RADAR GUI applications
user guide
version 1.0

IT4Innovations
national01$#&0
supercomputing
center@#01%101

Czech Republic
January 18, 2019

Contents

1 Introduction

This document presents how to analyze data produced by MERIC library using RADAR tool and its GUI applications - *RADAR configurator* and *RADAR visualizer*.

RADAR can be run without any GUI, it does MERIC's output data analysis and provides MERIC's configuration file for future production runs of the optimized application and LaTeX report describing the application behavior when different tested configurations were applied. Unfortunately RADAR requires very complex configuration file to specify. *RADAR configurator* is a graphical application that can be used to overcomes the problem and generate the configuration file very easily.

The repository also contains *RADAR visualizer*, another graphical application that creates another layer above RADAR. It runs the RADAR underneath and provides interactive graphic elements to show the application behavior in much more synoptic way than the LaTeXreport does.

MERIC repository: https://code.it4i.cz/vys0053/meric
RADAR repository: https://code.it4i.cz/bes0030/readex-radar
RADAR GUI apps repository: https://code.it4i.cz/vys0053/SGS18-READEX

1.1 RADAR tools requirements

RADAR itself is written in Python3 using scikit-learn module for mathematical analysis. Also *RADAR configurator* and *RADAR visualizer* are as well as the RADAR itself based on Python3 together with PyQt5 GUI toolkit. In order to provide various graphic representations of the data, the applications has several dependencies. Complete list of dependencies contains:

- Python3
- scikit-learn
- PyQt5
- seaborn – used to generate heatmaps
- matplotlib – used to generate region's graphs
- pydot (part of the repository) – used to generate a graph of regions' callpath
- pyEd (part of the repository) – used to generate a graph of regions' callpath in yEd graph format [1]

To settle connection with RADAR, it is necessary to create new file named *"pathToRadar.json"* in the RADAR GUI repository root directory. The file must specify path to RADAR repository in the following format (with the quotation marks).

```
{
        "pathToRadar": "/path/to/RADAR/"
}
```

[1] yWorks' graph editor yEd can be download from: https://www.yworks.com/products/yed

2 *RADAR configurator*

As already briefly presented, the *RADAR configurator* provides simple way how to create RADAR configuration file for specific MERIC output data. All the features of the application presents this Section.

To use *RADAR configurator* user should run "`runRadarGUI_config.py`", that will show similar dialog as in the Figure 1.

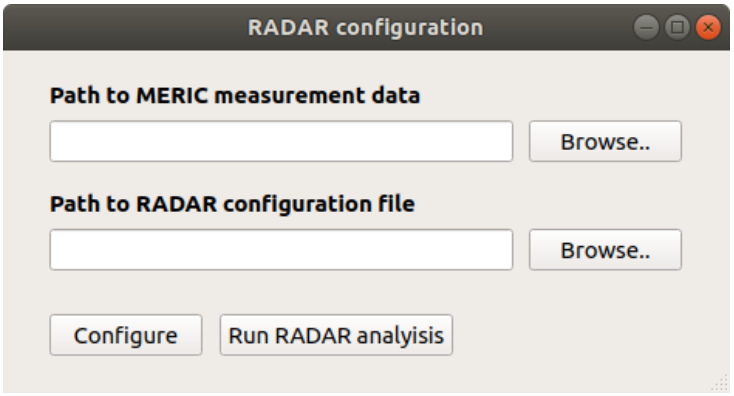


Figure 1: *RADAR configurator* startup dialog

There are three ways how to proceed from this dialog:

- Enter path to MERIC measurement data and click on button `Configure` to create a new configuration file for the specified data.
- Enter path to RADAR configuration file to either edit existing configuration file or to directly run RADAR analysis.
- Enter both a data path and a configuration file path to edit the existing configuration file to fit the data provided. Use `Configure` button to proceed.

When running RADAR with an existing configuration file, the progress is written into the command line and the RADAR LaTeXreport is generated, optionally together with the MERIC configuration file.

Now we will focuse on the process of the RADAR configuration file creation. Window of the *RADAR configurator*consits of three tabs `Regions` (Figure 2), `Data parameters` (Figure 4) and `Aditional options` (Figure 5).

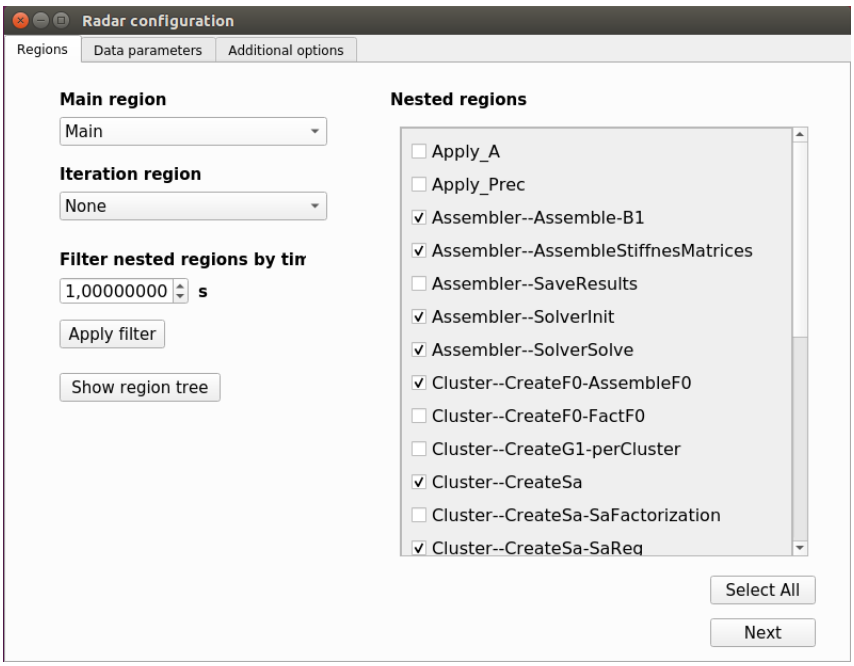


Figure 2: *RADAR configurator* tab – Regions

In the Regions tab a user must specify one Main region, a region that covers all the nested regions that will be evaluated. Next step is selection of the nested regions for the analysis. For a quick selection of the nested regions a filtering by time can be used. If none of the nested regions is selected the filter will select all regions with run time higher than specified, otherwise the filter will be applied at the list of already selected regions and will remove regions, that last shorter time than specified.

To see region's callpath graph structure (together with information about the regions run times) use `Show region tree` button. Example of such graph is shown in Figure 3. This window allows to store the graph as an image or into yEd graph format.

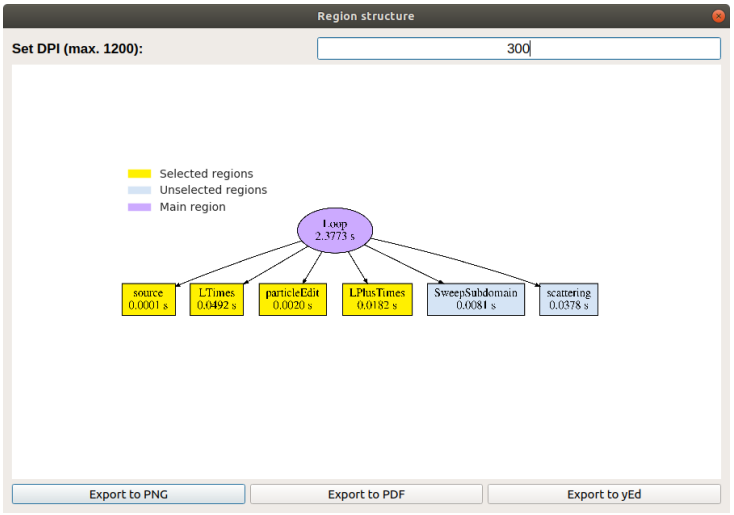


Figure 3: Graph of region's callpath structure

In the second tab `Data parameters`, shown in Figure 4, we specify which of the measured data we want to evaluate and how the RADAR should work with them. The `y label` part allows us select which of the captured energy, time or performance counters should be included into the analysis. We recommend not to include all the data, since the analysis may take very long and the final report may become very long and not very synoptic.

In the `Parameters` part of this tab, we must give name each of the parameter we changed in the application measurement time. Names of the default system parameters should be already filled-in, based on the information from the *measurementInfo.json*, which is located in the measured data directory. Besides the names, also role of each parameter must be specified, There are 4 types of a role:

- `xLabel`: parameter that we want to present on the x-axis of the graph in the analysis report. The parameter must be integer.
- `funcLabel`: parameter that we want to present on the y-axis of the graph in the analysis report. The parameter must be integer.
- `key`: another parameter, that we changed during the analysis.
- `config`: the parameter does not influence measured values, e.g. name of the computational node.

Always one `xLabel` and `funcLabel` must be specified. Since all the parameters values are strings or integers, we may apply on `xLabel` and `funcLabel` (the only parameters that we are sure that the type is integer) a multiplier, that will change the parameter unit. This way in the Figure 4 the multiplier 0.1 change the unit from 0.1 GHz to GHz.

The default value of each parameter is in the analysis considered as the configuration for evaluation of the savings.

Some energy measurement systems does measure CPUs, memories but not cover the computational node itself. In that case we may add so-called baseline, static power consumption of the parts not included in the measurement, for each measured value to approximate complete node energy consumption. To be able do such calculation, user must specify, for which source of energy and time measurement want to apply the baseline. This option is enabled by `Time-energy variables` check-box. Even if no baseline is applied, the information about time and energy measurement source are used in the analysis to evaluate how the runtime will change when the dynamic switching to the optimal energy-saving configuration is applied. In that case 0 W baseline should be used.

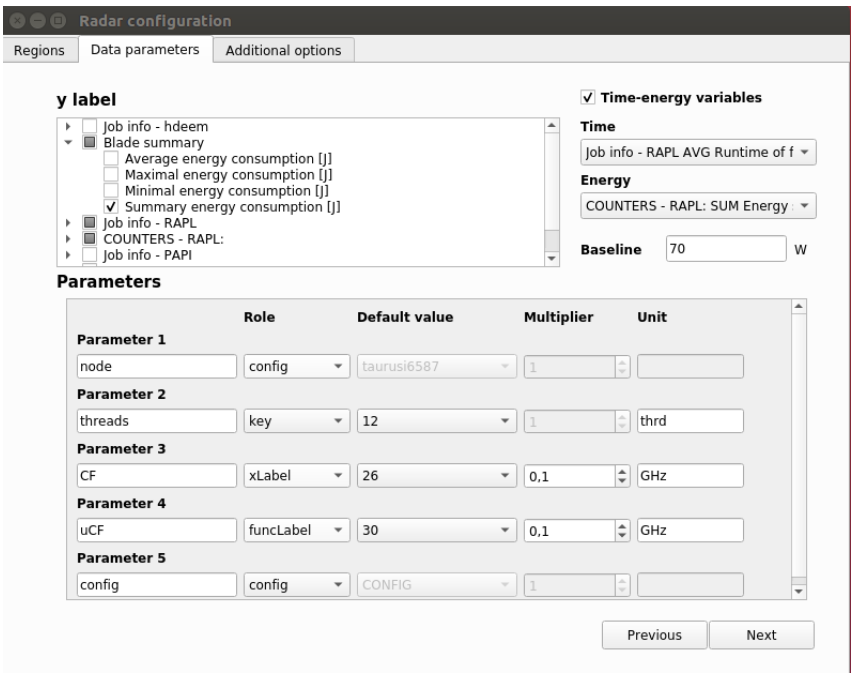


Figure 4: *RADAR configurator* tab – Data parameters

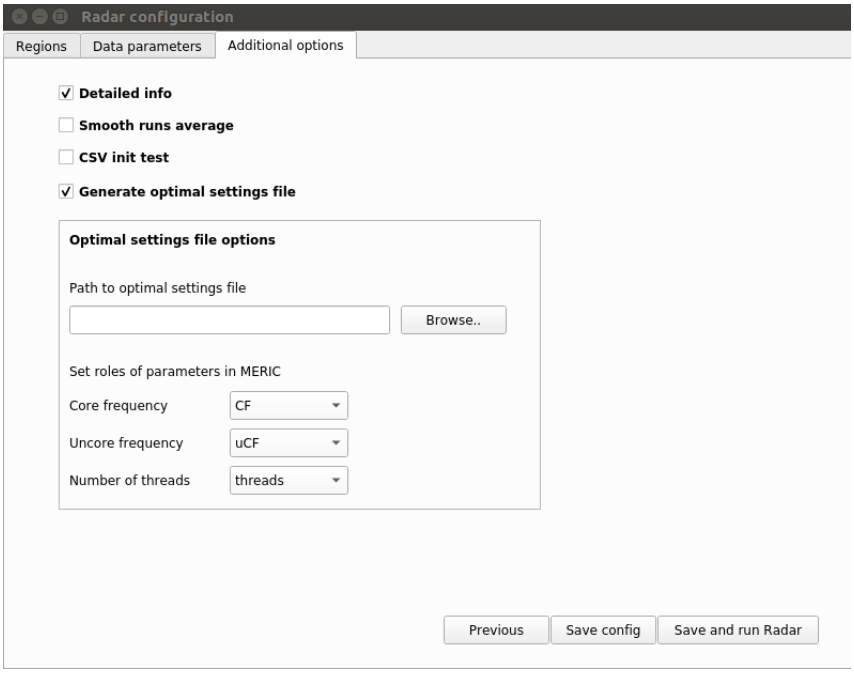


Figure 5: *RADAR configurator* tab – Additional options

The last tab of the application, shown in the Figure 5, has several check-boxes:

- **Detailed Info**: generate LaTeXreport with detailed information about each reagion of the analyzed application.

- **Smooth runs average**: omit outliers when computing mean values.
- **CSV init test**: turn on or off search in the analyzed data for potential corruption [2].
- **Generate optimal settings file**: generate MERIC configuration file with the optimal configuration for each analyzed region.

For the MERIC configuration file we must specify, which parameters represents the parameters, that MERIC is able to change during the application runtime. If any of the parameters had not been used, the corresponding item should remain unspecified.

When all the required configurations are specified, `Save and run RADAR` button will start RADAR analysis (only if *pathToRadar.json* is correctly specified). The process may take a moment, depends on the size and complexity of the analyzed data. Progress of the RADAR computations will be printed into the terminal.

[2] CSV init test may significantly slow down analysis process.

3 *RADAR visualizer*

RADAR visualizer is another graphical application from the RADAR toolbox. It provides detailed insight into the application behavior as well as the RADAR report, however it also brings possibility to select only the information we are interested in, and moreover interactive graphic elements that provides better insight into the behavior of evaluated application's regions.

Previous Section 2 describes the RADAR configuration, which is mostly shared for both *RADAR configurator* and *RADAR visualizer*. Also in case of the *RADAR visualizer* we have to specify list of the regions to include to the analysis, select sources of energy, time and performance counters, as well as names and roles of all the tuned parameters. Configuration files made by *RADAR configurator* can be used in *RADAR visualizer* as well as the ones from *RADAR visualizer* in *RADAR configurator*. To understand the correct application settings, please, read the Section 2 first.

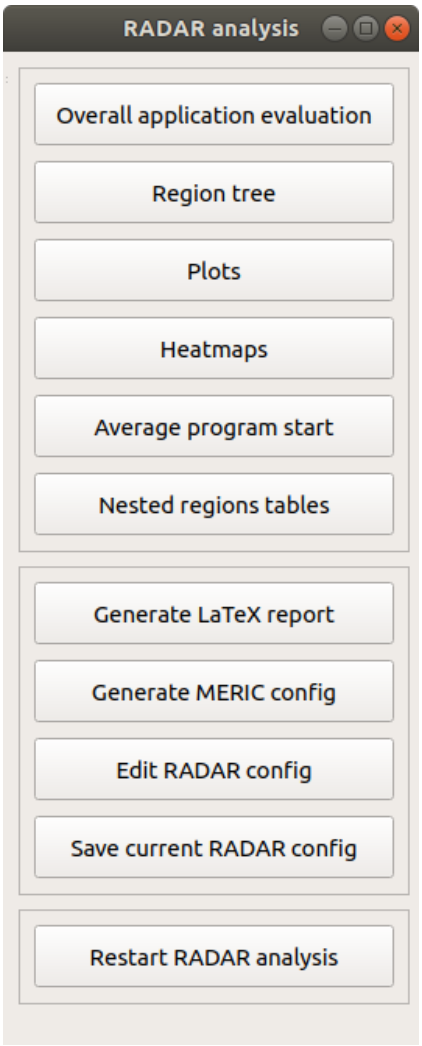


Figure 6: *RADAR visualizer* main menu

Use "`runRadarGUI_analyze.py`" script to run *RADAR visualizer*, that will start the application with RADAR configuration specification or loading and after successful RADAR analysis (the analysis progress is also printed into the terminal), that precedes the visualization of the application behavior in graphical representation, main menu is displayed, that provides list of options as shown in Figure 6.

The buttons of the menu are divided into two main sections. The first section provides graphic elements that represents the application behavior, the other one provides MERIC's and RADAR's configuration options and extended LaTeX report generation interface.

All the elements describing the application behavior are presented in the Section 4, however among all of these are two common features, that we will describe now. In the the window of the elements you can find following options:

- `Generate LaTeX code` – generates LaTeX code of the current graphic content in the users' defined configuration, and saves it into a file.
- `Add to LaTeX report` – when generating a LaTeX report of the whole application, this graphic content in the current configuration will be part of the report.
- `Save` – Graph and heatmap also provide option to save the element as image.

The `Generate LaTeX report` window, shows all the available elements in the corresponding report hierarchy, as shown in the Figure 7. In case that the `Add to LaTeX report` option was used, the hierarchy tree will also contain the extra elements that were added. In default only these extra elements are selected to be part of the final application report.

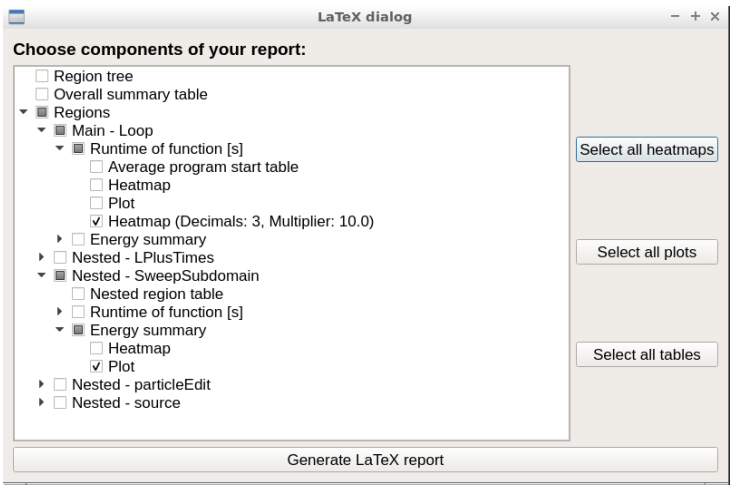


Figure 7: *RADAR visualizer* structure of the report dialog

In the Figure 7 is a user-defined heatmap for the main region named *Loop* with specified number of decimals and applied multiplier.

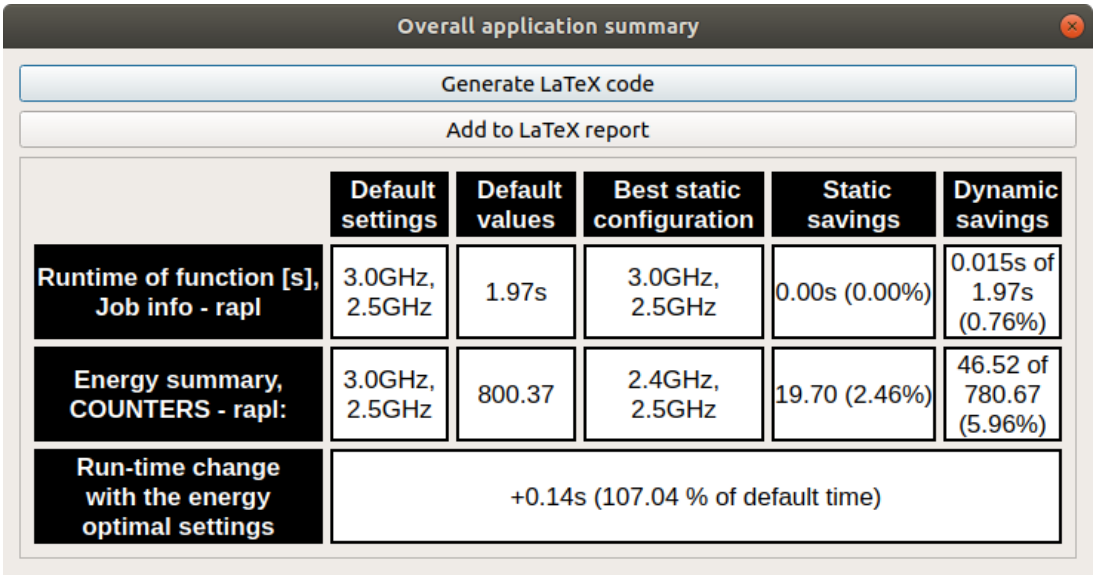
Stand alone LaTeX code of a one single element as well as the whole report, always include *"readex_header.tex"* file, that specifies all the necessary settings and LaTeX packages used in the report.

4 Understanding RADAR report

This section presents elements that are part of the RADAR report and also can be visualized from the *RADAR visualizer*, namely graphs, heatmap and tables describing behavior of the selected regions in different configurations. The elements are present in the order that we can find them in the RADAR report.

4.1 Overall application summary

Example of the Overall application summary table from the *RADAR visualizer* is shown in the Figure 8.



	Default settings	Default values	Best static configuration	Static savings	Dynamic savings
Runtime of function [s], Job info - rapl	3.0GHz, 2.5GHz	1.97s	3.0GHz, 2.5GHz	0.00s (0.00%)	0.015s of 1.97s (0.76%)
Energy summary, COUNTERS - rapl:	3.0GHz, 2.5GHz	800.37	2.4GHz, 2.5GHz	19.70 (2.46%)	46.52 of 780.67 (5.96%)
Run-time change with the energy optimal settings	+0.14s (107.04 % of default time)				

Figure 8: Overall application summary

This table provides summary information about the application in its default settings, the best static configuration and when the dynamic configuration switching will be applied. The dynamic savings are evaluated in compare to the best static configuration, overall savings of the dynamic savings are sum of the static savings and dynamic savings.

Run-time change information tells us how the runtime of the whole application will change if we apply dynamic switching to the optimal energy-saving configuration in compare to the application runtime in the default configuration. This information is provided only if the RADAR configuration had the information about `Time-energy variables`.

4.2 Plot and heatmap

Graph of the values measured in the different configurations that were applied during the measurement phase give us the most synoptic description of the region behavior. Below the graph is also information what is the optimal configuration for the region, not only for the `xLabel` and `funcLabel` but values of the `key` parameters too.

When displaying the graph in the *RADAR visualizer* it is possible to zoom to specific area of the graph or select from line to scatter plot type.

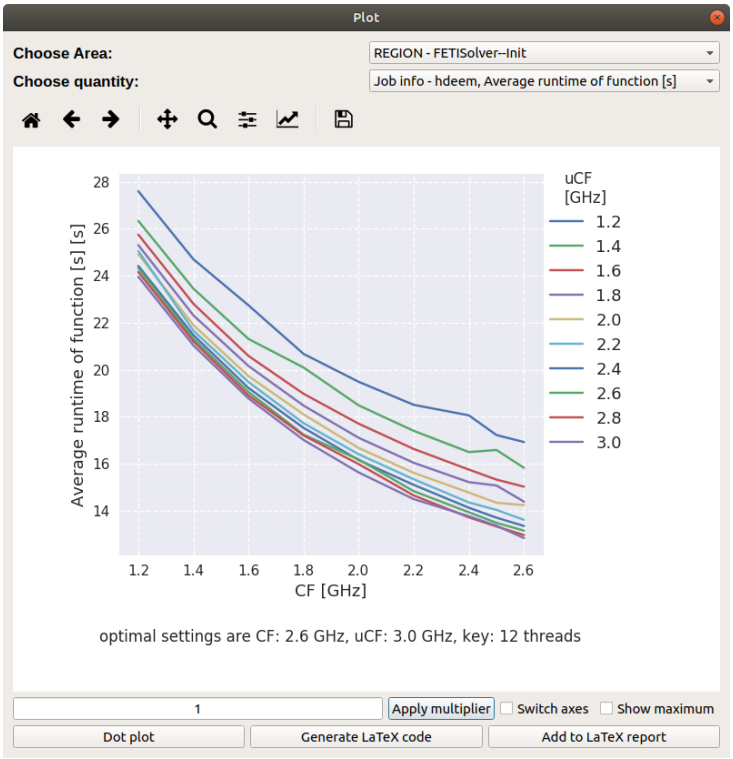


Figure 9: A region behavior when different CPU core and uncore frequencies had been applied

In compare to the plot, heatmap provides exactly the same information, however exact measured values can be read from it. Furthermore, by clicking on a specific cell of a heatmap, user can get information about the values, from which is the value in the cell computed from. This might be handy in case of outlying measurements to identify them.

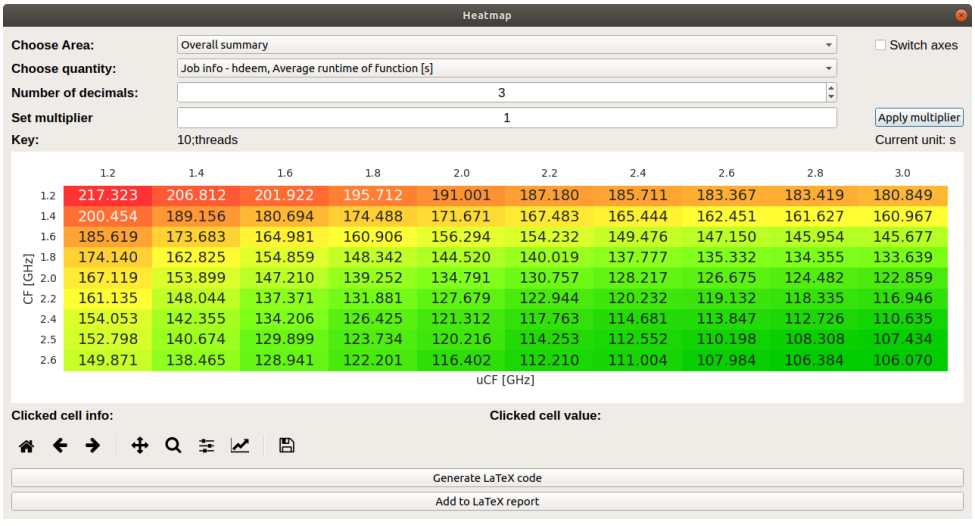
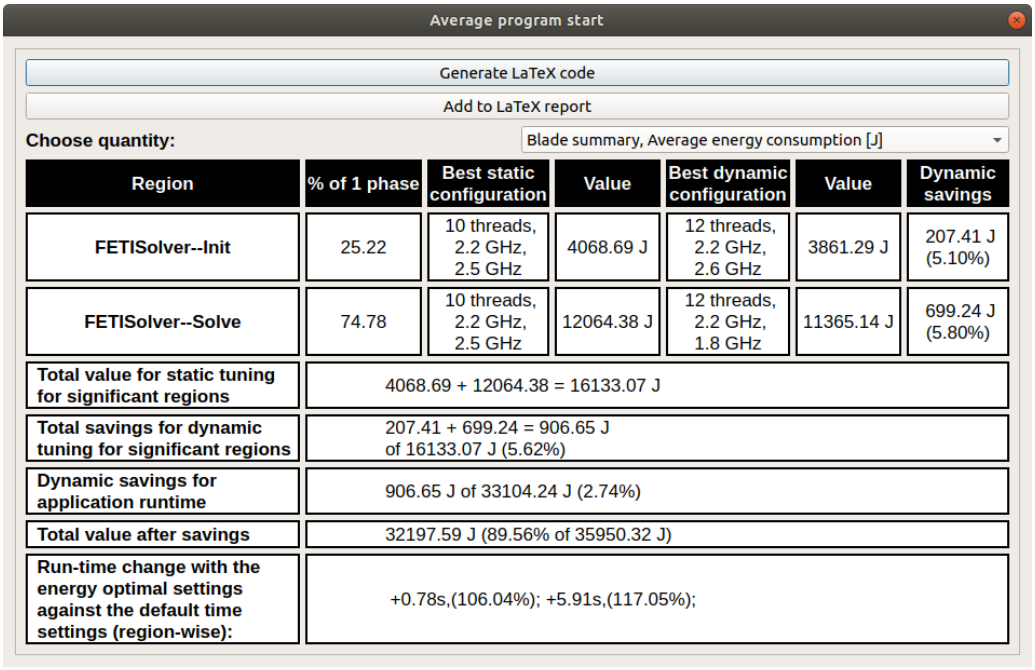


Figure 10: Heatmap representaion of a region behavior

4.3 Average program start

Average program start table shows all the nested regions included into the analysis. For each region we have information about the region size in compare to the main region according the time, energy or any other selected values.



Region	% of 1 phase	Best static configuration	Value	Best dynamic configuration	Value	Dynamic savings
FETISolver--Init	25.22	10 threads, 2.2 GHz, 2.5 GHz	4068.69 J	12 threads, 2.2 GHz, 2.6 GHz	3861.29 J	207.41 J (5.10%)
FETISolver--Solve	74.78	10 threads, 2.2 GHz, 2.5 GHz	12064.38 J	12 threads, 2.2 GHz, 1.8 GHz	11365.14 J	699.24 J (5.80%)
Total value for static tuning for significant regions	4068.69 + 12064.38 = 16133.07 J					
Total savings for dynamic tuning for significant regions	207.41 + 699.24 = 906.65 J of 16133.07 J (5.62%)					
Dynamic savings for application runtime	906.65 J of 33104.24 J (2.74%)					
Total value after savings	32197.59 J (89.56% of 35950.32 J)					
Run-time change with the energy optimal settings against the default time settings (region-wise):	+0.78s,(106.04%); +5.91s,(117.05%);					

Figure 11: Average program start table

The best static configuration is the same for all the nested regions (optimal configuration of the main region) and following column shows region consumption in this configuration. The best dynamic configuration is obviously individual for reach region. Also following column named *Value* presents region's consumption in the best dynamic configuration. Following dynamic savings give us information how much we save in compare to the best static configuration of the application. Summary for all the regions follows at the end of the table.

4.4 Cluster analysis

Some energy measurement systems store power samples and MERIC may store them for each region call. This samples carry important information, because it may show that the region should be split into two or more smaller regions, if there are continuous clusters of power samples with the approximately same value. If such region is not covered with one or more nested regions, we loose opportunity to exploit the dynamism.

Another kind of dynamism can be detected when the cluster analysis does not compare power samples within a region call, but overall energy consumption of all the region's calls. If the consumption is not stable, it may vary together with a different region callpath or input. RADAR does not have any information about region input, however it will report region callpath and call id, to provide the information how the region should be split not to loose dynamism to exploit.

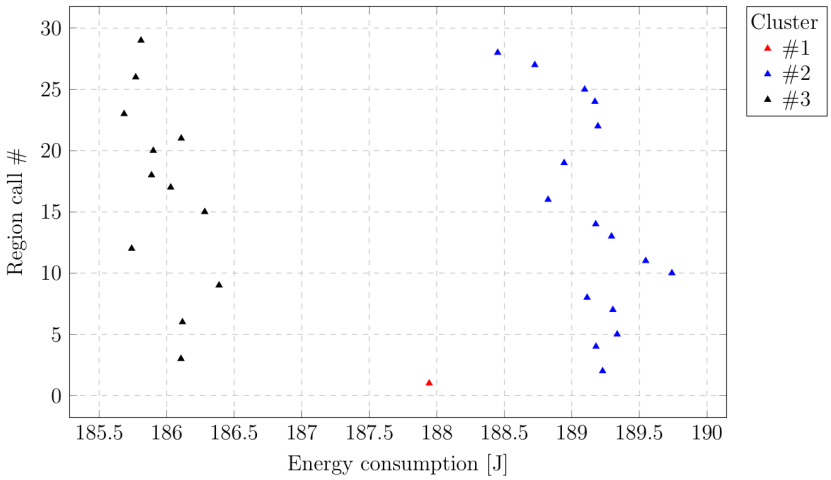


Figure 12: Cluster analysis

Cluster analysis is currently **not available** from the *RADAR visualizer*.

4.5 Nested region table

The last type of available table shows selected region behavior for all the measured values. Contribution of this table is in comparison of the region values and dynamicity for each region call (Phase ID).

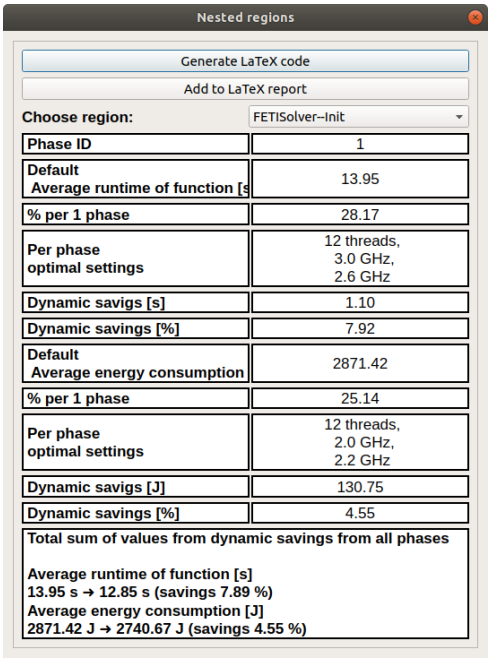


Figure 13: Nested region table