Swift: Primary Data Analysis for Next-gen sequencers

Nava Whiteford, Tom Skelly

October 31, 2008

Chapter 1

Introduction

Swift is an open source primary data analysis tool for next-gen sequence data, it is released under the LGPL3 licence. Currently we are focused on Illumina data from GA1 and GA2 instruments but the final intention is to process ABI SOLiD and possibly 454 data as well. The first question people ask is usually "why are you doing this?". There are lots of good reasons, the first is that the primary data analysis tools supplied by the instrument are not currently available under open source licences and their algorithms are not widely documented. The primary data analysis itself can introduce biases (for example optical duplicates, or biases toward a particular base call), it's therefore important to know how your primary data has been processed on its way to becoming a base call. A second and related reason is as a platform for future research efforts, the image analysis and base calling problems are far from trivial and as we've seen with microarrays it has been beneficial to view primary data analysis as a research problem. At the moment if you want to try a new background subtraction method you will either not be able freely distribute your modifications (in the case of the Illumina pipeline) or not even have the source code to modify (in ABIs case). That's where Swift comes in, you can change the algorithms with relative ease and release your work under a GPL style licence, that's better for the users and better for the scientific community as a whole.

Chapter 2

Running Swift

Right now Swift processes Illumina data, ABI data shouldn't be too much of a leap for it (the images look very similar). It operates on a single tile at a time and you'll need to add scripts to run it over a full runfolder and parallelise it on a cluster. Swift has been developed on Linux, it's written in C++ and uses the GSL and FFTW libraries, consult your distribution documentation on installation instructions for these packages (you'll need gsl, gsl-dev and fftw3/fftw3-dev).

The first step is to check Swift out from the subversion repository:

```
svn co https://swiftng.svn.sourceforge.net/svnroot/swiftng/trunk
```

To build it simply type `make swift` that will build Swift using g++, there's also a `make swift_intel` target, which uses the intel compiler and OpenMP. The make files need a lot of work, if anyone is interested in setting up automake for us we'd welcome your contribution. Now you've build swift you can run it, here's the standard output:

```
Swift
                    align_every : align every Nth read
 background_subtraction_window : Background subtraction window size
                calculate_noise : calculate noise estimates
                         config : Configuration file
   correlation_aggregate_cycle : Sum images to this cycle for generating cross-channel offset
   correlation_median_channels : Take the median between channels before cross-channel offseting
   correlation_reference_cycle : Reference cycle for first pass of correlation
         correlation_subimages : The number of subimages to use, e.g. 2 will produce different offsets for
      correlation_subsubimages : The number of subsubimages, these are used to make the offset calculation
         correlation_threshold : Thresholding used when correlating images (fraction of max value)
  correlation_threshold_window : Thresholding used when correlating images (window size)
                          crop : Crop the input images?
                    crop_end_x : If cropping, end x position (or -1 for max x)
                    crop_end_y : If cropping, end y position (or -1 for max y)
                  crop_start_x : If cropping, start x position (or -1 for 0)
                  crop_start_y : If cropping, start y position (or -1 for 0)
                   dump_images : Save images to disk at various stages of processing (unused?)
                         fast4 : fast4 file prefix
                         fastq : fastq file prefix
                         img-a : A images filenames list
                         img-c : C images filenames list
                         img-g : G images filenames list
                         img-t : T images filenames list
                       intfile : Load for Solexa style intensity file, instead of performing image analysi
```

```
                      intout : GAPipeline style intensity file (optional)
                  max_clusters : Maximum number of clusters to generate, will fail if more than this are c
  optical_duplicates_distance : Distance in which to search of optical duplicates
optical_duplicates_mismatches : Number of mismatches allowed in sequences found
                   pair_break : Position of second end (first end length+1)
                purity_length : Length over which purity is calculated (no longer used)
              purity_threshold : Threshold on purity (minimum value must be more than this)
                           ref : Reference sequence for alignment (optional)
                        report : Report file
                segment_cycles : Segment images up to this cycle (identify clusters in all these cycles)
                          sigs : Signals file (optional)
                           tag : Tag to write at the top of the report, for example Run ID, lane and tile
                      textmode : write fast4 ascii files in text mode, full probabilities not ascii encode
                     threshold : Thresholding for segmentation, fraction of max value
               threshold_window : Thresholding for segmentation, distance
                     watershed : Apply watershed segmentation to thresholded images?

Where <X Images> is a line delimited list of tile images, in cycle order.
Note: This binary processes a single tile at a time
```

Swift can be run in two modes. In the first it runs as a basecaller only. That is to say it doesn't perform any image analysis but processes the intensity files produced by the Solexa pipeline. The following command would run Swift on an intensity file and write of fastq files, it would also align every 50th read against PhiX 174 to produce an error rate:

```
./swift --align_every 50 --intfile s_1_1_int.txt --fastq phitest --tag IL1_1023_1_1 --report runreport.xml
```

Swift also contains a native image analysis component, this currently excepts files which contain lists of image files to process, we've done it that way so that if the run folder format changes we can just rewrite our driver scripts. There's a simple bash script called `runswifttile` included which I use to run Swift on single tiles. You can run it as follows:

```
./runswifttile MYRUNFOLDER <lane> <tile> <TAG>
```

MYRUNFOLDER should point to the runfolder you wish to process (contains an Images subdirectory). ¡lane¿ and ¡tile¿ and the lane and tile number to process. ¡TAG¿ is a tag to place in the fastq, run reports, and files that will but output. So for example:

```
./runswifttile /staging/080000IL5_1500 1 1 IL15_1500_1_1
```

Would run swift on the runfolder at /staging/080000IL15_1500 on lane 1, tile 1. It will dump fastq files in the current directory at `pf.IL15_1500_1_1.fastq` and `nonpf_1500_1_1.fastq`. It will also create a runreport called `runreport.IL15_1500_1_1`. Right now this script also aligns reads against phiX using a brute force aligner (similar to `phageAlign`). You will obviously need to hack this for running in production, Tom Skelly at the Sanger Institute should be able to help you with this.

Swift writes out 2 fastq files, one for "PF" data the other for "non-PF" data, Illumina filter all their reads based on signal ambiguity and we use a similar metric in Swift (this is described in the algorithms section.

Chapter 3

Algorithms

The primary data analysis occurs in two phases, image analysis and base calling. Images analysis identifies individual clusters of DNA in the images, maps these clusters between sets of images (across cycle and for illumination under different lasers/filters) and extracts intensities from these clusters. The result is therefore a set of sequence of intensities, one for each cluster, across cycle and for each of the 4 channels.

3.1 Image Analysis

A raw Solexa/Illumina image is shown in figure **??**. Illumina image sets are broken down in to tiles, a tile is simply a region of imaging on the flowcell. Each tile as a set of images associated with it, 4 images for every cycle of chemistry.

3.2 Offset calculation

While these images cover the same general region they are slightly offset against each other. These offsets come for two sources. Firstly the stage has moved between cycles and the alignment will not be absolutely accurate when we perform subsequent imaging. There is therefore an offset which differs across cycle within each channel. Secondly because imaging has occurred under two lasers and two different filters the optical path for the different imaging frequencies is different. There is therefore a constant offset between each channel. The first problem is therefore to bring all these images in to alignment. We first compensate for the offset due to stage movement within each channel, then the cross-channel offset.

Images are thresholded to identify clusters, they are then cross-correlated within each channel (A,C,G and T) this is performed using an FFT (phase correlation). Thresholding is based on a window around each pixel within the image, the maximum value within this window is found and the threshold is set at a faction of this, for example, if the current pixel is within 0.7 of the maximum pixel in a 6 pixel window it will be set to 1 in the threshold image.

Offsets are calculated on a "sub-image" bases, currently Swift is using 4 "sub-images" (the original image is divided in to quarters). The offset of each of these is calculated independently. Further to this, these "sub-images" are divided in to "sub-sub-images". "sub-sub-images" are used to make the offset calculation robust, the median value of "sub-sub-image" offsets is used for the "sub-image". A sub-imaging offset calculation is being used until we can accurately determine the transformation caused by the instrument optics. It is important note that while the optical track differs between lasers/filters, the offsets within each channel will remain the same (and are simply due to stage movement). To make the offset calculation robust we therefore take the median offset of all channels for a given cycle/subimage.

At this point images within each channel should be correctly aligned. We now need to bring them in to alignment between channels. For each channel an aggregate image is created, this is simply the sum of all the images within that channel. This should for a normal genomic sample contain all the clusters on the tile. These aggregate images are then correlated against each other, again using sub-images. The offsets found are add to the channel offsets to create the resulting offset maps which are applied to the images.

3.3 Background Subtraction

Background subtraction is performed using morphological opening. This simply finds the minimal pixel within a window around each pixel and subtracts that value. As the minimal pixel is almost certainly at the image background level this provides a conservative estimate of the background.

3.4 Segmentation

The image is again thresholded using the method described in the offset calculation.

Clusters are extracted from thresholded images as groups of connected foreground pixels, they are stored in run length encoded form. Only the first n images are segmentated (where n is a tunable parameter). The clusters identified in these cycles are used as reference position, you should be wary of introducing optical duplicates, Swift contains an explicit optical duplicate filter to mitigate against this.

3.5 Registration

Reference positions are simply extended through all cycles in the aligned images.

3.6 Intensity Extraction

At this point each cluster as a position in all cycles and channels. Intensities are now extracted, the maximum intensity within the cluster of each cycle/channel is simply found. An intensity sequence for each cluster is therefore produced.

Chapter 4

Basecalling

Basecalling Illumina data is not straight forward as a number of artifacts remain in the data. These artifacts have 5 sources:

1. Crosstalk: Overlap in dye response between channels.
2. Phasing: Multiple or non incorporation of labelled bases.
3. Mixed Clusters: Clusters which are grown from more than one DNA template.
4. Sticky T: Build up of the T dye on the flowcell (not present in revised chemistry).
5. Signal Loss.

Corrections exist for the first 4 artifacts, unfortunately there is nothing that can be done about signal loss (in data analysis) and this is a significant limiting factor read length.

4.1 Crosstalk correction

The first correction we apply is for crosstalk between the dyes. The dye response simply overlaps, this means that if the A dye is attached to a molecule not only will it illuminate in the A channel, but also a little in the C channel. With the current Illumina chemistry the overlap is between the A and C channels and G and T channels. Errors are consequently biased between these bases.

There is a simple correction for crosstalk which comes from capillary sequencing and is due to Lee *(*et al) [**?**]. It is simple to visualise this method as putting regression lines though pairwise plots of intensities and using the slopes of these lines to derive a correction matrix. This is the method we use is Swift and that is also employed by the Illumina pipeline. In Swift the crosstalk matrix is derived from the first cycle and applied to all cycles. The first cycle is used because this is the only cycle where crosstalk does not become convolved with "phasing".

4.2 Normalisation against signal median

The intensities are re-expressed as a deviation from the median in this channel/cycle. The median should provide a fixed reference point, invarient to the build up for background signal.

4.3 Phasing correction

Phasing is the non-incorperation or multiple incorporation of labelled bases. The result manifests itself as a kind of cross-cycle crosstalk. That is if you measure the cycle 2 intensity you will have contributations from the true position 1

and position 3 bases. In Swift phasing is compensated for itteratively. We first examine the cycle 1 intensity where a given base was "called" (had the highest intensity) we compare these to the cycle 2 intensities where this base was not a call (not the brightest intensity). The faction of the intensity that is incorperated into the "non-calls" is calculated. This is then used to correct all bases. That is to say for all bases we subtract this fraction of the cycle 1 intensity from cycle 2. This compensates for forward crosstalk (phasing), that is the non-incorperation of a base in molecules of this cluster. A reverse calculation is also performed to compensate for multiple incorperation ("pre-phasing"). Not only are the adjacent bases compensated for, but all bases, using successively smaller fractions. Once the correction has been applied to cycle 1, it is then applied to cycle 2 and so on.

There is a limit on phasing, we currently assume that any faction over 0.5 is a miss calculation, we therefore only apply a correction up to a maximum of 0.5. Because of this a single phasing correction pass does not compensate for all the phasing. Three phasing passes are therefore currently used.

NOTE: I realise this section needs much clarification, helpful emails and questions are welcome: new at sgenomics dot org

4.4 Purity Filtering

Purity filtering isn't really a correction but it is designed to remove mixed clusters. Mixed clusters are those that have formed from more than one template, two fragments of DNA have simply attached themselves to the flowcell at almost adjacent positions. The cluster therefore represents sequence from both these templates resulting in bright peaks in more than more channel. Purity filtering discards these mixed clusters.

So called "Chastity" purity filtering is used. Chastity is defined as the ratio of the largest intensity to the sum of the largest and second largest intensities. If the average purity across a read falls below a given value the read is deemed "unpure".

4.5 Basecalling

After these corrections and filter have been applied basecalling can occur, the maximal intensity in a given cycle is used as the base call. Quality values are assigned based on the purity of the base scaled over a reasonable range. It appears that this provides a better indicator of quality than the Illumina raw quality scores, however no systematic evaluation of these scores has been made.