IX International Course of Massive Data Analysis FOR GENOMICS

Cristina Y. González
cgonzalez@cipf.es

Introduction to HPG Variant architecture and development

Overview

Global schema: Binaries

- HPG Variant VCF Tools
- HPG Variant Effect
- HPG Variant GWAS

Describing the architecture by example: GWAS

- Main workflow
- Reading configuration files and command-line options
- Parsing input files
- Parallelization schema

How to compile: Dependencies and application

Hacking HPG Variant

Let's talk about...

- Global schema: Binaries
  - HPG Variant VCF Tools
  - HPG Variant Effect
  - HPG Variant GWAS
- Describing the architecture by example: GWAS
  - Main workflow
  - Reading configuration files and command-line options
  - Parsing input files
  - Parallelization schema
- How to compile: Dependencies and application
- Hacking HPG Variant

Binaries: HPG Variant VCF Tools

HPG Variant VCF Tools preprocesses VCF files

- Filtering
- Merging
- Splitting
- Retrieving statistics

Binaries: HPG Variant Effect

HPG Variant Effect retrieves information about the effect of mutations

- Querying a web service
- Uses libcurl (client side) and JAX-RS/Jersey (server side)
- Information stored in CellBase DB

Binaries: HPG Variant GWAS

HPG Variant GWAS conducts genome-wide association studies

- Population-based: Chi-square, Fisher's exact test
- Family-based: TDT
- Read genotypes from VCF files
- Read phenotypes and familial information from PED files

Let's talk about...

Global schema: Binaries
- HPG Variant VCF Tools
- HPG Variant Effect
- HPG Variant GWAS

Describing the architecture by example: GWAS
- Main workflow
- Reading configuration files and command-line options
- Parsing input files
- Parallelization schema

How to compile: Dependencies and application

Hacking HPG Variant

Architecture: Main workflow

The main flow of the application involves:

- Reading configuration files (libconfig) and command-line options (Argtable library)
- Parsing PED and VCF files (Ragel State Machine Compiler)
- Running analysis in parallel (OpenMP)
- Writing the output

Everything is implemented using C99

Architecture: Reading configuration files

There are two kinds of options:

- Shared among all tools in the HPG Variant suite
- Specific to a tool

Interesting to use nested options in configuration files: **libconfig**

```
config_t *config = (config_t*) calloc (1, sizeof(config_t));
if (!config_read_file(config, filename)) {
    LOG_ERROR_F("Configuration file error: %s\n", config_error_text(config));
    return CANT_READ_CONFIG_FILE;
}

if (!config_lookup_int(config, "gwas.epistasis.num-threads", num_threads_opt->ival)) {
    LOG_WARN("Number of threads not found in config file, must be set via command-line");
} else {
    LOG_DEBUG_F("num-threads = %ld\n", *(num_threads_opt->ival));
}
```

Architecture: Parsing command-line options

getopt not enough for merging those sets of options! **argtable**

```
struct arg_file *vcf_filename_opt = arg_file1("v", "vcf-file", NULL, "VCF file used as input");
struct arg_int *num_threads_opt = arg_int0(NULL, "num-threads", NULL,
                                           "Number of threads to run a task in parallel");

void **tool_options = malloc (2 * sizeof(void*));
tool_options[0] = vcf_filename_opt;
tool_options[1] = num_threads_opt;

int num_errors = arg_parse(argc, argv, tool_options);
if (num_errors > 0) {
    arg_print_errors(stdout, end, "hpg-var-gwas");
    exit(1);
}

char *vcf_filename = *(vcf_filename_opt->filename);
int num_threads = *(num_threads_opt->ival);
```

Architecture: Parsing input files

VCF and PED files are plain text formats containing genotypical and phenotypical information

- Contents can be expressed using regular expressions
- Difficult to implement good error-checking by hand
- Better to use a parser generator: **Ragel**
- Similar to Lex + Yacc/Bison, but easier to use
- It also generates graphs!
- Read in batches for memory and computation efficiency

Architecture: Parsing input files (example)

Projections represent the structure:

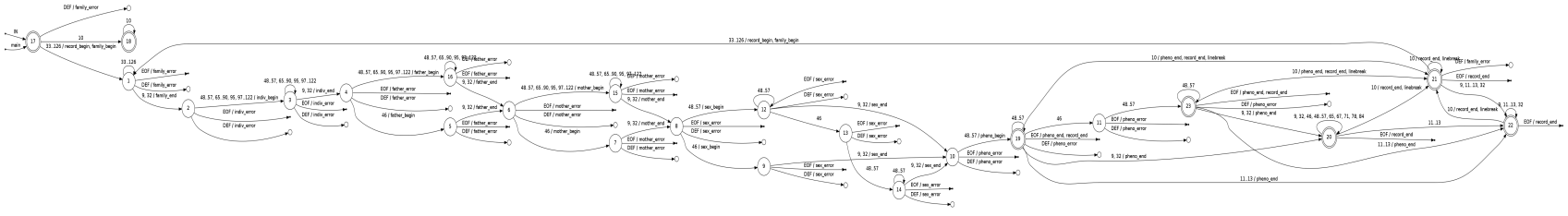
```
main := (fileformat)? ("\n")* (header)? ("\n")* (delimiter)? ("\n")* (records)? ("\n")* ;

fileformat  = "##fileformat=" format_name
              >fileformat_begin %fileformat_end $err(fileformat_error) "\n" >linebreak;
format_name = (alnum | punct)+ ;
```

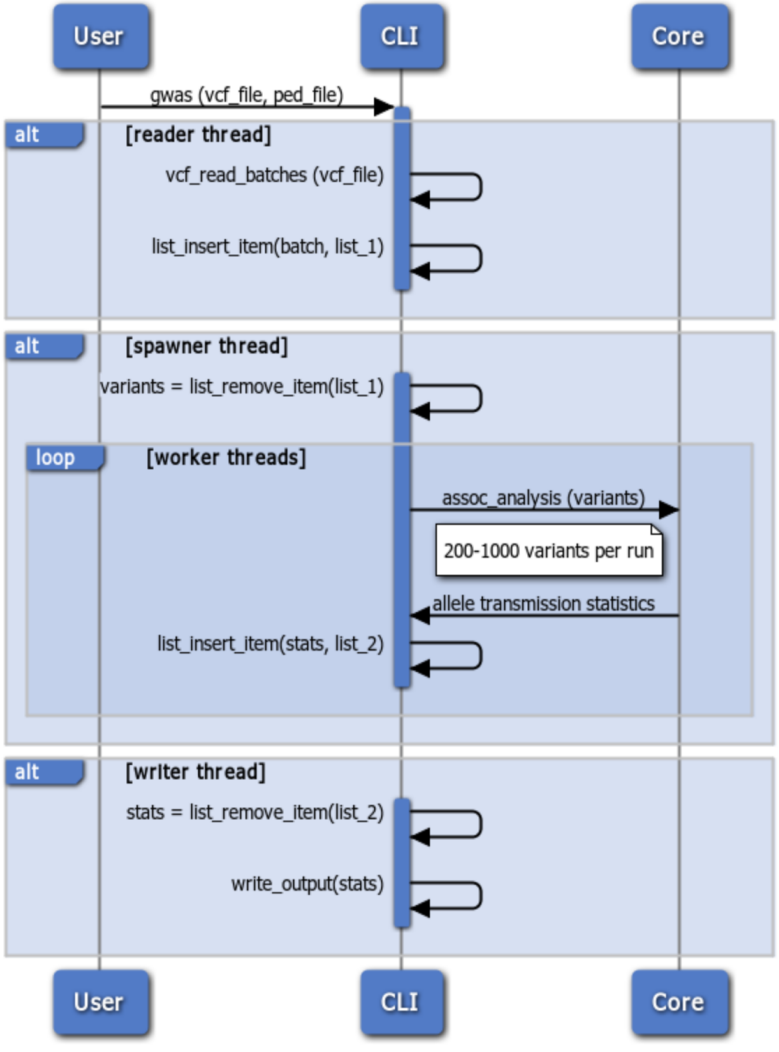
Actions react to certain transitions:

```
action fileformat_begin { start = p; }
action fileformat_end { set_vcf_file_format(start, p-start, file); }
action fileformat_error { printf("Line %d (%s): Error in file format\n", lines, file->filename); }
```

PED automata:



Architecture: Parallelization schema




```
#pragma omp parallel sections
{
#pragma omp section
    {
        vcf_read(vcf_file, list_1);
        notify_end_reading(vcf_file);
    }
#pragma omp section
    {
        omp_set_nested(1);
        #pragma omp parallel num_threads(n)
        {
            while (item = get_item(list_1))
            {
                assoc_analysis(item, list_2);
            }
        }
    }
#pragma omp section
    {
        while (item = get_item(list_2))
        {
            write_output(item, output_file);
        }
    }
}
```

Let's talk about...

- Global schema: Binaries
  - HPG Variant VCF Tools
  - HPG Variant Effect
  - HPG Variant GWAS
- Describing the architecture by example: GWAS
  - Main workflow
  - Reading configuration files and command-line options
  - Parsing input files
  - Parallelization schema
- How to compile: Dependencies and application
- Hacking HPG Variant

How to compile

- Instructions for Debian/Ubuntu and Fedora at the OpenCB project wiki: `http://www.opencb.org/projects/hpg/doku.php?id=building`
- Requires a C compiler, Linux headers and the SCons build-system
- In Debian-based distros, only cprops needs to be compiled
- In Fedora, only argtable needs to be compiled
- We're working on optimizing this process

Hands-on: Getting the code (I)

Fork the following repositories at GitHub:

- opencb-hpg/hpg-variant
- opencb-hpg/bioinfo-libs
- opencb-hpg/common-libs
- opencb-hpg/math

Hands-on: Getting the code (and II)

Recommended cloning setup:

- `$HOME/appl/bioinfo-c/hpg-variant`
- `$HOME/appl/bioinfo-c/libs/bioinfo-libs`
- `$HOME/appl/bioinfo-c/libs/common-libs`
- `$HOME/appl/bioinfo-c/libs/math`

Hands-on: Configuring the working directory

Need to initialize the submodules pointing to libraries:

- `cd $HOME/appl/bioinfo-c/libs/common-libs`
- `touch .git/git-daemon-export-ok`
- Repeat for `bioinfo-libs` and `math`
- `cd $HOME/appl/bioinfo-c/hpg-variant`
- `git submodule update --init`

It should notify that the submodules' files are being downloaded, if not, **please tell me** because you won't be able to code anything without them!

Hands-on: At last... compiling!

Everything should be ready, so simply run:

```
scons
```

And 3 binaries will be created in the `bin` subfolder!

It is easy to switch to debug mode, just run: `scons debug=1`

Let's talk about...

- Global schema: Binaries
  - HPG Variant VCF Tools
  - HPG Variant Effect
  - HPG Variant GWAS
- Describing the architecture by example: GWAS
  - Main workflow
  - Reading configuration files and command-line options
  - Parsing input files
  - Parallelization schema
- How to compile: Dependencies and application
- Hacking HPG Variant

Hands-on: Hacking HPG Variant

Add new statistical values about variants contained in a VCF file

Important source folders:

- `hpg-variant/libs/bioinfo-libs/bioformats/vcf`: Core of VCF statistics
- `hpg-variant/vcf-tools/stats`: Interface that uses the previous from HPG Variant

Hands-on: Main steps

1. `.../bioinfo-libs/bioformats/vcf/vcf_stats.*`: Add new fields to `file_stats_t` and `variant_stats_t` structures
2. `.../bioinfo-libs/bioformats/vcf/vcf_stats.c`: Add new calculus to functions `get_variant_stats` and `update_file_stats`
3. `hpg-variant/vcf-tools/stats/stats_runner.c`: Add new `fprintf` sentences to functions `write_variant_stats` and `write_file_stats`

Acknowledgments

This work is supported by grants from
the Spanish Ministry of Economy and Competitiveness and the
Consellería de Educación of the Valencian Community,
and by the Chair in Computational Genomics funded by Bull.
It has been carried out as part of the HPC4Genomics initiative.
We also thank the support of the National Institute of
Bioinformatics and the CIBER de Enfermedades Raras (CIBERER).