ONE-View ONE

How to generate and use the report ONE

Version 1.2

April 2018

1 Introduction

ONE-View is the MAQAO module in charge of leading all other MAQAO modules in order to produce reports aggregating results from all these tools. It automatizes the execution of other modules (such as LPROF or CQA) to generate reports in HTML pages or XLSX data sheets.

ONE-View offers several built-in reports. This document details one of them, called report ONE. This report combines both static and dynamic approaches to get an overview of the application performances. It uses MAQAO modules LPROF (a dynamic profiler) and CQA (a static code analyser).

2 Running ONE-View

To generate the report ONE using ONE-View:

```
$ maqao oneview –create-report=one –c=<config> [-xp=<dir>] [--format=<format>]
```

Name of the configuration file describing the experiment. Configuration file is described in next section

Report format:
- XLSX (default)
- HTML
- all (all existing formats)

Name of the directory created by ONE-View. If it is not specified, the directory is called *maqao_YYYY-MM-DD_hh-mm-ss.* It is referred in this file as *experiment directory*.

To list all options for ONE-View:

```
$ maqao oneview --help
```

3 Filling the Configuration File

3.1 Creating the configuration file

To generate a template of configuration file:

```
$ maqao oneview –create-config[=<file>]
```

Name of the generated configuration file. If it is not specified, the configuration file is called *config.lua* and it is created in the current directory.

The template contains all available fields and it is fully documented. This document details all fields in next subsections. The configuration file uses the Lua syntax; in particular, commented lines are prefixed with "--" (two dashes).

3.2 Main fields

- *binary*: Path to the binary to analyze
- *dataset*: Path to a directory containing the application dataset. If filled, this directory is copied into the experiment directory so it must be as small as possible and the experiment directory must not be created in the dataset directory.
- *run_command*: A string detailing how the binary must be run. In this string, the binary is referred as *<binary>*. This substring is automatically replaced by the correct binary name when ONE-View needs to run any version of the binary.
  If the application does not need any parameter, the field has "*<binary>*" as value. If the application needs two parameters, *-a=5* and *--b*, the field has "*<binary> -a=5 --b*" as value.
- *run_directory*: A string detailing where the binary should be run. Some applications must be run from a specific directory, most of the time a directory related to the dataset directory. This field is used to specify this path. The substring *"<dataset>"* can be used to represent the path to the dataset directory located in the experiment directory and it is automatically substituted by the real path by ONE-View during runs.
- *mpi_command*: A string detailing the MPI command to use to run the application. If MPI should not be used, this string must be empty. If MPI is used, this field must contain the call to *mpirun* or *mpiexec* with all its parameters, except the application and its own parameters.

For example, if a binary needs the following command to be run:

```
$ mpirun –n 4 ./my_app 250 –output=./log.out
```

The corresponding configuration file contains:

```
binary = "./my_app"
run_command = "<binary> 250 –output=./log.out"
mpi_command = "mpirun –n 4"
```

- *omp_num_threads:* A number specifying how many OpenMP threads must be used for the experiment. If the number is greater than one, the environment variable OMP_NUM_THREADS is defined before executing the application.
- *sbatch_script:* Path to a script used by Slurm. Slurm is a tool used to manage clusters. This variable can be set to specify which script will be passed to Slurm through the command sbatch in order to run the application on the cluster. The script must be adapted to ONE-View by using the code *<run_command>* instead of the classic binary execution command. For example, a sbatch script adapted for ONE-View can be:

```
#! /bin/bash

#SBATCH SETTINGS
#SBATCH –J myJob
…

#APPLICATION SETTINGS
module load …
export MY_VAR …

#RUN THE APPLICATION
# mpiexec –n 16 ./my_app
<run_command>
```

3.3 Secondary fields

- *external_libraries*: An optional table describing dynamic libraries to analyze in addition of the binary. Most of the time, linked libraries are system libraries that cannot be easily optimized so it is not interesting to analyze them but sometime, the application to optimize is not an executable but a library. Each entry in the table is a string with the name of a library to analyze.
- *source_code_location*: An optional string detailing where the source code is located. It is needed to localize the source code of your application if it is not at location defined in debug data (which is set when compiling the application).
- *job_submission_threshold*: An optional string representing a time matching the following regular expression: ***(%d**h**)?(%d**m**)?%d**s*. Some clusters use an automatic submission system where all executions are deported on other cores. If greater than zero second, this time indicates to ONE-View how many time it should wait dynamic runs before stopping its process.
- *maximal_path_number*: A number indicating the maximal number of paths in the control flow graph a loop can have. If a loop has too many paths, it is not analyzed.
- *excluded_areas*: A table describing areas (loops or basic blocks) which must be skipped from analysis. The table is a set of sub tables containing three fields: *type* which can be either *loop* or *block*; *id* which is the identifier of the area to exclude; *module* which is either *binary* (the default value) when the area is in the main binary or an external library name (as defined in the field external_libraries).

The following fields are not used by report ONE and reserved for future releases:

- *filter*
- *freqencies*
- *profile_start*
- *additional_hwc*

- *decan_multi_variant*
- *is_sudo_available*

4 Reading Report ONE

Reports are generated in *<experiment_directory>/REPORTS/* as *<report>.<format>*, where *<report>* is the value of the parameter *–create-report* and *<format>* the value of the parameter *format*.

4.1 XLSX Output

XLSX files can be read by several softwares: Microsft Office Excel, LibreOffice, OpenOffice. The file contains several tabs described in next subsections.

4.1.1Experiment_Summary

The tab **Experiment_Summary** presents some parameters about the experiment and the machine used in a first table, and in a second table, some metrics summarizing the application performances. Metrics are highlighted from green to red to signal best to worse performance.

4.1.2Application_Categorization

The tab **Application_Categoriation** presents a chart detailing in which categories the time is spent.

An example is shown by figure 1. In the example, most of the time is spent in the binary and there is no time spent in OpenMP or MPI.

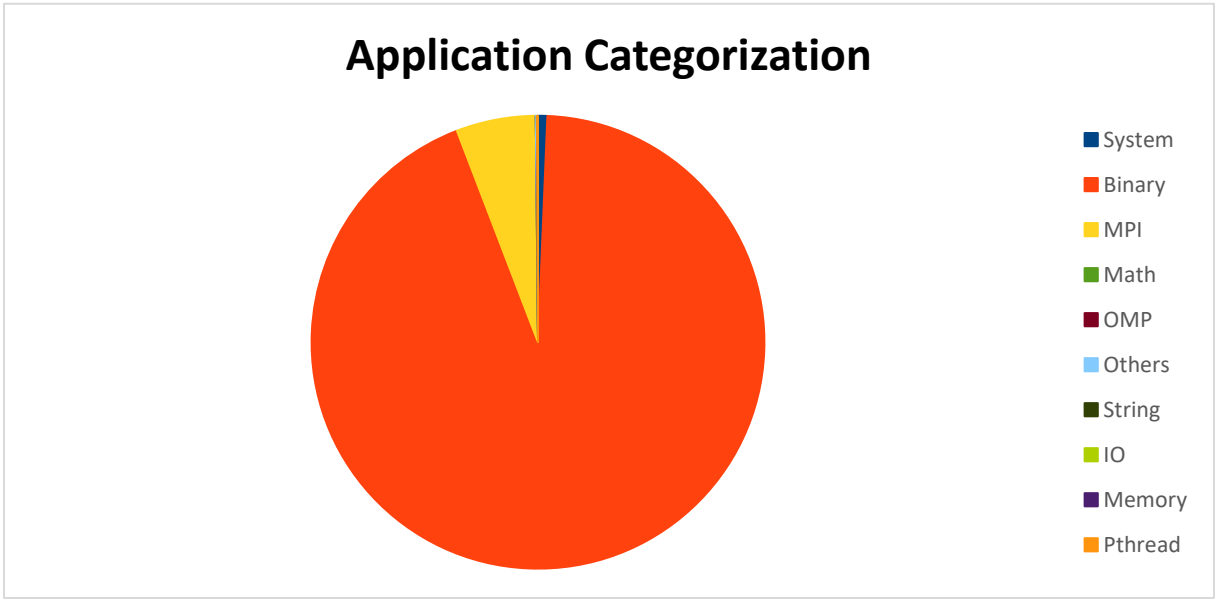

Figure 1: Application Categorization Chart

4.1.3Functions_Based_Profile

The tab **Function_Based_Profile** presents a chart with a profile of the application at the function level, grouping functions by their coverage.

The figure 2 presents an example of the chart. On the example, there are five functions whose coverage is greater than eight percent of the application and four functions whose coverage is between eight and four percent. Functions whose coverage is greater than four percent represent about seventy two percent of the total time of the application.

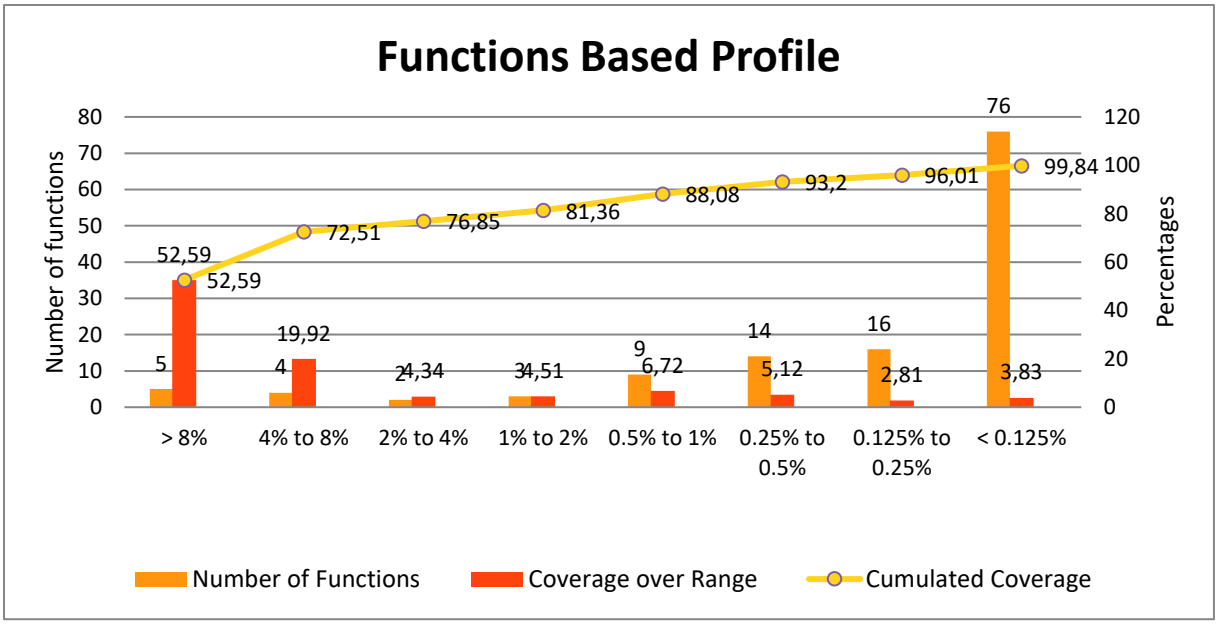

Figure 2: Function Based Profile Chart

4.1.4Functions_Listing

The tab **Functions_Listing** presents a full listing of all profiled functions, shorted by coverage.

4.1.5Application_Profile

The tab **Application_Profile** presents a chart with a profile of the application at the loop level, grouping functions by their coverage. The profile uses only innermost loops. An innermost loop is a loop which does not contain any loop.

Figure 3 presents an example of the chart. On the example, there is no innermost loop whose coverage is greater than eight percent and only three innermost loops whose coverage is between four and eight percent. All innermost loops represent about seventy four percent of the application time.

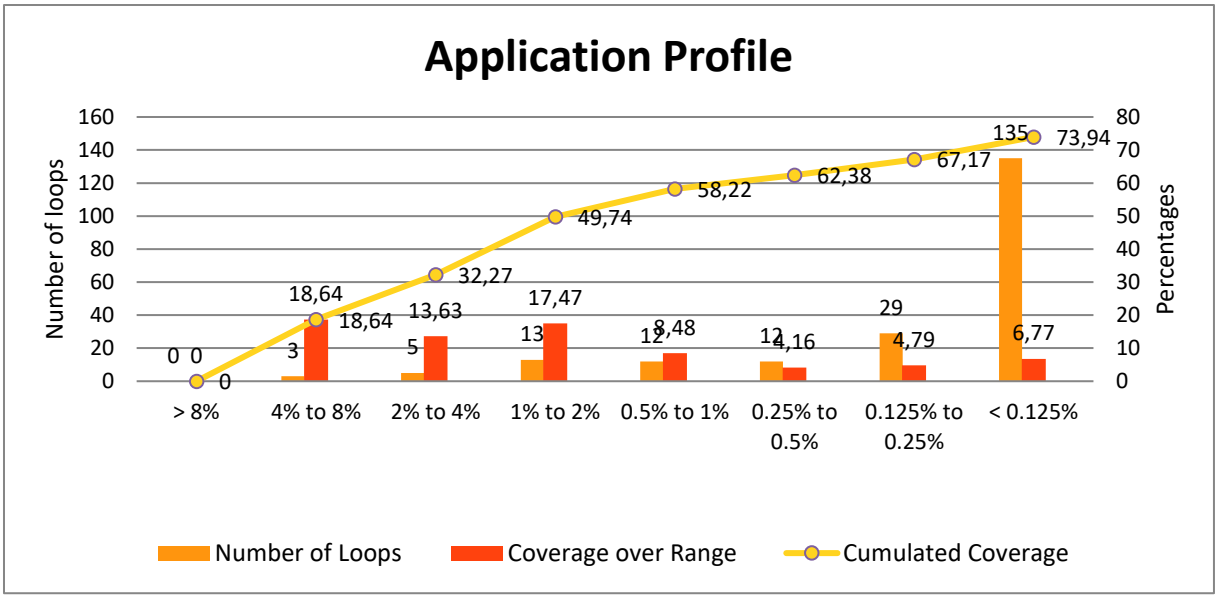

Figure 3: Application Profile Chart

4.1.6 Application_Loops_Profile

The tab **Application_Loops_Profile** presents a chart with another profile of the application at the loop level, grouping functions by their coverage. It is more detailed than **Application_Profile** as it uses loops of any level (outermost, in between, innermost). An outermost loop contains at least another loop and an in between loop contains at least one loop and is contained in another loop.

Figure 4 presents an example of the chart. On the example, loops whose coverage is between four and eight percent are innermost loops and there is about ninety one percent of the time spent in loops of any level.

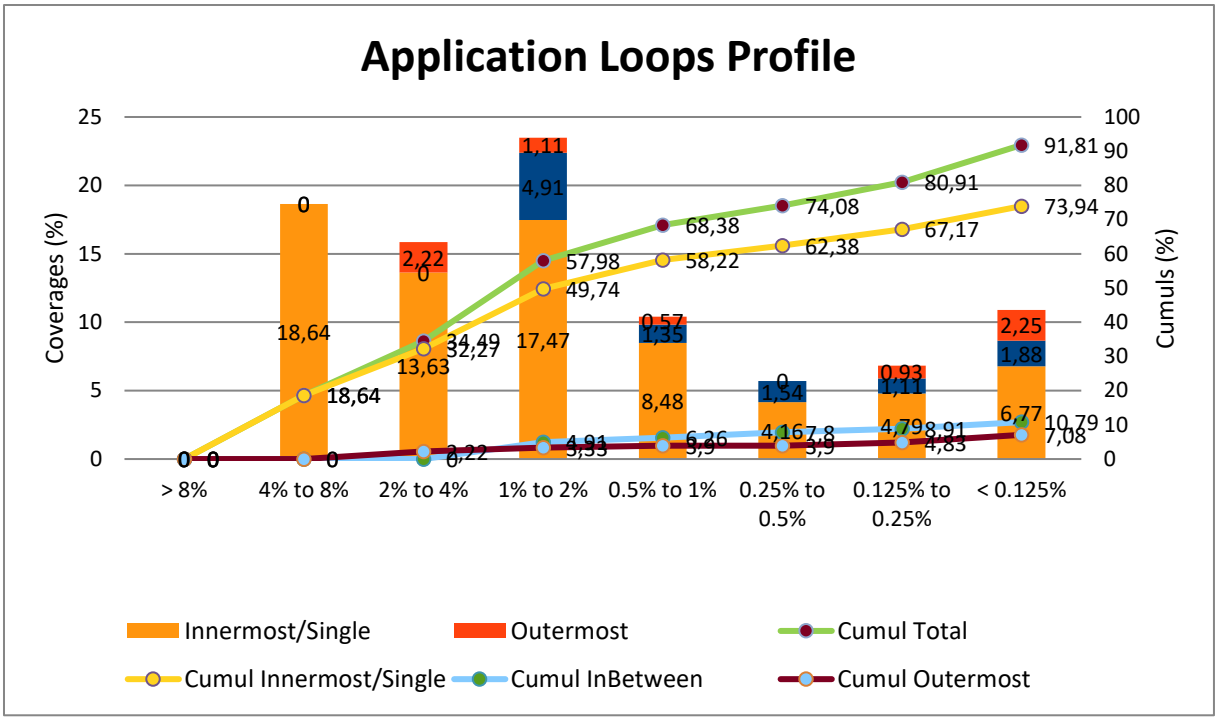

Figure 4: Application Loops Profile Chart

4.1.7Optimization_Report

The tab **Optimization_Report** presents a table containing advanced data from CQA for each path of each analyzed loop.

4.1.8Potential_Speedups

The tab **Potential_Speedups** presents several charts detailing potential speedups obtainable under specific conditions, according to the number of loops modified to satisfy the condition. These charts are ROI oriented (return on investment). Charts present speedups obtainable if the assembly code is modified to satisfy some conditions:

- **If Code Clean** means that all instructions which do not perform floating-point computation or memory accesses are deleted. Instructions used to handle the loop control flow are not included in the instructions set to remove;
- **If FP Vectorized** means that all instructions performing floating-point computation are vectorized;
- **If Fully Vectorized** means that all instructions performing floating-point computation and all memory accesses are vectorized.

There are two types of charts:

- Summary charts where the X axis represents the number of loops to modify in order to obtain the speedup. An example is shown by figure 5;
- Ordered charts where the X axis represents loops identifiers to modify in order to obtain the speedup. An example is shown by figure 6.

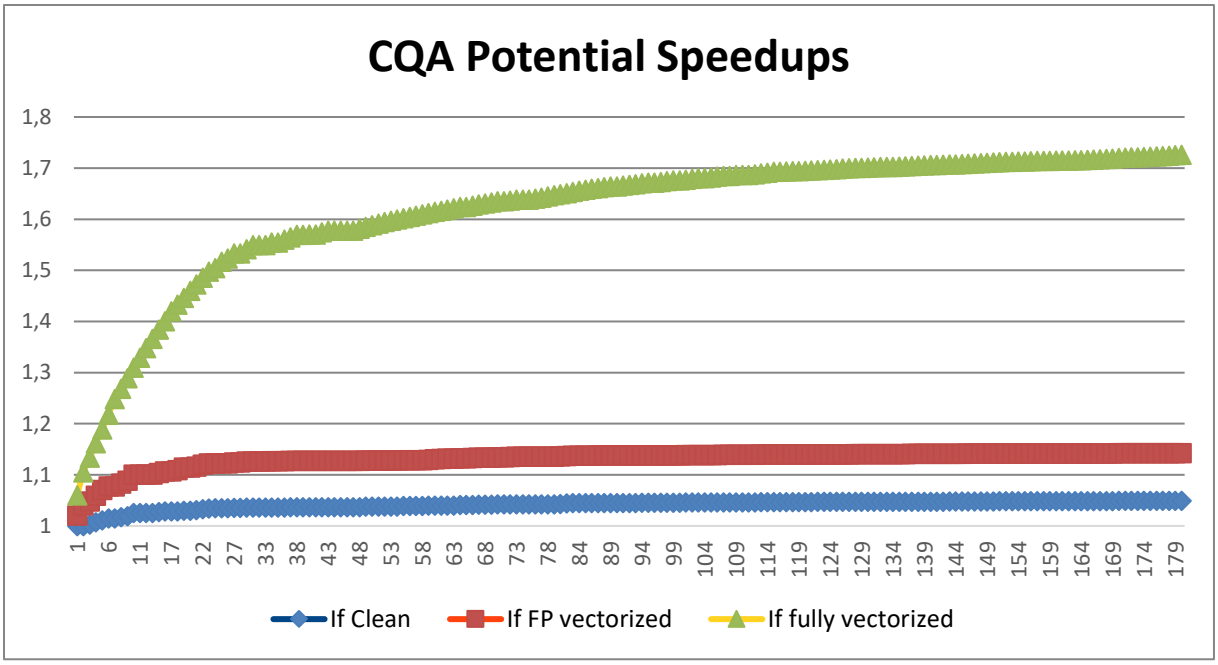

Figure 5: CQA Potential Speedups Summary Chart

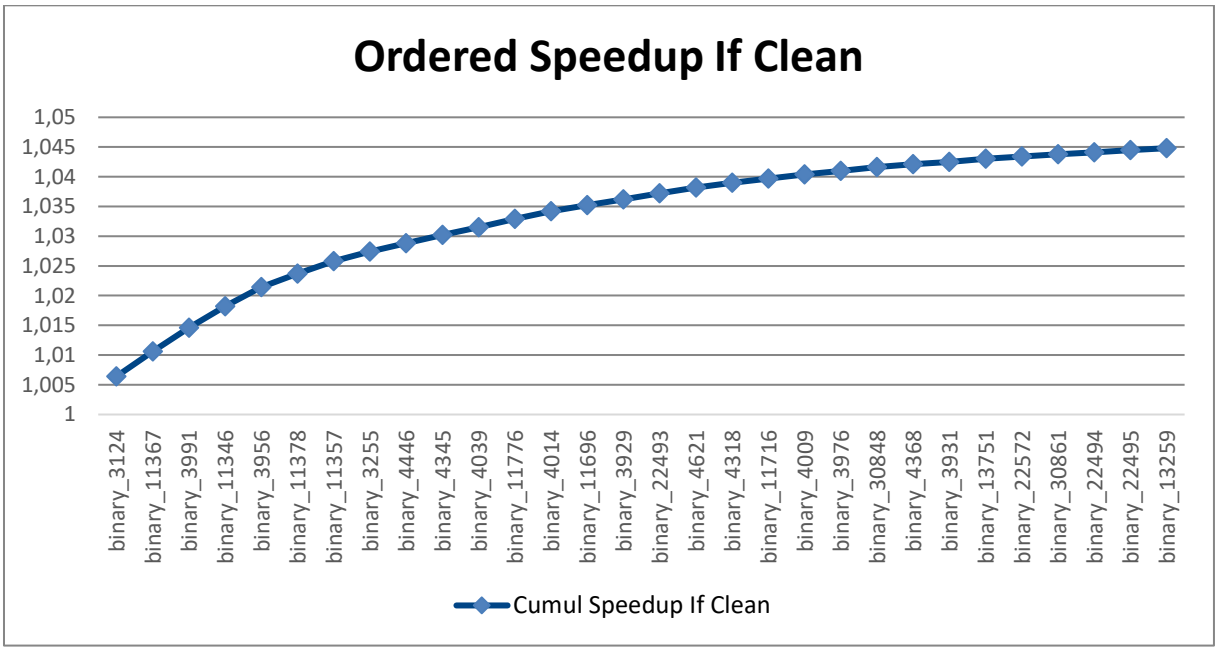

Figure 6: CQA Potential Speedup Ordered Chart

4.2 HTML Output

HTLM reports can be read by any internet browser. The main file is **index.html**, located in *<experiment_directory>/RESULTS/<report>_html/*. All tabs have a menu located at the top of the tab which can be used to navigate between tabs. All tabs are described in next subsections.

4.2.1Index

The file **index.html** is the report index and it presents several sections. One presenting some parameters about the experiment and the machine used in a first table. A second one presents some metrics summarizing the application performances. Metrics are highlighted from green to red to signal best to worse performance. Next sections present several charts detailing potential speedups obtainable under specific conditions, according to the number of loops modified to satisfy the condition. Charts present speedups obtainable if the assembly code is modified to satisfy some conditions:

- **If Code Clean** means that all instructions which do not perform floating-point computation or memory accesses are deleted. Instructions used to handle the loop control flow are not included in the instructions set to remove;
- **If FP Vectorized** means that all instructions performing floating-point computation are vectorized;
- **If Fully Vectorized** means that all instructions performing floating-point computation and all memory accesses are vectorized.

There are two types of charts:

- Summary charts where the X axis represents the number of loops to modify in order to obtain the speedup. An example is shown by figure 5;

Ordered charts where the X axis represents loops identifiers to modify in order to obtain the speedup. These charts can be opened or collapsed by clicking on the header. An example is shown by figure 6.

These charts allow to know if the application can be optimized and what speedup can be achieved

4.2.2Application

The **Application** tab shows several charts. The first chart details in which categories the time is spent as presented by figure 1. Two next charts present profiles of the application. The first one is at the function level and groups functions by their coverage as presented by figure 2, the second one is at the loop level and groups loops by their coverage as presented by figure 3.

4.2.3Functions

The **Functions** tab presents a profiling of the application at the function level, listing all used functions with their coverage. By clicking on the arrow on the left of any functions, the box can be opened to reveal all profiled loops belonging to the function represented as a tree. Loops can also be opened by clicking on the left arrow. If a loop has a circle instead of an arrow, it means it is an innermost loop. All coverages are global to the application.

By right-clicking on a row (either loop or function), a chart presenting the load balancing between processes is displayed.

By double-clicking on an innermost loop, a new tab presenting all results for the loop is opened in the browser. Details about this tab are described in the subsection **Loop**.

4.2.4 Loops

The **Loops** tab presents a profiling of the application at the loop level, listing all analyzed loops. For each loop, there is the MAQAO identifier, data about the location in the source code and the coverage with a color associated to it. The color is red when the loop is hot and it goes to blue when the loop is cold.

By clicking on a loop, a new tab presenting all results for the loop is opened in the browser. Details about this tab are described in the subsection **Loop**.

4.2.5Loop

The tab **Loop** is not accessible from the menu, but only from tabs **Functions** and **Loops**. This tab contains all available data about a specific loop:

- The location in the source code and source code of the loop if possible;
- The assembly code of the loop in a box that can be opened by clicking on the header;
- The full CQA report associated to the loop. More details about CQA are available in the CQA tutorial available at http://maqao.org/release/MAQAO.Tutorial.CQA.intel64.pdf;
- A subset of CQA advanced metrics in a box that can be opened. These metrics can be used by experts to analyze the loop behavior;
- An analysis of memory groups in a box that can be opened. A memory group is a set of instructions accessing to the same memory area. Each color represents a memory group, such as the number between brackets at the end of assembly lines. Data about the group appears in a tooltip when the cursor is placed over the instruction.

4.2.6Help

The **Help** tab presents all features of the report. Each section can be opened or collapsed by clicking on it.