Fast Illumina Genotyping and Other Adventures

By J. Kollenberg

Contents

Chapter 1 - The Goal

Before you begin, I think it is important to understand, in simple terms, the goal of this entire project. For come context, we are trying to sequence individuals' genes. As it stands, Wits uses the products of a company called Illumnia and the process is as follows.

Step 1
A few drops of blood are dropped onto chips. This makes up the actual test. From this step .idat files are created with all the information from the test stored in these large files.

Step 2
The .idat files require some processing and to be turned into .gtc files. This takes quite a bit of time and processing power.

Step 3
The .gtc files need to be turned into a Final Report. This is done by a very slow Python script written by Illumina and the Final Reports are very large files, containing much redundant information.

Step 4
A few columns of the Final Report are taken and transposed to create a .lgen file which can be imported into Plink, the final destination.

The goal of this project is to write a single, super-efficient C script which takes in a .idat file and returns a plink file, in one go. This script can also be custom designed to fit into Wits' pipeline effortlessly.

Well, you may ask, how do we do that?

To be honest, it is not that difficult… Mostly.

The first thing one must appreciate is that python is an incredibly slow language and C is incredibly quick. (To give you a sense of the scale, the official Python takes about 302 seconds to read the .gtc's manifest file. My direct translation to C script takes 12.)
Almost all of the code in the original process is open source. And so, much of this problem simply requires one to sit and translate the Python into C. However, some degree of understanding is required in order to know which functions can be safely omitted when turning the general script into a specialised one.

Chapter 2 - My Progress

We have decided to split up this project into multiple steps. The first big milestone would be completing a C script that take in a folder full of GTC files, and manifest file and creating a .plink file.
All of my work so far has been contained in this step. As it stands, the script reads in all the information, does necessary changes/processors, locates important bits and discards the unimportant bits and attempts to put the data into a .plink 2D array.

The current issues I am struggling with are:

1. The original python script is very versatile, being able to handle as many GTCs as the user would like, with as many people in each GTC as desired. Mine is still trying to perfect single GTC with single person. In future it should be expanded to accommodate the variations.
2. GTCs come in different versions, with version 3,4 and 5 being acceptable. I have been testing my code exclusively with version 5 and support for 3 and 4 may be required.

Chapter 3 - Next steps

First, you would want to complete the first milestone and get the GTC to Plink up perfectly. This should include all the versatility of the original script.

Next, I would assume you would want to extend the code backwards meaning instead of taking GTCs as inputs, take Idats, do all processing that would have created GTCs and use those results for plink. This should not be a difficult task as python for processing Idats should be accessible.

This will not be the end of the project. There has been talk of all the interesting files we can create if we have full control of the pipeline. From extending to VCFs to potentially creating our own file structures built to only support necessary information.
Chapter 4 - Tips

Intricate projects such as this one often seem daunting at first. This is especially true if one is coming in without much C experience. However, I can confidently say that with a little perseverance C comes surprisingly easily, and it is such a useful tool to know. Below is a list of tips that drastically sped up the development process.

Proper Testing

1. Before you begin, I would suggest cloning the BeadPoolArray git into your local repository.
2. Move each of the modules (files in the directory "Modules") into the directory called "Examples".
3. In gtc_final_report.py (or any other example), change the imports to use to local files in the same directory
   I.e. change `from .BeadArrayUtility import read_int, read_string, read_byte`
   to `from BeadArrayUtility import read_int, read_string, read_byte`
4. This allows you to edit the modules and print out intermediate values as opposed to running the official module and only having access to final results.

Function Testing

1. The main file has gotten quite long and can take a bit of time to run. And so, when adding a function that you are not fairly confident will perform as expected, you should test the function in its own script using sample values.
2. This can either be done by creating a separate C file in isolation of the main one. Alternatively, I often use an online C compiler which runs without any set up required. It is particularly good for testing memory management. The link can be found here.

In General

1. Watch your #includes.
   This may seem obvious but can save you a large amount of time.
   When replicating code in another environment make sure to use the proper #includes. And you compile your code, and you suddenly have a bunch of errors in code that used to work, make sure the #includes are spelt correctly and that you have not mistakenly added a letter.

Important emacs commands on Windows

There are so many possible commands, but the following are, in my opinion, the essentials.

Note, when I write “Ctrl s” I mean press (and release) control and then press s.

1. Ctrl s - Save
2. Ctrl x - exit
3. Alt x [type words] goto-line [click enter] 123 - it scrolls down to line 123
4. Alt s w [type a word] - searches for something
5. Control [space] [use arrow keys to select text] - Highlights and selects text
6. Ctrl w - cuts selected text
7. Ctrl y - pastes cut text

Chapter 5 - Contact me

If you have any questions, please feel free to pop me an email at:
jonahkollenberg1@gmail.com