variantApp.sh - system requirements , and LINUX environment

variantApp.sh and its helper scripts - (C) Jelena Telenius 2019 , MIT licensed

A TOOL TO FIND OUT IF YOUR REGIONS OF INTEREST CAN BE VARIANT CALLED

The aim of variantApp.sh is NOT to provide a fully comprehensive variant call, but rather investigate, whether a BAM file carries a potential to such a run.

Variant call runs can take weeks, and this tool investigates the sorted bam overnight - to see whether it is worth it to fine-map the reads to reach "perfect variant call" - or whether there simply is not enough data to say anything about the variance within the sample.

As traditional - the tool gives one-click loadable data hubs, to visually inspect the presence of variant sites (SNPs , indels) within the data. If no variants are present - the sample is either too low quality, too shallow sequencing, or mapped in a too stringent manner (discarding the variant-containing reads in the mapping stage).

The tool can thus be used to investigate whether the "first round mapping" was throrough enough and suitable for downstream variant call : the variants should be "somewhat visible" already at this mapping stage : the subsequent "real" variant calling run cannot change the "approximate position" of the mapped reads - but it rather just finetunes the exact base locations within the mapped reads.

So - if no variants are seen in the variantApp.sh run - the sample needs to be mapped more carefully (or rather : more "carelessly" - i.e. allowing more mismatches and indels, to let the variance show in the data properly) - before giving the data for true variant calling.

Please note that the SNP positions and signal depth for them are EXACT, but the INDEL positions and signal depth are APPROXIMATE (provide the "maximum width" and "maximum depth" of signal rather than the actual signal from the BAM file). Also - the tool shouldn't be used as a "variant caller" - but rather just as a preliminary tool to investigate whether the data carries the potential to be variant called !

Please note that the tool is aimed for investigating "regions of interest" - not whole genome.
The regions of interest should be farther away from each others, than the estimated mapping distance (i.e. "secuencing library fragment length" -i.e. sonication size). For RNA mapping this would naturally mean "regions of interest" being GENES rather than exons. If overlapping regions ("within library fragment estimated mapping width") are given to the tool - the overlapping parts of the regions are counted twice (or for multiple overlaps - once for each overlapping region).

TOOLKITS AND SETTINGS THE PIPELINE NEEDS TO RUN

Extenal users : Download the code from :
https://github.com/Hughes-Genome-Group/variantApp

Internal users (WIMM) :

/home/molhaem2/telenius/variantApp/RELEASE/**variantApp.sh -h**

/home/molhaem2/telenius/variantApp/RELEASE/**variantApp_serial.sh -h**
(if the above parallelised version somehow doesn't run properly)

Example run command :

```
/home/molhaem2/telenius/variantApp/RELEASE/variantApp.sh \
-l regions.bed -b sorted.bam \
-f /databank/igenomes/Homo_sapiens/UCSC/hg38/Sequence/WholeGenomeFasta/genome.fa \
 -v params.txt
```

The params.txt needs to be filled, if UCSC data hub is to be generated.

param.txt (example)

```
SAMPLENAME testrun
GENOMENAME hg38
UCSCSIZES /databank/igenomes/Homo_sapiens/UCSC/hg38/Sequence/WholeGenomeFasta/chr_sizes.txt
SERVER http://userweb.molbiol.ox.ac.uk
SERVERPATH /public/telenius/DNase_PIPE/variants/VA_RUNS
DISKPATH /t1-data/public/telenius/DNase_PIPE/variants/VA_RUNS
USESYMBOLICLINKS
```

The web address of this readme PDF is :
http://userweb.molbiol.ox.ac.uk/public/telenius/variantApp/variantApp_JTelenius_2019.pdf

As is traditional, send all confusion, problems and comments to
jelena.telenius@imm.ox.ac.uk - or visit Jelena on her desk !

TOOL INPUT FILES

The tool takes in :

- BAM file (sorted by coordinate). The file can contain chimeric reads, unmapped reads, and orphan reads. The variant caller does not fine-map the data, so special care should be taken in the mapping stage - see attachment (1) below about a potential workflow to provide a suitable input bam to the tool. Multimapped reads are not treated differentially to uniquely mapped reads. Single-ended data is supported.

- BED file of regions of interest (used as the parallelisation unit). These regions should not overlap each others - any sequencing reads overlapping more than 1 of these regions, will be counted twice. Thus, the regions should be separated by at least +/- sequencing read lengths, to be on the safe side. The decision to use "regions of interest" as the parallelisation unit derives from the wish of providing meaningful re-use to the parallelisation sub-folders. Now they can be readily used in creative digging into the region-wise count data, without any new development measures.

- FASTA file of the reference sequence the BAM mapping was conducted with. Does not support variant aware mappers (sadly). Or, rather, the BAM may be mapped using variant aware mapper, but in this FASTA you need to set one of the alleles as “reference”.

STARTING THE RUNS

Usage instructions :

variantApp.sh -h

The runs are to be started in an empty folder - except the run script and parameter file provided for the script, and all files and folders are overwritten freely. User shouldn’t trust in this, however, but after a crashed run the folder should be cleared up, to ready it for a new run.

Exception to this rule is the visualisation generation (UCSC data hub) step of the run. The run will NOT overwrite existing public area folder (even when it is just a “fake folder” simulating a true server area visualisation folder). These folders and these data hubs are often times actively modified by users, so the script just crashes with the error message of “existing public data folder”, which the user can then remove (and not lose their special modifications to the visualisation settings with a bold auto-overwrite).

The visualisation is provided with extra restart function --onlyHub to restart only the visualisation part of variantApp.sh : this is to enable easier setup of the public server and disk area, as well as to support systems where the public server area is not visible to the cluster nodes, but is visible to the front end. In this case the visualisation part (which is not very heavy) can be ran in the front end, after the production run.

The parallelisation is an old-fashioned qsub -t threaded approach, but as the parallelisation is being handled via a separate script, and each sub-process have their own run scripts, this can easily be replaced by something different, driving the parallel runs in a different setting.

INTERPRETING THE VISUALISATION TRACKS

The colors of the tracks in the UCSC hub :

Overlay track (A) - all bases.

This track shows overlay of

i) total count of all mappings (base-wise counts) in RED
ii) all counts over the set Q-score (default 30) in BLUE
iii) all counts over the set Q-score giving the reference allele in GREEN

Overlay track (B) - base counts (for SNPS) - in percentages.

This track shows overlay of

i) total percentage of NON-REFERENCE base A (for each position) - in BLUE
ii) total percentage of NON-REFERENCE base T (for each position) - in YELLOW
iii) total percentage of NON-REFERENCE base C (for each position) - in RED
iv) total percentage of NON-REFERENCE base G (for each position) - in GREEN

Overlay track (C) - INDEL counts and locations (approximate).

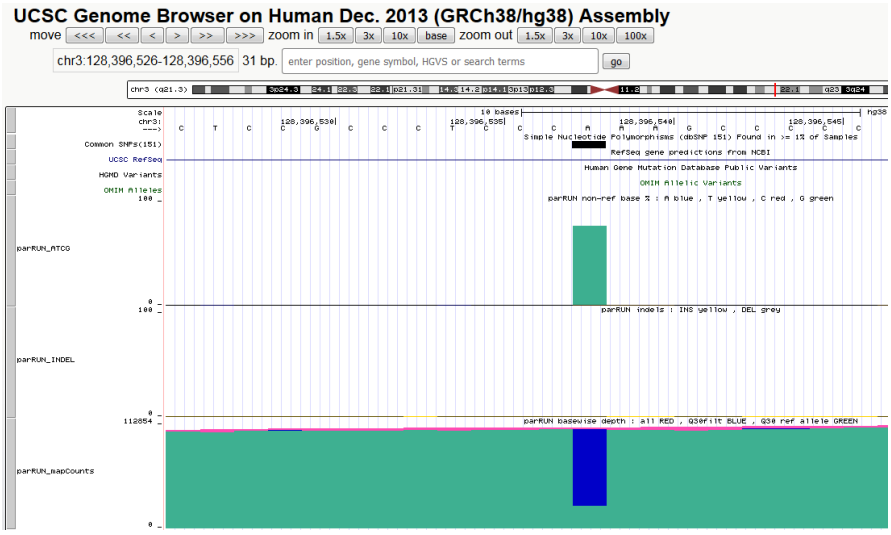
This track shows overlay of

i) total percentage of INSERTIONS (for each position - approximation) - in YELLOW
ii) total percentage of DELETIONS (for each position - approximation) - in GREY

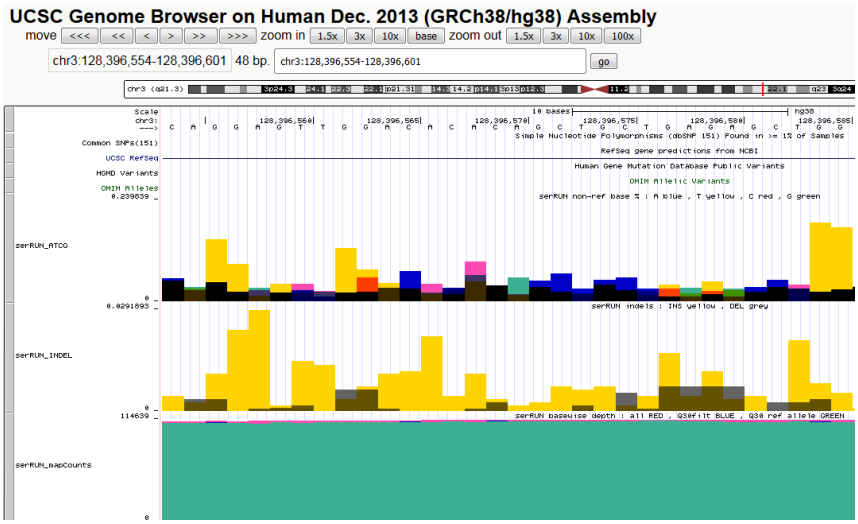
Example visualisation screenshots - after single-click loading the resulting data hub link to UCSC web browser :

1) SNP was found (A) ~ 77% of mapped reads show this non-reference allele
2) Background of false positive SNP and INDEL calls (less than 0.5% and 0.05% respectively)
3) Html page with log output files - accessible within the “track settings” page
4) Parameter input file shown via the above html file link

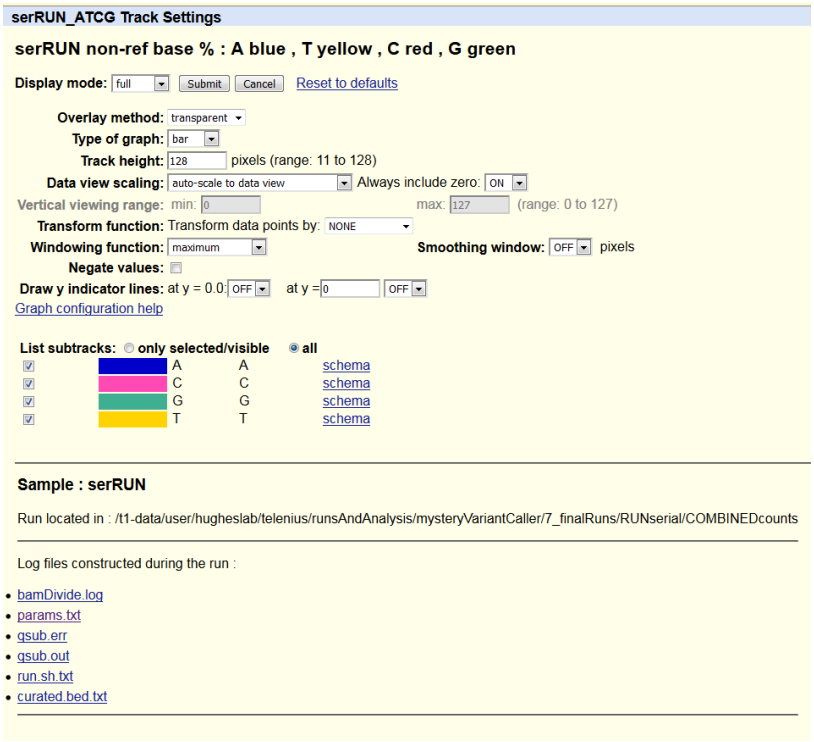
1)



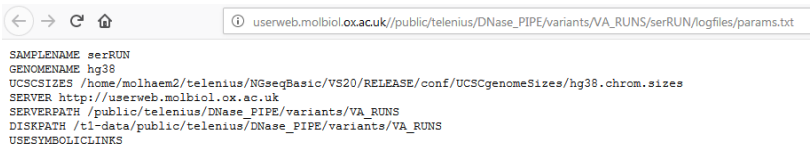
2)


UCSC Genome Browser on Human Dec. 2013 (GRCh38/hg38) Assembly
move
zoom in
zoom out
chr3:128,396,554-128,396,601
48 bp.
go
Common SNPs(151)
UCSC RefSeq
HGMD Variants
OMIM Alleles
serRUN_ATCG
serRUN_INDEL
serRUN_mapCounts
RefSeq gene predictions from NCBI
Human Gene Mutation Database Public Variants
OMIM Allelic Variants
serRUN non-ref base % : A blue , T yellow , C red , G green
serRUN indels : INS yellow , DEL grey
serRUN basewise depth : all RED , Q30filt BLUE , Q30 ref allele GREEN

3)


serRUN_ATCG Track Settings
serRUN non-ref base % : A blue , T yellow , C red , G green
Display mode: full Submit Cancel Reset to defaults
Overlay method: transparent
Type of graph: bar
Track height: 128 pixels (range: 11 to 128)
Data view scaling: auto-scale to data view Always include zero: ON
Vertical viewing range: min: 0 max: 127 (range: 0 to 127)
Transform function: Transform data points by: NONE
Windowing function: maximum Smoothing window: OFF pixels
Negate values:
Draw y indicator lines: at y = 0.0: OFF at y = 0 OFF
Graph configuration help
List subtracks: only selected/visible all
A A schema
C C schema
G G schema
T T schema
Sample : serRUN
Run located in : /t1-data/user/hugheslab/telenius/runsAndAnalysis/mysteryVariantCaller/7_finalRuns/RUNserial/COMBINEDcounts
Log files constructed during the run :
bamDivide.log
params.txt
qsub.err
qsub.out
run.sh.txt
curated.bed.txt

4)



EXAMPLE OUTPUT

The app produces a fully single-click downloadable data hub.

Example hub available here :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/parRUN/hub.txt

Same run, but ran with the serial executor (variantPipe_serial.sh) instead

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/serRUN/hub.txt

How to load that to UCSC browser with single click (if not familiar with UCSC data hubs) :

http://userweb.molbiol.ox.ac.uk/public/telenius/CaptureCompendium/CCseqBasic/DOCS/HUBtutorial_AllGroups_160813.pdf

The hub also combines the run statistics to a html page (integrally linked to the data hub - see the above instructions how to access this html page).

The page can be accessed outside the hub as well, direct address here :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/parRUN/hg38/description.html

A tar.gz package of the input, run scripts, and output of this run is downloadable in :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/RUNparallel.tar.gz

The input files for the run can be downloaded from these links :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/input/genome.fa

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/input/genome.fa.fai

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/input/regions.bed

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/input/regions_small.bed

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/input/sorted.bam

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/VA_RUNS/input/sortedTest.bam

INSTRUCTIONS FOR EXTERNAL USERS (if you are using intra-WIMM the below does not concern you)

Below all documentation about the system, environment, toolkits, parallelisation etc in detail, should someone want to lift the tool up in another system than the WIMM.

Also - if you are interested in the code workflow in general (to scavenge my scripts, perhaps ?) - the codes are described in detail below (you are welcome to re-use them - just cite me and this tool under MIT license, when re-using).

TOOLKITS AND SETTINGS THE PIPELINE NEEDS TO RUN

TABLE 1 - list of tools variantApp needs to run.

Toolkit	Tested with	Supports	Does not support
samtools	1.1	1.x	0.x (*)
bedtools	2.25.0	2.2x	2.1x
varscan	2.3.6	2.3.x	?
bedGraphToBigWig (**)			

(*) Samtools 0 can be used in combination with samtools 1 (to compare different ways to quantitate overlapping R1/R2 read ends to variants)

(**) bedGraphToBigWig is part of UCSCtools - and given with the variantApp.sh code.
It is believed any version of bedGraphToBigWig would perform here.
Another version can be downloaded from :
http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/

The setup of the tools is made in the file variantAppEnv.sh - and the user is to fill in the preferred commands to bring in the abovementioned toolkits (samtools, bedtools, varscan). The example setup uses the “module” system to bring the tools to the path. The tools are loaded only when they are needed. If the tools are wanted to be loaded “only once” - before the whole variantApp.sh is started (so that it inherits the existing environment), samtools1 has to be loaded instead of a “combination of samtools1 and samtools0”). In that case all the environment loading steps in variantAppEnv.sh can be commented out.

The variantAppEnv.sh also sets the amount of available MBs of RAM , and the preferred directory to store the temporary files - to set the file sorting to safe default values. Please modify these to match the architecture you are running in.

The tool assumes the default white field separator of unix basic tools in the system (awk, cut, paste) is tab \t

SERVER ADDRESS AND PUBLIC ACCESS DATA AREA

To utilise also the visualisation subroutines fully, variantApp.sh needs a server address, and a publicly visible data area served via this server.
Your command line tools should be allowed to write into this publicly visible disk area (i.e. wherever you run the pipeline, you should have the public area visible and write-able).

If you have no access to such disk space, the visualisations of the tool cannot be fully presented, but the bigwig files and the html page will be generated (just not moved to a public server area).

CODE WORKFLOW - (excluding the parallelisation implementation)

All the scripts should be kept in same folder (the folder is self-contained, but the scripts need to be in the same folder for them to function properly).

The scripts are designed as follows :

variantApp.sh (parallel) and variantApp_serial.sh (serial)

The main script of the application. Takes in user parameters, and starts the other scripts.

Usage :

variantApp.sh -h

variantApp_serial.sh -h

The serial script functionalities are exactly the same, and it calls exactly the same scripts in same way as the parallel one. It implements the “parallel stage” as a for-loop over the divided bams, instead of a true parallel implementation.

variantAppHelp.sh

The usage subroutine of variantApp.sh (just to make the script more readable.).

variantAppEnv.sh

All the tool loading subroutines - to be modified by the user to load the tools in their preferred manner. The user is to fill in the preferred commands to bring in the abovementioned toolkits (samtools, bedtools, varscan). The example setup uses the “module” system to bring the tools to the path. The tools are loaded only when they are needed. If the tools are wanted to be loaded “only once” - before the whole variantApp.sh is started (so that it inherits the existing environment), samtools1 has to be loaded instead of a “combination of samtools1 and samtools0”). In that case all the environment loading steps in variantAppEnv.sh can be commented out.

The variantAppEnv.sh also sets the amount of available MBs of RAM , and the preferred directory to store the temporary files - to set the file sorting to safe default values. Please modify these to match the architecture you are running in.

The rest of the scripts conduct the actual analysis. Most of them can work also as stand-alone scripts, and their help can be asked the same way as for the main script.

Here the **standalone scripts** (and their help command) and *subroutine-containing scripts* in the chronological order they appear in the workflow :

(1) variantBamdivide.sh -h **(2) variantPile.sh -h** *(3) variantCountcombine.sh*

(4) variantIndelApproximate.sh -h **(5) variantBigwigs.sh -h** **(6) variantVisualise.sh -h**

Below a description of each of them (good idea can be got from the output of the help commands as well).

(1) variantBamdivide.sh -h
Divides the bam along the bed file regions. Each of these get placed into its own output folder.

(2) variantPile.sh -h
Generates the variant calls (without fine-mapping the data first - this tool assumes EXCEPTIONALLY CAREFULLY mapped data - an example of such mapping protocol in Appendix (1) The tool uses *samtools mpileup* to count the bases in their locations, and *varscan readcounts* to parse the mpileup output file, and filter it for high sequencing quality bases. Filtering based on mapping quality is not conducted.
VariantPile.sh has a subroutine - containing script *variantPileVisNorm.sh* , which separates the different variant types to different output files, and normalises to variant percentages (over the total read counts).

(3) variantfileCombine.sh
This script has subroutines, which are called from the main **variantApp.sh** . It finds the generated variant-type-wise separated files from all the region-wise subfolders, and combines them to a single file per variant type. This subroutine needs the MBs of RAM and a location for temporary files (for file sorting). Before running, check that these parameters are well approximated in your *variantAppEnv.sh* file.

(4) variantCountcombine.sh

This script has subroutines, which are called from the main **variantApp.sh** . It contains the merge commands for the file combining (if the same bases are found in multiple files to be combined). This is quite fine, if the merging happens for single nucleotide wide regions, but for INDELS (which can be wider than this, it causes issues :
The major pitfall of this application is undoubtedly the visualisation of insertions and deletions. The script naiively assumes that no Insertions can overlap each other in genomic coordinates. The same for deletions. This is naturally not true - and a quick fix is provided here, to be able to visualise the percentage depths of these variants. An approximation is made, that the width of the variant is the “sum of the widths of all overlapping variants of the same type” (either INS or DEL), and the signal depth is the “sum of the signal depths” of all of them. This way the reported INS or DEL regions may be wider (but not narrower) than in reality, and look more prevalent (but not less prevalent) than they actually are. If the tool would ever be actually used for anything - this would be the first thing to be fixed ASAP, certainly.

(4) variantBigwigs.sh -h

Now as the the bedgraphs have been generated by **variantPile.sh** , combined by *variantCountcombine.sh* and modified by *variantIndelApproximate.sh* - this script generates the bigwigs from them, to prepare them for loading to UCSC genome browser

(5) variantVisualise.sh -h

This script generates a single-click loadable UCSC data hub from the bigwigs - complete with integral html description page (available by clicking any track in the UCSC browser). The description page loads all the log files generated during the run, as well as a list of regions of interest - to help the user navigate to interesting regions in the visual representation of the variants). The tool also prints a link to a pdf help sheet - to assist in loading the data hub to UCSC browser, if the user has not done that before.

All colors are checked to be color-blindness friendly.

PARALLELISATION IMPLEMENTATION

The parallelisation implementation mirrors the constraints of the cluster environment of the WIMM. This means an "old-fashioned" SGE cluster, where no memory limits and no processor limits are set for the jobs. This means that most of the tasks traditionally handled by the queue allocator itself, will need to be manually handled by the job submitting scripts instead.

This is a very hard-coded version of the parallelisation, highly optimised to the local cluster - sending jobs in a "courteous" manner, to not to fill up the whole cluster, and to monitor the memory and processor usage in a careful manner run-time, and respond to that accordingly, and keep solid log files of all this for troubleshooting and finetuning runtime hardcoded parameters.

This of course renders the parallelisation quite hard to test outside the WIMM (wher it was developed).

A serial version of the code is provided for testing purposes, should the parallel testing prove difficult of impossible : running variantApp_serial.sh instead of variantApp.sh replaces the parallel implementation with a serially excecuted for loom instead.

The parallelisation needs this infrastructure to work :

1) qsub as the job allocator
2) starting the parent job with qsub -cwd
3) available qsub options : -N -hold_jid -cwd -t
4) available environment variables : $SGE_TASK_ID $TMPDIR $SGE_O_WORKDIR (the $TMPDIR should point to the in-node temporary disk area, and combining $SGE_O_WORKDIR with qsub -cwd should have the $SGE_O_WORKDIR pointing to the directory the job was started in. This folder should thus be queue-visible run-time.

The parallelisation code structure is as follows :

1) variantApp.sh reads in the wished parallelisation options (defaults to ordinary multithreaded qsub -t job with 4 cores). User can set the amount of cores, and switch between "-t" and "wholenode" modes - wholenode mode assumes the job to be submitted so, that a whole node is reserved for the run in the qsub command of the parent script (i.e. the children are

started WITHIN the node which Is already reserved), instead of starting the parent script in the queue, and interactively spawning new children within the run. The threaded -t mode needs one of these two options to be true for the queue system in question : either a) the parent job can interactively spawn new qsub jobs when already in the queue (this is how the run is designed to work in the WIMM where the code was developed), or b) the headnode allows jobs to be running ON IT - so that the main script can operate in the head node, and interactively submit from there. Option b) would need a bit finetuning to the script, as the steps before and after parallelisation are NOT qsubbed separately, but rather ran straight in the main script (this would need to be changed if this kind of "head node as monitoring spot" design would be implemented. If the run is set to be ran in "wholenode" mode, the user can also set to use temporary directory to store the run run-time (instead of the work directory of the run)

2) variantApp.sh calls a subroutine in variantParallelRuntools.sh to write a run script called run.sh for each of the to-be-submitted bam files - into the folder where the bam itself is stored.

3) variantAll.sh calls subroutine in variantMultitaskers.sh to start the requested jobs in an orderly manner using qsub commands within the script itself. The multitasker subroutine links the submitted jobs to the folder they take the run command and data from, with a text file list runlist.txt generated above when preparing the runs for parallelisation with the variantParallelRuntools.sh subroutine. These qsub jobs start the requested script (variantRunInThread.sh or variantRunInThreadWorkDir.sh or variantRunInWholenode.sh or variantRunInWholenodeWorkDir.sh), which know knows where to find the folder in question and the run.sh it is to be running based on run list text file generated for it to investigate. The jobs report their error and output logs to a designated place, for later inspection, and also report if they exited properly or crashed. The multitasker subroutine is not under set -e -o pipefail , as the errors are dealt carefully manually, and inspected before crashing - to ensure orderly cleaning up the temporary areas and generating crashing reports before aborting the whole run. Each within-qsub script (variantRunInThread.sh or variantRunInThreadWorkDir.sh or variantRunInWholenode.sh or variantRunInWholenodeWorkDir.sh) is under set -e -o pipefail , however, so they do crash out quickly - and also are equipped with a cleaner subroutine to clean up "as they crash" and document the crash event and temporary files clearly for troubleshooting, before exit (i.e. they have an exit trap to manage this).

4) variantAll.sh calls subroutines in variantParallelRuntools.sh are used to summarise if all runs went fine, and if they crashed, which ones crashed, and how. Memory and task usage summary files are generated as well. The main variantApp.sh script will be aborted if issues were seen.

The scripts list for the parallelisation implementation :

variantMultitaskers.sh (to execute the multitasking runs)

variantParallelRuntools.sh (to set up the run scripts and run folders, to collect reports after all runs)

variantRunInThread.sh (threaded run, using $TMPDIR to store data run-time)

variantRunInThreadWorkDir.sh (threaded run, using run starting dir store data run-time)

variantRunInWholenode.sh (wholenode run, using $TMPDIR to store data run-time)

variantRunInWholenodeWorkDir.sh (wholenode run, using run starting dir store data run-time)

variantTesters_and_loggers.sh (to keep cd mv cp rm commands safe when referring to ${folder} style arguments - to not to end up doing devastating things due to typos in code)

variantTesters_and_loggers_test.sh (the tester main of the above script - to abort the script execution if the above testers cannot be properly loaded - better safe than sorry)

TESTING AND ERROR REPORTING

The code was tested in default parameters. It can be considered to be “late beta” - i.e. some of the flags may not work (possible but improbable as not thoroughly tested), but all error situations should be accompanied with a clear crash message. The code is mostly based on scripts which are part of solid analysis pipelines (all of these constructed by Jelena Telenius) and already in production and stable - so all issues should be minor, and easy to fix.

Finding the error log files of each of the submitted parallel child processes is a known “issue” of the tool the parallelisation implementation was taken from - and was not fixed in making this application (time constraints). The error messages are stored in the error log files in folder DIVIDEDbams/qsubLogFiles (that sounded like a logical place to store them - but it seems to be hard to find when inspecting the crashed runs in practise).

SYSTEM SPECIFICATIONS

(within the system where CCseqBasic was developed and tested)

GNU basic tool versions

(GNU coreutils) 8.4 - Copyright (C) 2010 Free Software Foundation, Inc.
GNU which v2.19 - Copyright (C) 1999 - 2008 Carlo Wood.
GNU Awk 3.1.7 - Copyright (C) 1989, 1991-2009 Free Software Foundation.
GNU sed version 4.2.1 - Copyright (C) 2009 Free Software Foundation, Inc.
grep (GNU grep) 2.20 - Copyright (C) 2014 Free Software Foundation, Inc.

Other BASH utils (not obligatory) :

hostname - hostname 1.100 (2001-04-14), net-tools 1.60
module - VERSION=3.2.10 , DATE=2012-12-21

Machines where the pipe was tested : (uname -a)

Linux 2.6.32-642.11.1.el6.x86_64 #1 SMP x86_64 x86_64 x86_64 GNU/Linux
Linux 2.6.32-642.6.2.el6.x86_64 #1 SMP x86_64 x86_64 x86_64 GNU/Linux

Centos version : (cat /etc/centos-release)

CentOS release 6.10 (Final)

Sun grid engine version

GE 6.2u5

APPENDIX (1) - CREATING A CREDIBLE BAM FOR THE TOOL

1) pre-processing pipe, to give eligible bam (to ensure we know what goes in into the tool)

The basis of all variant calling is a good initial mapping of the fastq data.

The app I have built assumes the following to be true for the input bam file :
- if read multimapped, only one of these mappings is reported (like bowtie2 default output)
- the bam can contain orphan reads, and chimeric reads in such manner that R1 and R2 mapped far away from each others in the chromosome, or even to different chromosomes

To illustrate the generation of a bam file for the app, an example script using the tool NGseqBasic to generate the needed bam files and catenating them for the final input for the app is provided as well.

(NGseqBasic is available in : https://github.com/Hughes-Genome-Group/NGseqBasic)

This script conducts :

- First mapping with NGseqBasic default parameters, to map "easily mappable" reads in a stringent fashion for best quality mapping for "easy-to-map" reads
(this round is assumed to be able to map ~ 90% of the data)

- Second mapping for NGseqBasic output fastqs (reads not mapped as concordant proper pairs in first mapping) - with more allowing mapping parameters, to catch larger indels and larger amount of complex snp-combinations within-one-read.

- Third mapping and fourth mapping with STAR - first in end-to-end mode, then using local mapping (or hisat2 to allow common variants in the reference), to allow mapping of reads deviating more from the reference, than traditional mappers such as bowtie2 would allow. Fifth mapping like the fourth mapping, but allowing single-end mappings to be reported (rather than forcing paired mapping as before).

A full example script of this available in :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/credibleBam/combo_sh.txt

And its output logs in here :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/credibleBam/COMBO_qsub.err

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/credibleBam/COMBO_qsub.out

The NGseqBasic output data hubs and description pages (steps 1-2) available in here :

First step :
http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/round1/HubFolder/hub.txt

Second step :
http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/round2/HubFolder/hub.txt

How to load a data hub (such as the above ones) to UCSC genome browser :
http://userweb.molbiol.ox.ac.uk/public/telenius/CaptureCompendium/CCseqBasic/DOCS/HUBtutorial_AllGroups_160813.pdf

The quality-control html-pages of steps 1-2 (also available via the above data hubs) :

First step :
http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/round1/BigWigFolder/run1_sample1/hg38/description.html

Second step :
http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/round2/BigWigFolder/run2_sample1/hg38/description.html

The combining stage (merging bams of all 5 steps above) script available here :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/credibleBam/merge_sh.txt

And the output log files of the combining stage :

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/credibleBam/MERGE_qsub.out

http://userweb.molbiol.ox.ac.uk/public/telenius/DNase_PIPE/variants/credibleBam/MERGE_qsub.err

APPENDIX (2) - ABOUT THE DECISIONS MADE DURING THE CODING OF THE APPLICATION

The aim of variantApp.sh is NOT to provide a fully comprehensive variant call, but rather investigate, whether a BAM file carries a potential to such a run. Variant call runs can take weeks, and this tool investigates the sorted bam overnight - to see whether it is worth it to fine-map the reads to reach “perfect variant call” - or whether there simply is not enough data to say anything about the variance within the sample. The tool can also be used to investigate whether the “first round mapping” was

throrough enough and suitable for downstream variant call : the variants should be “somewhat visible” already at this mapping stage : the subsequent “real” variant calling run cannot change the “approximate position” of the mapped reads - but it rather just finetunes the exact base locations within the mapped reads. So - if no variants are seen in the variantApp.sh run - the sample needs to be mapped more carefully (or rather : more “carelessly” - i.e. allowing more mismatches and indels, to let the variance show in the data properly) - before giving the data for true variant calling.

Below described how I approached this task - and what are the pitfalls and strong points of this approach.

This is what I decided to do :

- **Focus first to the definition of "eligible bam"** : and to generate such a file
(using this as a showcase of expertise in read mapping and mappability, data visualisation, and analysis tool development)

- **Then identify subroutines and scripts to re-use in the actual variant calling task.**
Here decided to go with rudimentary scripts which were originally built by Jelena Telenius for off-target crispr analysis, to assist a publication to go through in revision stage in late 2016.
These codes were never polished to the level of a finalised tool, and due the limited time given to complete the variant calling app, certain parts of these codes do remain rudimentary.
Especially the extensive use of 'cut' 'paste' and 'awk' instead of a proper Perl parsing script with proper error messages when file format is not what is expected, is a great pitfall of these scripts.
Jelena acknowledges the fact, that the approach taken here is not of the traditional variant calling,with extensive fine-mapping stage, but rather an alternative one, where emphasis is put into the mapping stages, and fine mapping is skipped, and variants are just called "as-is" and counted. Naturally this approach is not suitable for clinical scale analysis of variant data. However, in the interest of time, no proper fixes and adidtions to the existing codes were made, but rather they were used as-is.
These data of off-target crispr sites also provided the test data set used in this task. So the app is not tested with real exome sequencing data, but rather a data set where a couple of dozen genomic loci were sequenced extremely deeply, to see possible crispr off target on the loci. The user-defined parameters are defined so, that the app is believed to be useful for more generalised situation as well.

- **Last the parallelisation subroutines and scripts to re-use were identified.**
These scripts are much more mature than the variant calling scripts, and are a part of a stable toolkit built by Jelena Telenius (just released from late beta to production). However, the parallelisation is for intra-house use only - so the most fine-tuned features presented in the solution only show their true value in the infrastructure they were custom-built for.
There is a traditional qsub -t [threads] based alternative running mode provided, to allow more traditional execution. As Jelena doesn't have extensive experience in working with cloud-based systems, no parallelisation to such environment was provided. However, as the code generates and saves the scripts to-be-ran in the cluster beforehand, before submitting them, modifying this to cloud setting should be a trivial task. The cluster system of WIMM (where the code was developed) is also quite unpredictable when it comes to using automated pipelining approaches such as ruffus or snakemake,
so the Jelena doesn't yet have tested runs using these parallelisation methods - as she is only experimenting with such approaches, and does not want to present anything she hasn't rigorously tested before giving it out.

- **The main aim of Jelena was to provide something which she can**
  1) **somewhat trust** (familiar with the tools and the task enough, to be aware of the artifacts and pitfalls of the approach)
  2) **can complete the task in the time frame given** - including providing a proper input file, full test of the code, and proper documentation.

- **Jelena had extensive plans to investigate the non-mapping reads**, to verify no translocations or variants widely different from the reference, and only-repeatome-mapping reads are not omitted when analysing the data, but this was completely skipped in the interest of time. Jelena was **also planning to analyse the chimeric reads** (where the read pairs map to different chromosomes) carefully, to get better data for the translocation events, but this was fully skipped in the interest of time, as well.

- **The quality control measures given are reduced to the counts per 10 000 normalisation**, setting the treshold for reliable interpretation, and the sequencing base-quality based filtering before counting the variants. No mapping quality based filtering was done, even though it was seen relevant and necessary (as would have been to do fine-mapping for the poorly mapped reads). The sequential mapping protocol is believed to render most of the needed fine mapping unnecessary,
but Jelena has no experience how well this assumption actually holds, when real exome sequencing data is analysed this way without an actual fine-mapping step. The mapping errors were seen to be very few, when the method was originally developed - but it has to be kept in mind that the data type analysed was quite different (very deep sequencing, where deviations in mapping resulted mostly in PCR errors rather than real variation against the reference genome). However, Jelena DOES have full confidence that her method could pretty possibly be actually implemented even in clinical setting - the shortcuts taken are well tested and have not shown even minute accumulation of false positive variants (for the data Jelena has used to test it). When it comes to false negatives, it may be a different story (as the chimeric reads are not properly analysed, and the mapper used to generate the bam was not variant-aware). However, the sequential mapping used should deal with these situations quite well as well - and only when truly accurate allelic imbalance numbers are needed, may re-mapping of the data be necessary.

- All these codes are given under MIT license, and Jelena expects to be acknowledged for any re-use of the codes and the underlying analysis protocol within them.