single ended fastq **(better/faster if they are zipped)**. FILE_PE1 lists all the _1 paired end fastq (better zipped), etc.

Script provided: I've provided a script that does this - **script2_build_uncleaned_sample_graph.pl**. It takes the sample_id as its only argument - eg

```
perl script2_build_uncleaned_sample_graph.pl –sample NA12878
```

(but has hardcoded inside it the path to the root directory where you want all of this stuff to happen), so you can parallelise many instances of this by just calling this with different sample_id's as arguments. Each instance uses 75GB RAM (can be modified). Time taken to build a graph for one sample depends on speed of access to your disk, but could be anywhere between 2 and 6 hours.

7.5 Building a pool and error-cleaning it

We make a colourlist - a list containing one **tab-separated line with two fields. First, the name of a file listing all the uncleaned binaries. Second, the text that will go into the "sample-id" metadata in the header of this binary. In this case, use XYZ_pool (eg for the Luhya I would use LWK_pool).**

```
>cat colourlist_for_merge
list_all_uncleaned_binaries XYZ_Pool
>cat list_all_uncleaned_binaries
sample_1.uncleaned.q10.k31.ctx
sample_2.uncleaned.q10.k31.ctx
sample_3.uncleaned.q10.k31.ctx
sample_4.uncleaned.q10.k31.ctx
..
sample_80.uncleaned.q10.k31.ctx
>cortex_var_31_c1 –kmer_size 31 –mem_height 27 –mem_width 110 –
colour_list colourlist_for_merge –dump_binary XYZ_uncleaned_pool.q10.k31.ctx
–dump_covg_distribution XYZ_uncleaned_pool.covg.txt >& output_merge_uncleaned
```

The mem height/width arguments I specified above will allow you to go up to 14.7 billion kmers, which is as many as you can fit on a 256Gb RAM server (a cleaned population should have 3-5 billion (eg Luhya is 3.5 billion) , my UNcleaned LWK had to be split into two pools of 15 billion (obviously a lot of overlap, the total is not 30 billion) - so a 256GB server would not have sufficed). If you have a server with at least 512Gb RAM, the **preferred option if just use –mem_height 28 –mem_width 100** , which will support 27 billion kmers, **and use 420Gb of RAM**. This is overkill, but it pretty much guarantees success, and you don't need to split the pool into two etc (see next section).

This is a slow process, which may take up to 48 hours, depending on the load on your disk, and connection between disk and server. We've just added a significant speed improvement to Cortex though, so I'm not sure how long it will be. If CPU is the bottleneck, our performance boost will help, but if IO is the bottleneck, then it won't - depends on your compute infrastructure/usage at the time.

Script provided: I've provided ascript: **script3_build_unclean_pool.pl,**

```
perl script3_build_unclean_pool.pl –list LIST –outdir OUTDIR –pop XYZ
–mem_height 28 –mem_width 100
```

where LIST is a filelist of the uncleaned sample binaries, the OUTDIR is wherever you want to put the merged population binary. The script is very simple, but it will complain if you try to allocated too little memory.

Error Modes: there are two ways in which this script can fail

1. Your server does not have enough memory to give you as much as you have asked for, so right at the start (immediately), it will fail to allocate the memory it needs. You will get a message saying "Giving up - unable to allocate memory for the hash table". If you use script3, that error will be in this file OUTDIR/XYZ_merged_uncleaned_samples.k31.q10.ctx.log

2. Your server gives you as much memory as you have asked for, but it turns out you have asked for too little, and your sequencing data won't fit. You will get a message saying "Dear user - you have not allocated enough memory to contain your sequence data.". If you use script3, that error will be in this file: OUTDIR/XYZ_merged_uncleaned_samples.k31.q10.ctx.log

7.5.1 In case you completely run out of memory

If at all possible on your server, increase mem_height and width in order to be able to merge all the samples into one pool. If that is impossible, then follow the instructions in this section.

If this fails because even the full capacity of a 256Gb RAM server is not enough, then split the samples into two lists, and do this:

```
>cat colourlist_for_merge_first_half
list_first_half_uncleaned_binaries
>cat list_first_half_uncleaned_binaries
sample_1.uncleaned.q10.k31.ctx
sample_2.uncleaned.q10.k31.ctx
sample_3.uncleaned.q10.k31.ctx
sample_4.uncleaned.q10.k31.ctx
..
sample_40.uncleaned.q10.k31.ctx
>cat colourlist_for_merge_second_half
list_second_half_uncleaned_binaries
>cat list_second_half_uncleaned_binaries
sample_41.uncleaned.q10.k31.ctx
sample_42.uncleaned.q10.k31.ctx
sample_43.uncleaned.q10.k31.ctx
...
sample_80.uncleaned.q10.k31.ctx
>cortex_var_31_c1 –kmer_size 31 –mem_height 27 –mem_width 110 –
colour_list colourlist_for_merge_first_half –dump_binary XYZ_first_half_pool.clean1.q10.k31.ctx
–remove_low_coverage_supernodes 1 >& output_merge_first_half
>cortex_var_31_c1 –kmer_size 31 –mem_height 27 –mem_width 110 –
colour_list colourlist_for_merge_second_half –dump_binary XYZ_second_half_pool.clean1.q10.k31.ctx
–remove_low_coverage_supernodes 1 >& output_merge_second_half
```

ie break the pool in two, and do the absolute minimal cleaning, and then merge. So first make a new colourlist:

```
>cat colourlist_for_merging_subpools
list_both_subpools
>cat list_both_subpools
XYZ_first_half_pool.clean1.q10.k31.ctx
XYZ_second_half_pool.clean1.q10.k31.ctx
```

and then merge the two halves:

```
>cortex_var_31_c1 –kmer_size 31 –mem_height 27 –mem_width 110 –
colour_list colourlist_for_merging_subpools –dump_binary XYZ_uncleaned_pool.q10.k31.ctx
–dump_covg_distribution XYZ_uncleaned_pool.covg.txt –remove_low_coverage_supernodes
1 >& output_merge_first_half
```

If we can possibly avoid doing this, it is best to, as it will kill low frequency variants which are split between the two pools.

Now we are ready to choose the proper cleaning threshold for the whole pool.

7.5.2 For Phase3 we all apply the same formula to get error cleaning threshold:

Here's how the choice is made:

If we have a 1% error rate, and depth of coverage D (for example when I built the Luhya, I had 85 individuals at 6x=510x), then our rough expectation is 0.01D errors at a monomorphic site. ie 5.1 errors at every site for the LWK . But there are 3 possible errors at a site. So we use threshold given by

$$\text{Default cleaning threshold T} = \text{int}\left(\left\lceil \frac{\text{TOTAL\_DEPTH\_REPORTED\_BY\_CORTEX}}{300} \right\rceil + 0.5\right)$$

No script provided! This is just a single command-line.