Guide

1 Program compilation and use of parameters

There is a *Makefile* file to build the project. Running the following order in a Unix shell the program should be compiled:

```
make POPULATION_SIZE=PS N_FEATURES=NF N_OBJECTIVES=NO
N_INSTANCES=NI
```

Where:

- ***PS*** is the number of individuals of the population. The population size must be 4 or higher.
- ***NI*** is the number of instances to use (rows) of the database. This number must be between 2 and the total number of instances of the database.
- ***NO*** is the number of objective functions. By default, the program is developed for two objective functions. To use only one or more than two, it is necessary to modify the "evaluation" module.
- ***NF*** is the number of features to use (columns) of the database. This number must be between 2 and the total number of features of the database.

The executable file, named *hpmoonCL* will be generated in the "*bin*" folder. For running it, in the shell the next order must be executed:

```
./bin/hpmoonCL –conf config.xml
```

Where *config.xml* is the necessary configuration file for the correct performance of the program, specified by the *–conf* option and located in the root folder of the project.

In addition, the user can indicate separately through line arguments each setting of the XML file. Table 1 summarizes the list of parameters and their possible values, and how to use them in the line of arguments. In any case, the special option *-h* displays the available options and examples of use.

Also, the *Makefile* file contains a rule to generate *Doxygen* documentation in the *"doc/html"* folder. This can be done by running the following command:

```
make documentation
```

Finally, the files and documents generated when compiling the project can be deleted. There are two types of cleaning depending on the content to be deleted. The command:

```
make clean
```

Deletes the following contents:

- **Binary files**
- **.o files**
- **~ files**
- **.a libraries**
- **Hypervolume project of Fonseca.** This project generates the necessary library for calculating the hypervolume.

For a complete cleaning, run the following command:

```
make eraseAll
```

Which will remove the same content as the previous command and also the following content:

- ***gnuplot* files.**
- **Documentation files** generated by *Doxygen*.

gnuplot files contain the fitness of the individuals in the first Pareto front and the necessary source code for the *gnuplot* program. If the user would generate a graph using *gnuplot* and the source code generated by the program, it will also be deleted when using this command.

2 The XML configuration file

The XML configuration file is required to run the program as well as the parameters used in the *make* command. The parameters of the XML file are read and used at runtime while the parameters used in the *make* command are read and used at compile time to avoid dynamic memory. The parameters are:

- ***DataBaseFileName*** is the name of the file containing the database.
- ***NGenerations*** is the number of generations to generate (iterations of the program).
- ***MaxFeatures*** is the maximum number of features initially set to "1". The value must be 1 or higher.
- ***TournamentSize*** is the number of individuals competing in the tournament. The value must be 2 or higher.

- ***DataFileName*** is the name of the file which will contain the fitness of the individuals in the first Pareto front.
- ***PlotFileName*** is the name of the file which will contain the *gnuplot* code for data display.
- ***ImageFileName*** is the name of the file which will contain the image data (graphic) after using the *gnuplot* command to generate it.
- ***DeviceType*** is the type of device that will run the program. The value must be *CPU* or *GPU*.
- ***CpuPlatformVendor*** is the name of the *CPU* device vendor that will run the *OpenCL* program.
- ***CpuDeviceName*** is the name of the *CPU* device model that will run the *OpenCL* program.
- ***GpuPlatformVendor*** is the name of the *GPU* device vendor that will run the *OpenCL* program.
- ***GpuDeviceName*** is the name of the *GPU* device model that will run the *OpenCL* program.
- ***WiGlobal*** is the total number of work-items (threads) that will run the program. This number must be 1 or higher.
- ***WiLocal*** is the number of work-items (threads) per compute unit that will run the program. This number must be a *WiGlobal* divider and less or equal than the maximum number of local work-items of the device.
- ***MaxIndividualsOnGpuKernel*** is the maximum number of individuals to be processed in a single execution of the kernel. The value must be 1 or higher.
- ***KernelsFileName*** is the name of the file containing the kernels with the *OpenCL* code.

The following table summarizes the restrictions of input parameters. The parameters passed to the *make* command are shown in uppercase. In lowercase, the parameters found in the XML configuration file.

PARAMETER	**RANGE**	**OPTION**
POPULATION_SIZE	4 <= ***PS***	-
N_INSTANCES	2 <= ***NI*** <= Number of instances of the DB	-
N_OBJECTIVES	***NO*** = 2	-
N_FEATURES	2 <= ***NF*** <= Number of features of the DB	-
DataBaseFileName	-	-db
NGenerations	0 <= ***NG***	-g
MaxFeatures	1 <= ***MaxF***	-maxf
TournamentSize	2 <= ***TS***	-ts
DataFileName	-	-plotdata

PlotFileName	-	-plotsrc
ImageFileName	-	-plotimg
DeviceType	CPU or GPU	-devt
CpuPlatformVendor	-	-vendor
CpuDeviceName	-	-devn
GpuPlatformVendor	-	-vendor
GpuDeviceName	-	-devn
WiGlobal	1 <= ***WG***	-wg
WiLocal	1 <= ***WL*** <= Maximum number of local work-item of the device ***WG*** % ***WL*** = 0	-wl
MaxIndividualsOnGpuKernel	1 <= ***MaxIndGPUKernel***	-maxind
KernelsFileName	-	-ke

Table 1. It shows the range of values of the input parameters and how to use them from the arguments line.

3 OpenCL optimization and limitations

The following points should be considered to obtain good performance when running the program:

1) The evaluation function for each individual has been parallelized with *OpenCL*. A compute unit is formed by *WiLocal* work-items and evaluates only one individual. Therefore, in *GPUs* it is recommended that *WiGlobal* has a value equal to the number of compute units of the device * *WiLocal*. *WiLocal* should be a multiple of 32 or 64 according to the device for improve the performance. The user can approximate the optimal value of WiLocal. The value is calculated as the number of stream processor or *CUDA* cores divided by the number of compute units. For example, the *GeForce GTX 770* has 1536 *CUDA* cores (*WiGlobal*) and 8 compute units, so 1536/8 = 192 local work-items, but sometimes it is better to increase this value, for example, 256. In this case, *WiLocal* = 256 and *WiGlobal* has 256 * 8 = 2048 work-items. The best combination is determined by the characteristics of the problem.
   In the *CPUs*, *WiGlobal* should have a value equal to the number of compute units (physical cores) and *WiLocal* must be set to 1. If another value is specified, it will be ignored.
2) The sort function according to the Pareto front, *nonDominationSort* contains one loop of quadratic order and are related to the number of individuals. For good quality results it is not necessary to increase the

number of individuals too. It's better to increase the number of iterations of the program (number of generations).

3) On *GPU*, the program gets better performance with values of *N_FEATURES* and *N_INSTANCES* higher than the number of local work-items. However, the database is stored in local memory and their capacity is very limited (approximately 49 KB depending on the device). So the program will abort if the database is too big.
4) The evaluation of all individuals is executed in a single kernel in the case of the *CPUs*. However, on *Nvidia GPUs*, a kernel has a certain time limit for its execution. If the number of individuals is high, the execution will fail. The *MaxIndividualsOnGPUKernel* parameter allows indicate the quantity of individuals which will be evaluated in a single execution of the kernel. Just reduce this parameter if the execution fails for this reason. The kernel is called iteratively to evaluate all individuals.