# ONE-View ONE

## How to generate and use the report ONE

**Version 1.4**

**October 2018**

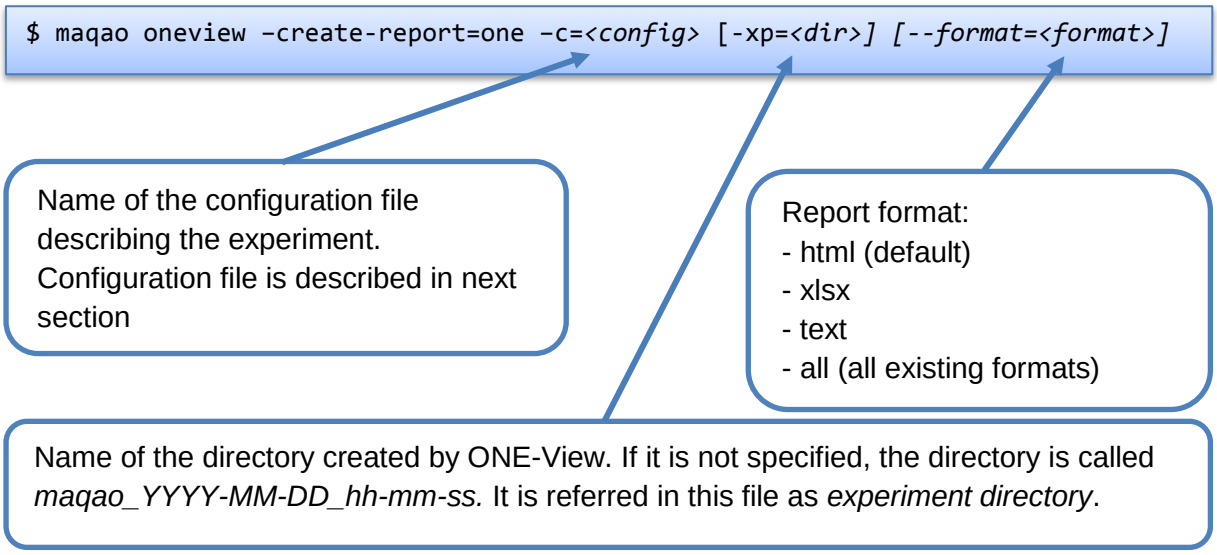
# 1 Introduction

ONE-View is the MAQAO module in charge of driving 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 performance. 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, the default command is:




it is also possible to provide the necessary parameters to ONE-View from the command line. It can be used when a small set of variables in the configuration file are needed. All existing parameters for the configuration of the experiment are available in sections 3.2 and 3.3.

```
$ maqao oneview –create-report=one –binary=./myapp -run_command=”<binary> -opt”
```

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 analyse
- *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. 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.
- *batch_command:* When the cluster uses a batch manager, this variable describes how to use it. If a script is needed, it must be referred as *<batch_script>.*
- *batch_script:* Path to a script used by a batch manager. The script must be adapted to ONE-View by using the code *<run_command>* instead of the classic binary execution command. For example, a batch script adapted for ONE-View for SLURM 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 analyse 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 analyse them but sometimes, 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 analyse.
- *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 the location defined in debug data (which is set when compiling the application).
- *maximal_path_number*: A number indicating the maximal number of paths in the control flow graph a loop can have. Loops with a greater number of paths will not be analysed.
- *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).
- *number_nodes:* Number of nodes to use to run the application on the cluster. Can be referred as *<number_nodes>* in the *batch_script* or the *mpi_command* fields.

- *number_processes:* Number of processes to use to run the application. Can be referred as *<number_processes>* in the *batch_script* or the *mpi_command* fields.
- *number_tasks_nodes:* Number of tasks per nodes to use to run the application. Can be referred as *<number_tasks_nodes>* in the *batch_script* or the *mpi_command* field.

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: Microsoft Office Excel, LibreOffice, OpenOffice. The file contains several tabs described in next subsections.

### 4.1.1 Experiment_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.

### 4.1.2 Application_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.

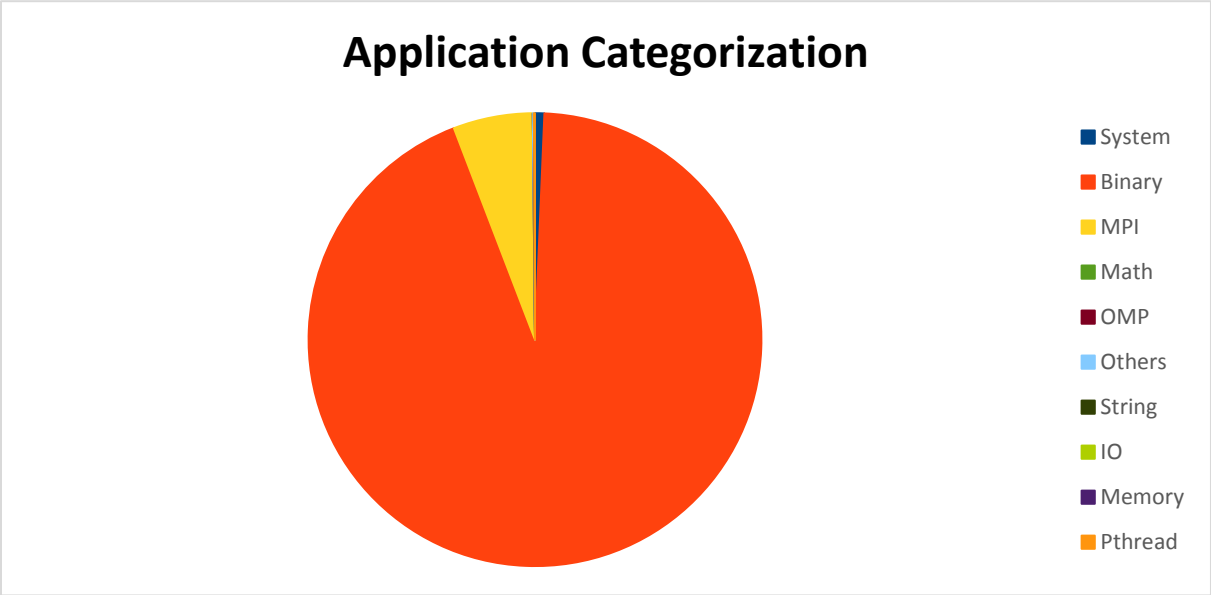



**Figure 1: Application Categorization Chart**

### 4.1.3 Functions_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.

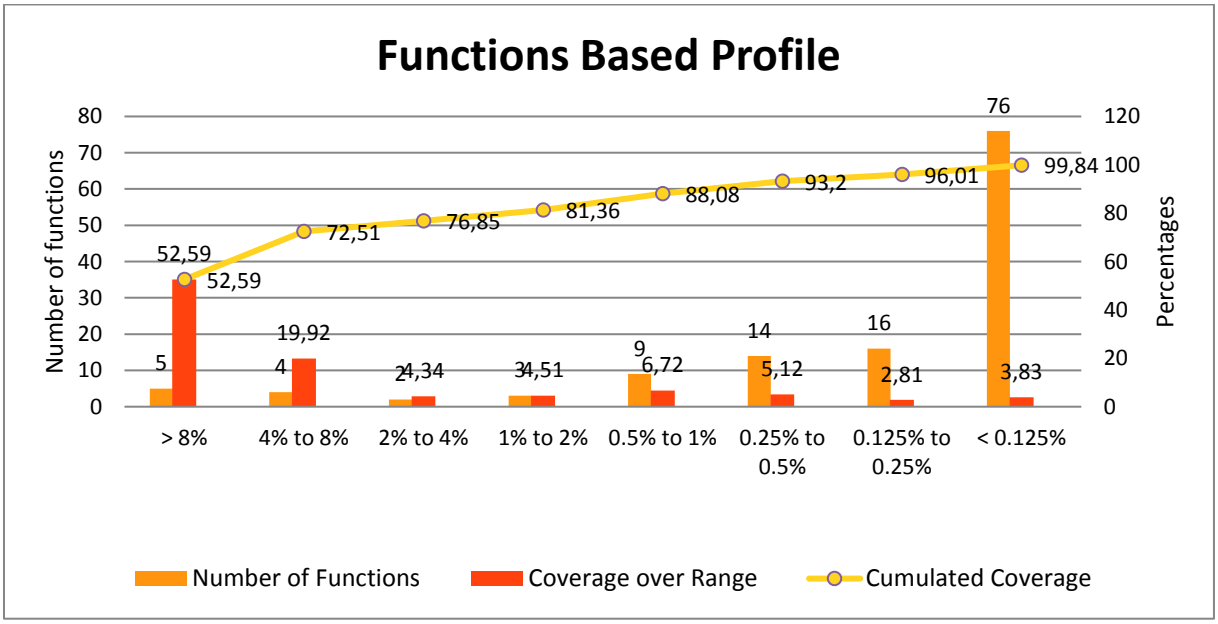



**Figure 2: Function Based Profile Chart**

### 4.1.4 Functions_Listing

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

### 4.1.5 Application_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.

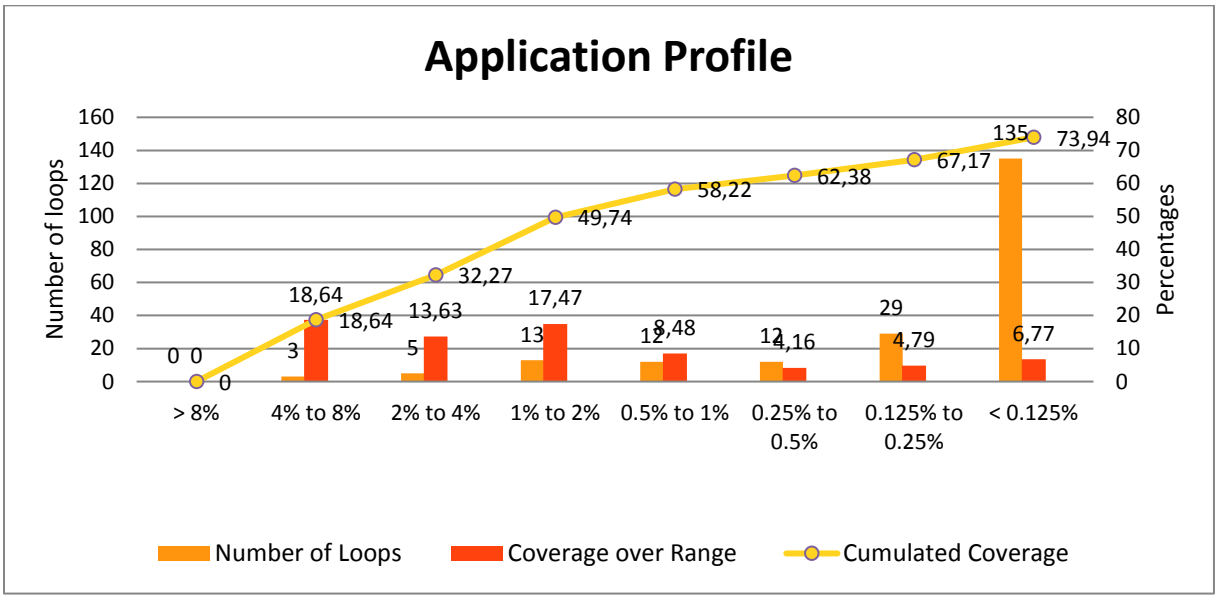



**Figure 3: Application Profile Chart**

### 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.

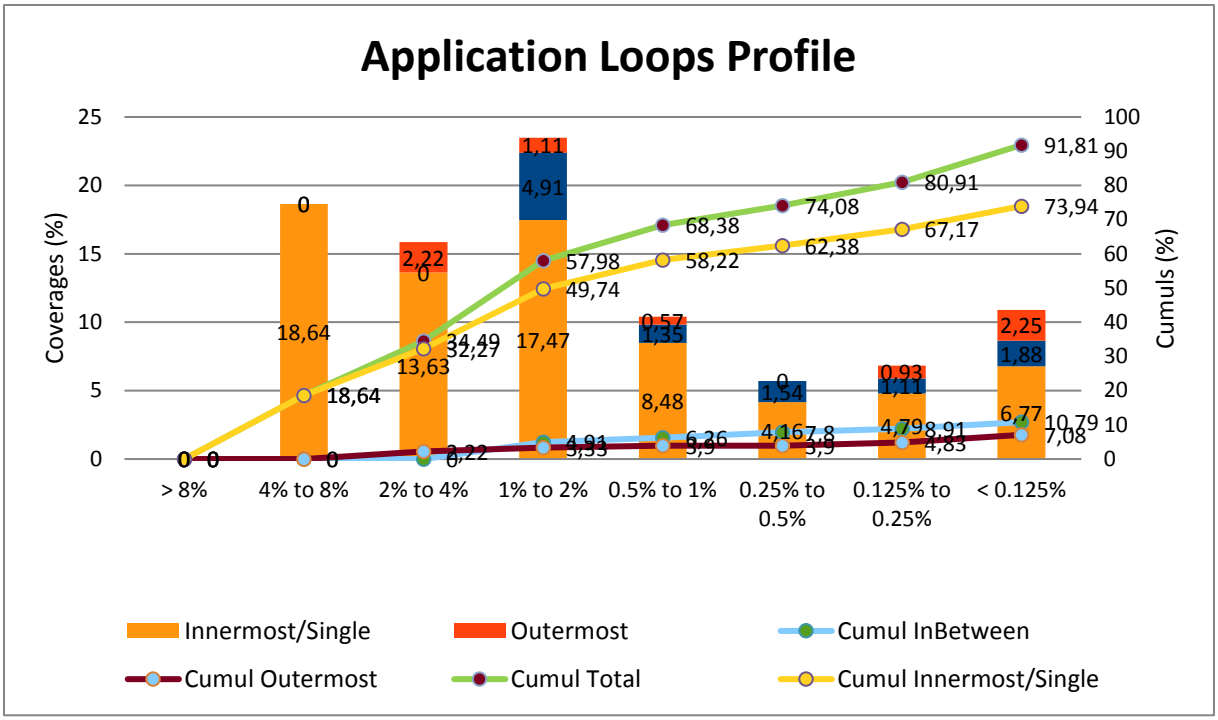



**Figure 4: Application Loops Profile Chart**

## 4.1.7 Optimization_Report

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

## 4.1.8 Potential_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.

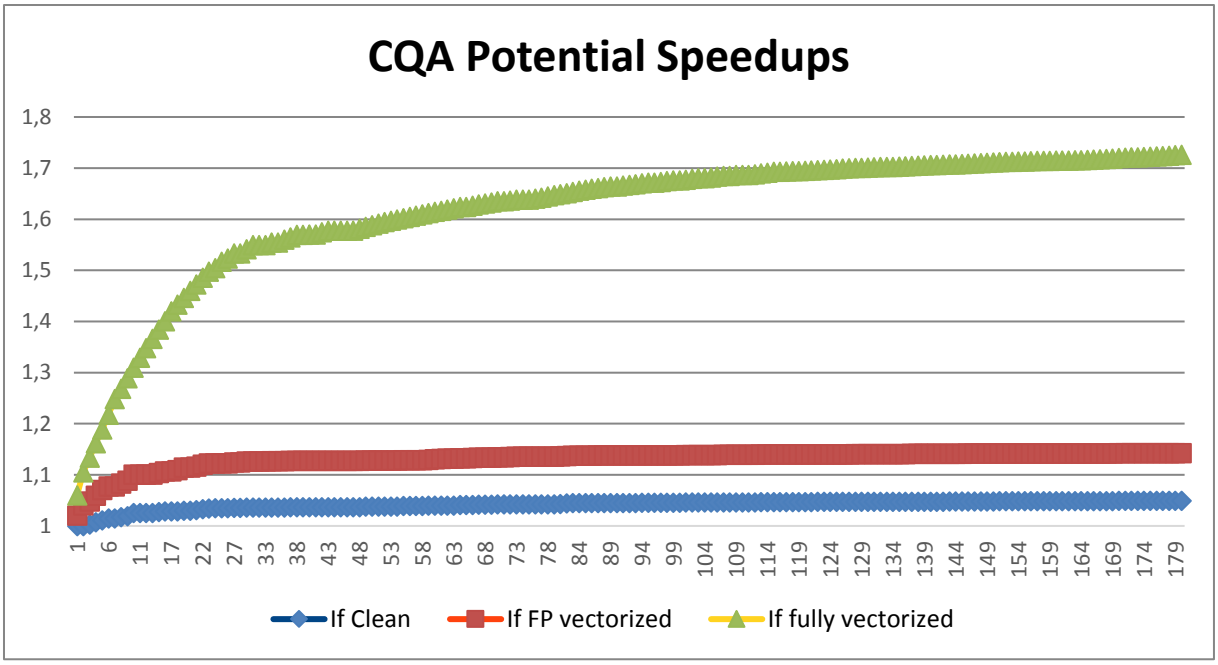



**Figure 5: CQA Potential Speedups Summary Chart**

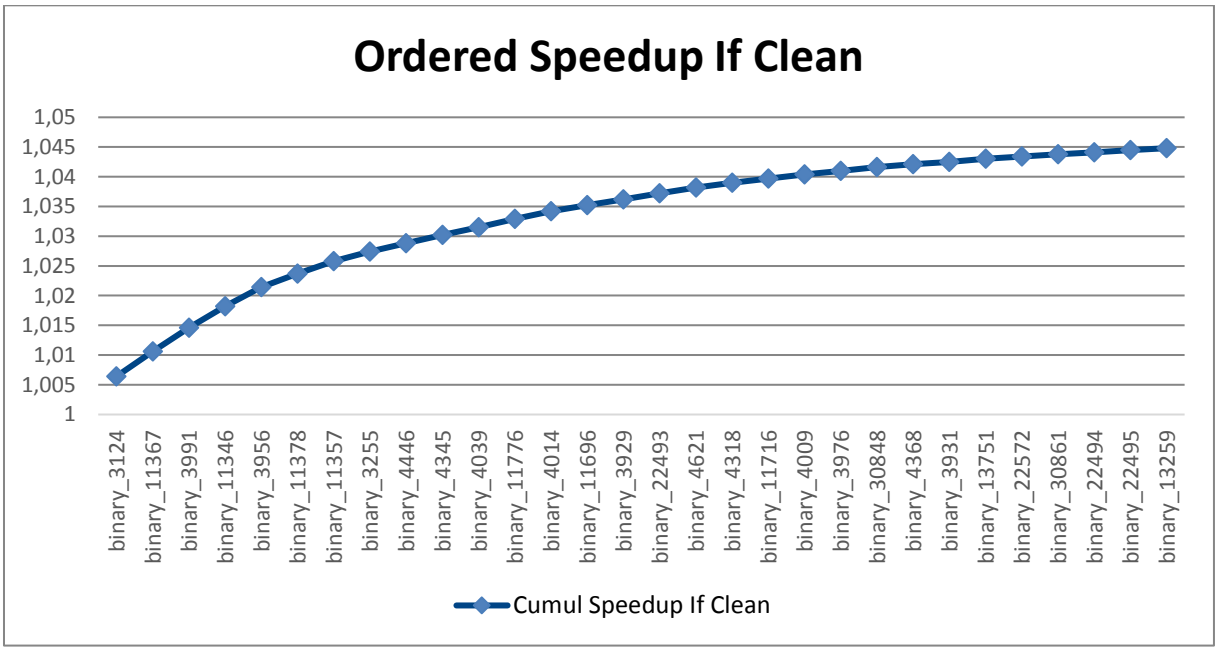



**Figure 6: CQA Potential Speedup Ordered Chart**

## 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.

### 4.2.1 Global

The file **index.html** is the report index and it presents four sections:

- **Experiment Summary** describes some parameters about the run
- **Configuration Summary** describes parameters from the configuration file
- **Global Metrics** presents some metrics summarizing the application performances. Metrics are highlighted from green to red to signal best to worse performance. Each metric is described by a tooltip appearing when the cursor is above the metric name. Metrics *Clean*, *FP Vectorised* and *Fully Vectorised* can be clicked to change the chart according to the selected metric. These metrics detailed some 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.

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

### 4.2.2 Application

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.

### 4.2.3 Functions

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 a function or a loop, a new tab presenting all results for the loop is opened in the browser. Details about loop tabs are described in the subsection **Loop** and details about function tabs are described in the subsection **Function**.

### 4.2.4 Function

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

- Its source code if available
- Its CQA report. More details about CQA are available in the CQA tutorial available at http://maqao.org/release/MAQAO.Tutorial.CQA.intel64.pdf
- The function loop hierarchy with links to each loop report

### 4.2.5 Loops

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

Additional columns can be displayed by checking to corresponding box just above the table.

By clicking on a column header, the table is sorted according to this column.

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**.

### 4.2.6 Loop

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 source code and the assembly code on a box on the left. Codes can be switched using the box header. The grouping analysis can be displayed on the assembly code by clicking on a button. A memory group is a set of instructions accessing to the same memory area. Each colour 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.
- A static analysis on the right box. The main static analysis is the CQA analysis. More details about CQA are available in the CQA tutorial available at http://maqao.org/release/MAQAO.Tutorial.CQA.intel64.pdf. A second static analysis contains advanced metrics from CQA. Each static analysis is computed for an assembly path that can be selected using the menu on the top of the box.

## 4.3 Text Output

The text report is displayed on the terminal. It can be customized using several options:

- *--text-global [=on/off]*: Display *Global* section if parameter is *on* (default), else do not display it if *off.*
- *--text-application [=on/off]*: Display *Application* section if parameter is *on* (default), else do not display it if *off.*
- *--text-functions [=on/off]*: Display *Functions* section if parameter is *on* (default), else do not display it if *off.*
- *--text-functions-full [=on/off]*: Display all data for *Function* section if parameter is *on* (default), else do not display it if *off.*

- *--text-loops [=on/off]*: Display *Loops* section if parameter is *on* (default), else do not display it if *off.*
- *--text-loops-full [=on/off]*: Display all data for *Loops* section if parameter is *on* (default), else do not display it if *off.*
- *--text-cqa [=on/off/[module:]id1, [module:]id2]*: Display *CQA* section if parameter is *on* (default), else do not display it if *off.* Analysed loops can be filtered by giving for each loop its module (*binary* (default) or an entry in *external_libraries*) and its MAQAO identifier.
- *--text-cqa-full [=on/off/[module:]id1, [module:]id2]* Display all data for *CQA* section if parameter is *on* (default), else do not display it if *off.* Analysed loops can be filtered by giving for each loop its module (*binary* (default) or an entry in *external_libraries*) and its MAQAO identifier.

Default output display sections *Global*, *Application*, *Functions*, *Loops* and *CQA*.

Text report sections are similar to corresponding HTML sections. CQA section is CQA reports of selected loops.