FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV-PPR

Semester Thesis on Parallel Programming Subject

Discovering the Correlation Formula between 3D Accelerometer Data and Heart Rate

Daniel Schnurpfeil (dschnurp@students.zcu.cz)

December 11, 2023

Contents

1 Assignment

The goal of the work is to find a formula for the correlation between a 3D accelerometer and heart rate. Download the relevant portions of data files - HR and ACC files from physionet.org. Normalize the input time series using threads. Then, write a simple function generator with the following prototype:

```
double transform(const double acc_x,
                 const double acc_y,
                 const double acc_z);
```

Calculate the correlation of the resulting time series with the heart rate. The smaller the value, the better the correlation. Essentially, this will give you a fitness function if you generate transformation functions with an evolutionary or genetic algorithm, such as genetic programming or grammatical evolution. Vectorize this algorithm.

Run the entire calculation on OpenCL or **Vulkan** devices - you don't need to write an expression parser, as the OpenCL driver will handle it for you. It is advisable for the calculation of the correlation measure and transformation to occur entirely on the GPU, avoiding unnecessary data copying.

Finally, select the transformation function with the highest correlation and display it on a graph along with the heart rate. Generate the graph in .svg format.

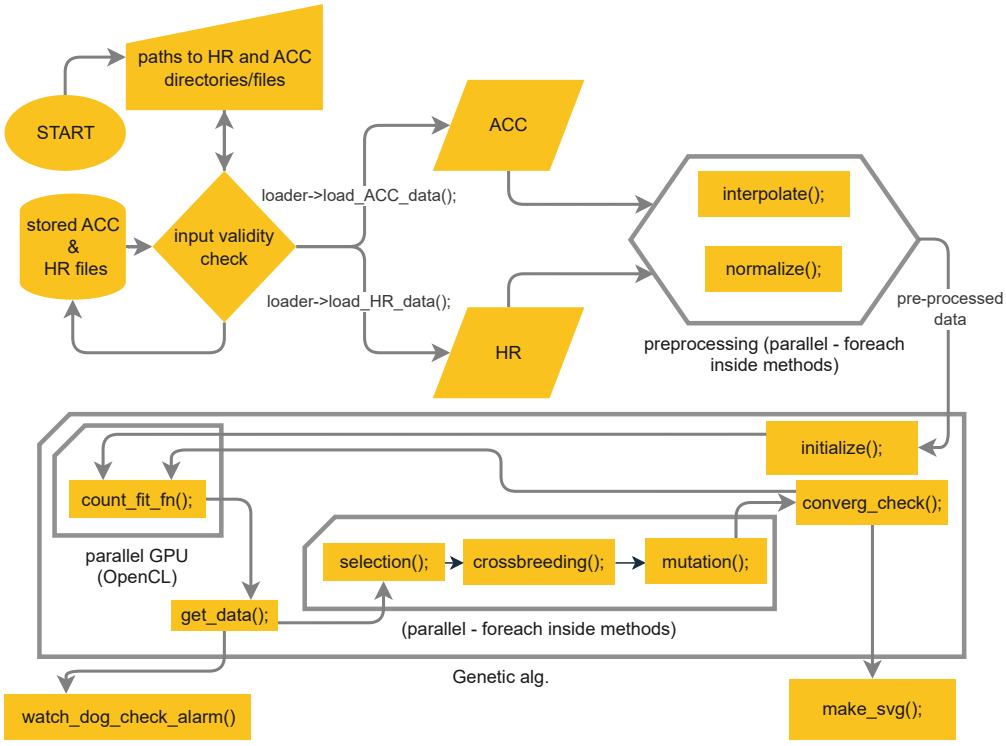

Figure 1: First application architecture proposal

2 Analysis

After first application architecture proposal there is need to look at input data deeper and understand their meaning and do some optimizations to reduce computation overhead.

2.1 Input Data

Input data for our goal are data from 3D accelerometer in CSV format:

- “Timestamp” as a datetime (Y-m-d H:M:S), seconds as float numbers (with milliseconds)
- ”X”, “Y”, “Z” as an orientation

and heart rate in CSV format:

- “Timestamp” as a datetime (Y-m-d H:M:S)
- ”X” as a heart rate (heart beats per minute)

Data with another format are considered as invalid. Such as HR_001.csv...

2.2 Application Architecture proposal

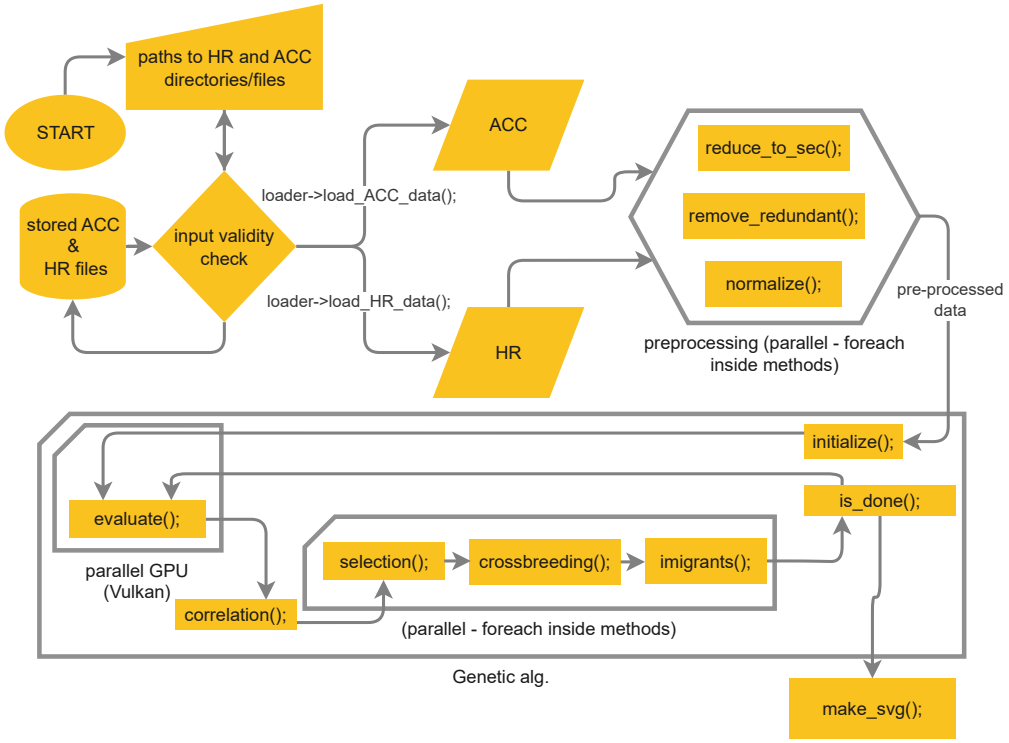

Figure 2: Application architecture

2.2.1 Loading

The loading larger data which can fit into RAM, there is an approach to load them to RAM and then, filter and structuring the data to structure of arrays for (implicit) vectorization availability.

2.2.2 Preprocessing

After data loading we have two options to preprocess the data. The first option is to reduce dataset and counting with seconds so we are loading only seconds from ACC file. The second approach is to synthetically extend dataset to interpolate timestamp between seconds in HR file.

2.2.3 Genetic Algorithm

The genetic algorithm begins with the initialization of a population, where individual equations are generated randomly. Following this, the fitness function is computed for each equation, assessing the person correlation with a reference metric HR (Heart Rate). The selection phase then identifies individuals with higher fitness, than mean fitness (no need to sort). Crossover, a genetic operation, takes place, where pairs of individuals exchange information to produce new 2 offsprings. The fitness function is then recalculated for these new individuals. By random crossover point. Finally, a new population is created based on the selected and evolved individuals, completing one iteration of the genetic algorithm.

2.2.4 Generated Formulas

We need to know that for simplicity during evaluation the multiplication has same priority as plus or minus signs. The division is defined as multiplication with number in interval 0 and 1. So the lenght of equation can have size from 3 nodes to hardcoded maximum 14 nodes and root coefficient.

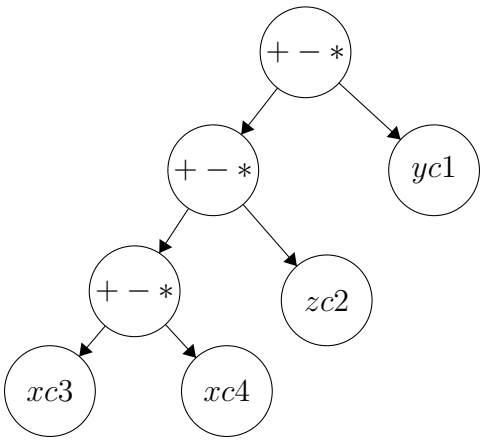

Figure 3: Formula sample

3 Implementation

For implementation is used C++ 20 standard and Google code style with the exception of enforcing line length restrictions. The application source code is structured to folders:

```
src
├── genetic_alg (genetic algorithm related code)
│   ├── Equation.cpp
│   ├── Equation.h
│   ├── Node.cpp
│   ├── Node.h
│   ├── Population.cpp
│   └── Population.h
├── gpu (gpu computation related code)
│   ├── ComputationUnit.cpp
│   ├── ComputationUnit.h
│   └── helpers (helper functions inspired by Slava Savenko)
│       ├── LICENSE
│       ├── vulkan_helpers.cpp
│       └── vulkan_helpers.hpp
├── io (loading and visualization output related code)
│   ├── Loader.cpp
│   ├── Loader.h
│   ├── RecordACC.cpp
│   ├── RecordACC.h
│   ├── RecordHR.cpp
│   ├── RecordHR.h
│   ├── svg_gen.cpp
│   └── svg_gen.h
├── main.cpp
└── preprocessing (prepossessing related code)
    ├── utils.cpp
    └── utils.h
```

Implementation corresponds to the application architecture can be seen at the figure 2 to a high degree. Genetic algorithm iterates over defined epochs. In each epoch there is computed Pearson correlation over whole dataset. The Pearson correlation is not computed on GPU due to little more complexity in Pearson correlation and optimization of precomputing the HR data and their variance for the Pearson correlation compute.

3.1 Data Structures

The data loading process involves organizing orientation and heart rate information into structured arrays. For orientation data, the structure of arrays includes vectors for microsecond, y-axis, z-axis, x-axis, and timestamps represented by

```
std::vector<float> microsecond;
std::vector<float> y;
std::vector<float> z;
std::vector<float> x;
std::vector<std::tm> timestamp;
```

respectively. Each vector corresponds to a specific aspect of the orientation data, such as time values and the three-dimensional orientation components. Similarly, heart rate data is structured with vectors for heart rate values

```
std::vector<float> x;
```

and timestamps

```
std::vector<std::tm> timestamp;
```

This organization into structured arrays enhances the efficiency of data handling and analysis, providing a systematic approach to access and manipulate the different components of both orientation and heart rate datasets within the program.

3.2 Preprocessing

Current implementation supports only the first option stated in 2.2.2 section and redundant data filter.

3.3 Equations Evaluation on GPU

For computations on GPU was used Vulkan framework and inspiration from Slava Savenko. Evaluation is computed for every new equation in population over whole dataset. Due to it is not known the length of evaluated equation as stated in section 2.2.4, there are defined 3 compute shaders for addition, subtraction and multiplication. So the evaluation is done iterative method where the computation is defined as vector of first operand and second operand. So the vectors have length of whole dataset size. For better expleantion ve can look at implementation which is in `genetic_alg/Population.cpp` on line 247.

3.4 Parameters

The application takes several parameters to facilitate the processing of physiological data. The parameter

```
--data_path_hr
```

specifies the path to the file containing heart rate information, while the

```
--data_path_acc
```

parameter designates the path to the file with 3D accelerometer data. Optional parameters include

```
--not_use_gpu
```

to disable GPU usage and compute only on CPU,

```
--load_data_sequentially
```

to load data sequentially (if it not used, the thread for HR and thread for ACC is created), and

```
--gpu_workgroup_size <INTEGER>
```

to set the GPU work-group size. Additionally, users can define the range of equation coefficients with

```
--minimum_equation_coefficients <FLOAT>
```

and

```
--maximum_equation_coefficients <FLOAT>
```

The training process is controlled by the

```
--epochs <INTEGER>
```

parameter, and the initialization length of equations is set by

```
--maximum_equation_init_length <INTEGER>
```

These parameters collectively allow users to customize the program for their specific data and computational requirements, particularly when analyzing heart rate and accelerometer data.

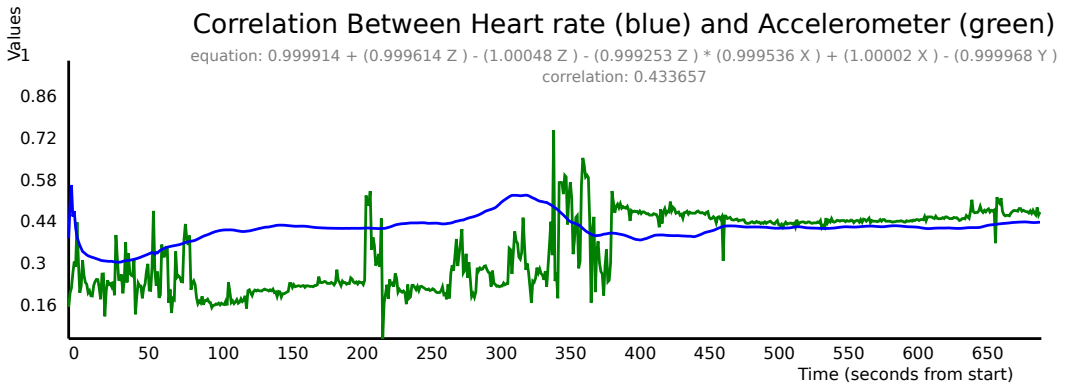

Figure 4: Example SVG output

4 Conclusion

To sum up, an implemented application fulfills the assignment to a considerable extent. The correlation values range between 0.2 and 0.7 (Pearson correlation), with the primary emphasis on speeding up the computation process.

In the code, various parallelization methods are employed. Threads are utilized to accelerate file loading, while vectorization is applied in multiple instances, such as normalization during the preprocessing phase. The evaluation of generated equations leverages GPU for its computational power.

The application is designed to process one patient (one pair of ACC and HR data) at a time. If there is a need to process multiple patients simultaneously, multiple processes can be initiated to read different files.

Discovering the Correlation Formula between 3D Accelerometer Data and Heart Rate

Prerequisites

- Download the relevant portions of data files - HR and ACC files from https://physionet.org/content/big-ideas-glycemic-wearable/1.1.2/ (HR and ACC)

- ## VulkanSDK-1.3.268.0

  Windows

```
VulkanSDK-1.2.182.1-Installer.exe --accept-licenses --default-answer --confirm-command install com.
```

- **linux (ubuntu jammy)**:

```
wget -qO- https://packages.lunarg.com/lunarg-signing-key-pub.asc | sudo tee /etc/apt/trusted.gp
sudo wget -qO /etc/apt/sources.list.d/lunarg-vulkan-1.3.268-jammy.list https://packages.lunarg.
sudo apt update
sudo apt install vulkan-sdk
```

- compile shaders

```
glslc plus.comp -o plus.spv
glslc minus.comp -o minus.spv
glslc multiply.comp -o multiply.spv
```

make sure you are using 64bit architecture on Windows

Usage

```
ppr(.exe) --data_path_hr <path_HR> --data_path_acc <path_ACC>
optional [
--not_use_gpu
--load_data_sequentially
--gpu_workgroup_size <INTEGER>
```

...