Horizon 2020
European Union funding
for Research & Innovation

GA no. 671657

READEX
Runtime Exploitation of Application Dynamism
for Energy-efficient eXascale computing

D2.2
Visualization of Dynamism

Document type:	Report
Dissemination level:	Public
Work package:	WP2
Editors:	Umbreen Sabir Mian (TUD)
	Anamika Chowdhury (TUM)
Contributing partners:	TUD, TUM
Reviewers:	Lubomir Riha, Jan Zapletal, Martin Beseda (IT4I)
	Zakaria Bendifallah, Othman Bouizi (Intel)
Version:	3.0

Document history

Version	Date	Author/Editor	Description
0.1	13/06/17	Anamika Chowdhury (TUM)	1^{st} TOC draft
0.2	01/07/17	Umbreen Sabir Mian (TUD)	Added reviewers and writing responsibilities
0.3	04/07/17	Umbreen Sabir Mian (TUD)	Initial version of switching visualization
0.4	04/08/17	Umbreen Sabir Mian (TUD)	Initial version of introduction, dynamism visualization and summary
0.5	05/08/17	Anamika Chowdhury (TUM)	Initial version of Section 2
1.0	07/08/17	Umbreen Sabir Mian (TUD)	First version ready for review
1.1	17/08/17	Umbreen Sabir Mian (TUD)	Section Dynamism Visualization revised Comments from first review addressed
1.2	18/08/17	Anamika Chowdhury (TUM)	1st review comments on Section 3 addressed
1.3	19/08/17	Umbreen Sabir Mian (TUD)	Vampir trace figure added in Section 2
2.0	20/08/17	Umbreen Sabir Mian (TUD)	Draft ready for Second review
2.1	28/08/17	Umbreen Sabir Mian (TUD)	Zapletal's comments from second review addressed
2.2	28/08/17	Anamika Chowdhury (TUM)	Second review comments on Section 3 addressed
2.3	29/08/17	Umbreen Sabir Mian (TUD)	Bendifallah's comments from second review addressed
3.0	29/08/17	Umbreen Sabir Mian (TUD)	Final Version ready for submission

Contents

1 Introduction

In READEX, we are focused on exploiting dynamic behaviour within the application for energy efficiency tuning. One of the tasks in WP2 is to implement a way for visualization of application dynamism.

Prior to applying the READEX tuning methodology on a given application, the tuning potential i.e., the exploitable dynamism in this application is checked as a preliminary step. The Task 2.4 specifically focuses on visualization of the dynamism information which is found in the application before applying the READEX tuning.

To estimate the tuning potential of an application, a tool called `readex-dyn-detect` has been implemented. The `readex-dyn-detect` tool presents the user with the application dynamism information in a text format. This dynamism information is also presented to the user in Vampir Trace visualization tool.

Figure 1 shows all components of the READEX architecture. In this deliverable, we will focus mainly on:

- The working of `readex-dyn-detect` and its output visualization in Vampir
- The visualization of configuration switching using visualization metric plugin from READEX Runtime Library (RRL)
- The visualization of the scenarios identified by Periscope Tuning Framework (PTF) and the tuning model generated by PTF

This deliverable outlines all the features which are designed in the READEX tool suite for visualizing the results of the READEX methodology. These include visualization of application dynamism, visualization of configuration switching and visualization of scenarios and the tuning model.

The structure of the deliverable is as follows: Section 2 will present the approach and output of the `readex-dyn-detect` tool. It will also show how the dynamism information is visualized in Vampir. Section 3 will outline the visualization of scenarios and the tuning model. The visualization of configuration switching will be depicted in Section 4.

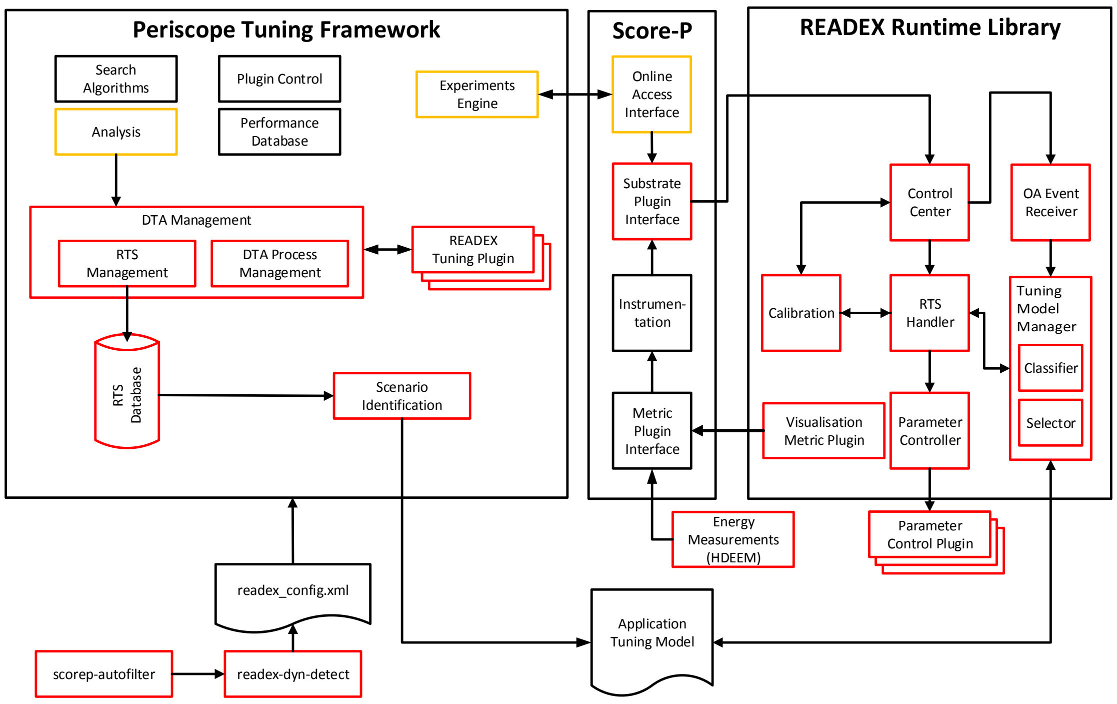


Figure 1: Overview about all components of the READEX tool suite

2 Dynamism Visualization

The idea behind READEX is to exploit the dynamism available inside the application and use it to reduce its energy consumption. Before applying READEX methodology, application is thus analyzed for its dynamic behaviour. If the application shows some dynamism in its behaviour, only then the READEX tuning is applied.

In this section, we start with explaining what is considered dynamism and how this dynamism is estimated for a given application using the `readex-dyn-detect` tool. Next, we present how the dynamism detected in the appliaction is visualized in Vampir.

2.1 Application Dynamism

The READEX tool suite tunes hardware, system software and application tuning parameters as described in D1.1 and D1.2 [3, 5]. In order to apply the best configurations for the tuning parameters during runtime application tuning (RAT) that are computed during design time analysis (DTA), the dynamism present in an application has to be first analyzed and quantified using dynamism metrics during DTA.

The dynamism metrics that are measured and used in the READEX methodology are:

1. Execution time.
2. Energy consumed.
3. Computational intensity.

Among these three metrics, the semantics of execution time and energy consumed are straightforward. Variation in the execution time and energy consumed by a region in an application during its execution is an indication of different resource requirements. The computational intensity is a metric that is used to model the behaviour of an application based on the workload imposed by it on the CPU and the memory. Presently, computational intensity is calculated using the following formula:

$$ComputationalIntensity = \frac{Total\ number\ of\ instructions\ executed}{Total\ number\ of\ L3\ cache\ misses} \tag{1}$$

Computational intensity can directly dictate the tuning of two hardware parameters: CPU core frequency and CPU uncore frequency. A low computational intensity indicates an application that is more memory intensive, which is the result of increased L3 cache misses. Since this would cause increased traffic between the L3 cache and the main memory, it will be desirable to increase the uncore frequency. On the other hand, a high computational intensity indicates an application that is more computation intensive. In this case, it will be desirable to increase the frequency of the CPU cores.

In the context of the READEX project we distinguish between two types of dynamism:

- Inter-phase dynamism: Each phase of a phase region in the application exhibits different characteristics. This results in different values for the measured objective values and thus may require different configurations to be applied for the tuning parameters.
- Intra-phase dynamism: Each runtime situation (rts) of the significant regions in a phase region exhibits different characteristics and thus may need different configurations to be applied for the tuning parameters.

Due to the different localities of dynamism in an application, the dynamism metrics are measured and analysed from the following perspectives:

- For all phases of the phase region in the application – this allows analysis of inter-phase dynamism that may be present in the application.
- For all runtime situtations of the significant regions in the application – this allows analysis of intra-phase dynamism that may be present in the application.

A detailed report on application dynamism has been presented in deliverable D5.1 [6].

2.2 Dynamism Analysis

Detecting the dynamism of an application is the initial step of the READEX approach. The tuning potential of an application is determined by measuring its intra-phase and inter-phase dynamism by `readex-dyn-detect`. The tool currently focuses on the execution time and compute intensity as the main characteristics. Variation in the execution time of significant regions across *rts's* indicates intra-phase dynamism. Variation in the execution time across *rts's* of the phase region indicates inter-phase dynamism. Furthermore, different compute intensity, such as compute vs. memory bound, of different significant regions also indicates intra-phase dynamism. An elaborate description of the architecture and working of the `readex-dyn-detect` tool can be found in deliverable D2.1 [4].

The tool analyzes for each significant region the variation in the time characteristics. It computes the standard deviation relative to the mean execution time of the region in percent ($deviation_r$) and relative to the mean execution time of the phase ($deviation_p$) as described in Equations (2) and (3) below, respectively. The values characterize how significant the variation in the execution time is for the region and phase execution respectively. All the information required to calculate these dynamism metrics is obtained from the "profile.cubex" file generated by Score-P after running the application with Score-P.

$$deviation_r^{reg} = \frac{dev\_t_{incl}^{reg}}{mean\_t_{incl}^{reg}} * 100 \tag{2}$$

$$deviation_p^{reg} = \frac{dev\_t_{incl}^{reg}}{mean\_t_{incl}^{phase}} * 100 \tag{3}$$

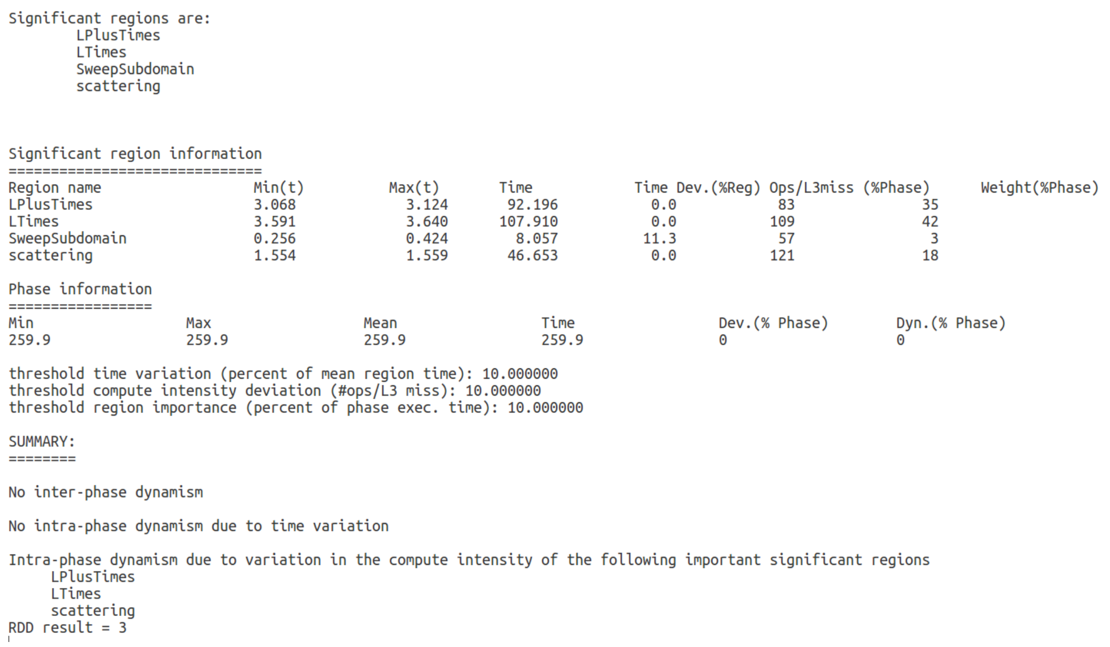
The variation is considered significant if it is beyond a threshold v_t. To decide whether this leads to significant dynamism, the tool computes the computational weight of the region, i.e., its percentage on the phase execution time, according to Equation (4)

$$weight = \frac{t_{incl}^{reg}}{t_{incl}^{phase}} * 100. \tag{4}$$

If the region's time variation is significant and its weight is larger than a threshold v_w then the tool will report intra-phase dynamism due to that significant region.

Another source of intra-phase dynamism is the variation based on the *compute intensity* of different significant regions. It is calculated by using Equation (1) presented above. The total number of instructions and the number of L3 cache misses are obtained using the PAPI counters "PAPI_TOT_INS" and "PAPI_L3_TCM".

Figure 2 shows the dynamism output produced by the `readex-dyn-detect` tool when applied to the Kripke application. It is only the partial output produced by the tool as the tool also produces output regarding the granularity of different regions in the application and phase information. As can be seen in Figure 2, there is no inter-phase dynamism reported. Intra-phase dynamism is reported due to the variation in compute intensity of three listed regions.



```
Significant regions are:
        LPlusTimes
        LTimes
        SweepSubdomain
        scattering


Significant region information
==============================
Region name                  Min(t)          Max(t)       Time            Time Dev.(%Reg) Ops/L3miss (%Phase)      Weight(%Phase)
LPlusTimes                   3.068             3.124       92.196           0.0            83               35
LTimes                       3.591             3.640      107.910           0.0           109               42
SweepSubdomain               0.256             0.424        8.057          11.3            57                3
scattering                   1.554             1.559       46.653           0.0           121               18

Phase information
=================
Min                  Max                  Mean                 Time                 Dev.(% Phase)        Dyn.(% Phase)
259.9                259.9                259.9                259.9                0                    0

threshold time variation (percent of mean region time): 10.000000
threshold compute intensity deviation (#ops/L3 miss): 10.000000
threshold region importance (percent of phase exec. time): 10.000000

SUMMARY:
========

No inter-phase dynamism

No intra-phase dynamism due to time variation

Intra-phase dynamism due to variation in the compute intensity of the following important significant regions
     LPlusTimes
     LTimes
     scattering
RDD result = 3
```

Figure 2: Dynamism information produced by `readex-dyn-detect` after Kripke tuning potential analysis

2.3 Dynamism Visualization in Vampir

Score-P generates Open Trace Format 2 (OTF2) [2] traces which can be visualized in Vampir. Similar to the `readex-dyn-detect` tool, it is required that the trace shall contain both the PAPI metrics “PAPI_TOT_INS” and “PAPI_L3_TCM” to calculate the compute intensity metric. Both of the PAPI metrics can be added to the trace by specifying through the environment variables “SCOREP_METRIC_PAPI” provided in Score-P. In order to visualize the dynamism information in Vampir, the trace generated after the application run, has to be read and rewritten with the dynamism metrics calculated.

A Python module has been implemented, which takes as input the existing OTF2 trace file and produces a new OTF2 trace file containing the dynamism metrics. The OTF2 library provides a Python interface to read and write the OTF2 files which has been employed for this task.

The Python module takes following as input:

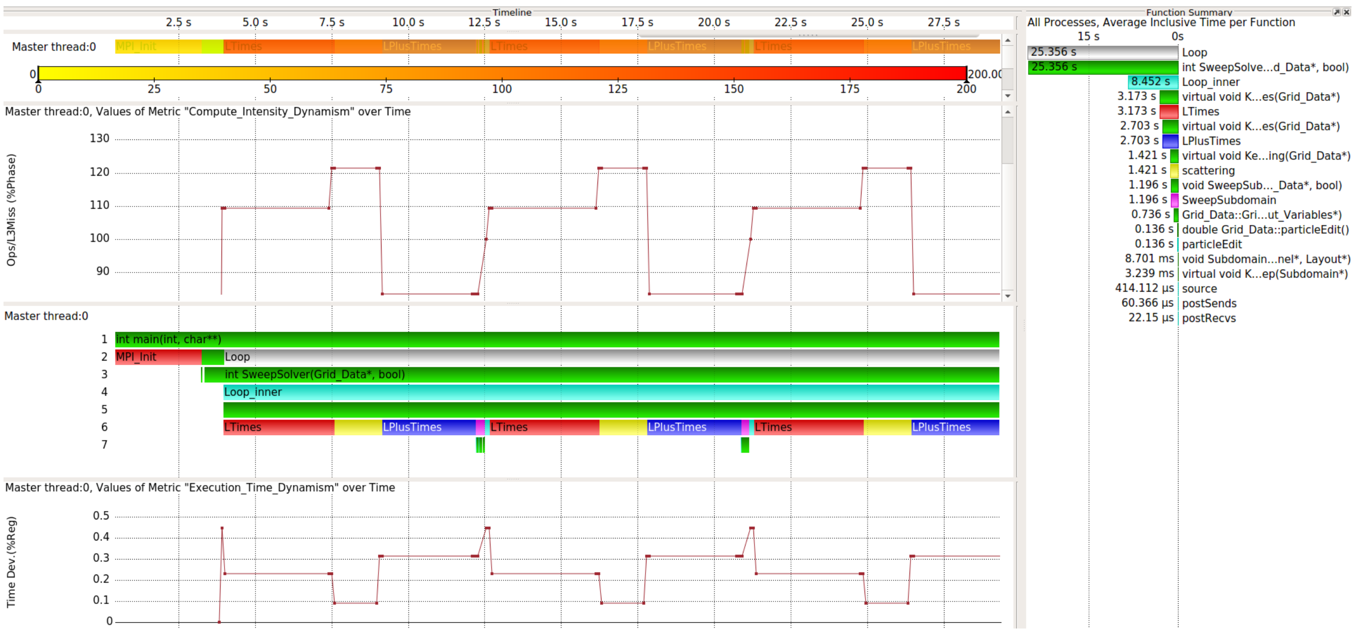
- Path for the trace file to read.
- Path to save the new trace file generated by the python module.
- Name of the phase region.
- Region execution time granularity threshold.

Two dynamism metrics, “Execution_Time_Dynamism” and “Compute_Intensity_Dynamism”, are added by the Python module to the trace file. The dynamism metrics are reported for each thread/process. It is important to mention here that each thread/process should contain only one invocation of the phase region, as the “Compute_Intensity_Dynamism” metric is calculated with respect to the phase region. Both the dynamism metrics are reported only for the regions which are inside the phase region and have their mean execution time greater than the granularity threshold.

Figure 3 shows the Vampir view of the dynamism metrics added to the trace for graphical dynamism visualization.

In Figure 3, the “Compute_Intensity_Dynamism” is shown as an overlay over the process timeline in top window. In this way, the user can visualize the presence of dynamism in the application in the form of a heat map. “Execution_Time_Dynamism” can be visualized in the same manner.

Furthermore, both the dynamism metrics are customizable as the other metrics using different Vampir features.



25.356 s	Loop
25.356 s	int SweepSolve...d_Data*, bool)
8.452 s	Loop_inner
3.173 s	virtual void K...es(Grid_Data*)
3.173 s	LTimes
2.703 s	virtual void K...es(Grid_Data*)
2.703 s	LPlusTimes
1.421 s	virtual void Ke...ing(Grid_Data*)
1.421 s	scattering
1.196 s	void SweepSub..._Data*, bool)
1.196 s	SweepSubdomain
0.736 s	Grid_Data::Gri...ut_Variables*)
0.136 s	double Grid_Data::particleEdit()
0.136 s	particleEdit
8.701 ms	void Subdomain...nel*, Layout*)
3.239 ms	virtual void K...ep(Subdomain*)
414.112 µs	source
60.366 µs	postSends
22.15 µs	postRecvs

Figure 3: Vampir trace showing “Compute_Intensity_Dynamism’ and “Execution_Time_Dynamism” metrics. The phase region is named “Loop” and granularity threshold is 500ms.

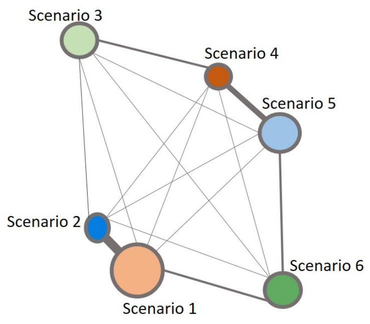


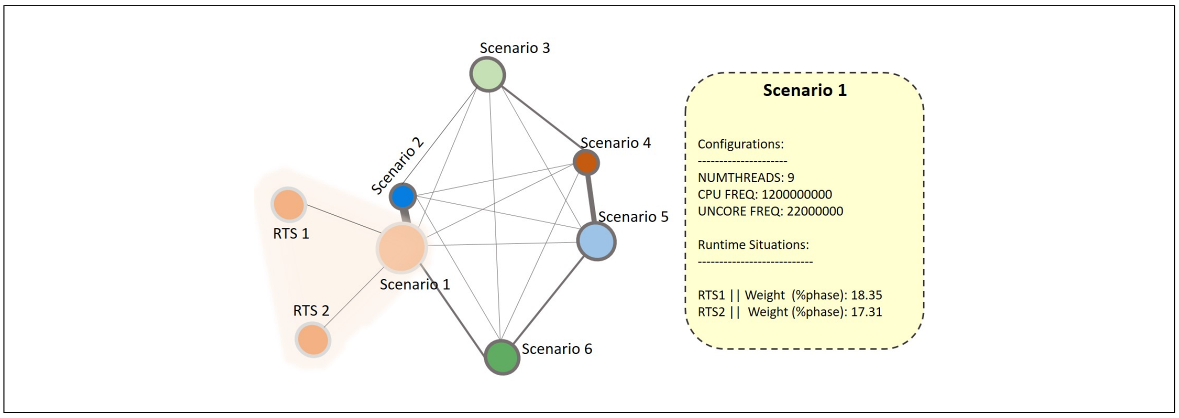
Figure 4: A forced layout graph to visualize the application tuning model of LULESH

3 Visualization of Application Tuning Model (ATM)

The READEX tool suite enables dynamic tuning through a two-staged methodology: DTA and RAT. During DTA a tuning model is precomputed with PTF that contains the best found configurations of tuning parameters for *rts's* of significant regions. The READEX methodology already supports two types of tuning parameters: hardware and system software parameters in the application tuning model. This tuning model can be improved by exploiting dynamism in application specific tuning parameters. To identify such parameters, the user should have in-depth knowledge about the applicaton structure and the influencing parameters inside of the application such as: the algorithm, data structure, blocking factor and so on. The effect of the best configuration of these tuning parameters can be inspected by comparing different scenarios. The comparison assists the user to insight on the relationship between scenarios in the tuning model and explains how similar scenarios appear in closer proximity, while dissimilar scenarios are apart.

To visualize the tuning model result of the READEX methodology, we used the *Forced layout graph*. The graph is constructed based on the JavaScript library D3.js [1]. It compares scenarios in the application tuning model with respect to their similarity and weight. In this context, similarity represents the distance of scenarios in a multi-dimensional tuning space, and weight is the aggregated execution time of *rts's* of a scenario relative to the phase execution time. While similarity is represented by the thickness of the edges between scenarios, the weight is visualized as the size of the circle representing a scenario. Eventually, the distance between scenarios is the result of all forces. The network adapts according to the forces dynamically.

Figure 4 shows the tuning model of the LULESH proxy application from the CORAL benchmark suite. The nodes in the figure represent scenarios found in tuning model. Each node is a cluster of *rts's* belonging to it. There are six scenarios in LULESH's tuning model where `Scenario 1` covers most of the execution time. On the other hand, `Scenario 2` and `Scenario 4` are the least significant nodes due to their lowest weights. As the figure shows,



```
Scenario 1

Configurations:
--------------------
NUMTHREADS: 9
CPU FREQ: 1200000000
UNCORE FREQ: 22000000

Runtime Situations:
--------------------------

RTS1 || Weight  (%phase): 18.35
RTS2 ||  Weight (%phase): 17.31
```

Figure 5: The expanded forced layout of tuning model upon clicking on a scenario node

`Scenario 1` and `Scenario 2` are the most similar scenarios and high thickness of the edge and lowest distance between them affirms that. On the opposite side, `Scenario 3` and `Scenario 6` are the most distant dissimilar scenarios.

To investigate each scenario, the user can click on a scenario node of the graph. Upon clicking on a node, the node expands with all the rts nodes of that scenario. Figure 5 shows the extended graph of the LULESH tuning model. A pop over box appears upon hovering on the node which shows the scenario information including rts's with their weight and configurations of the tuning parameters. In this figure, `Scenario 1` contains two rts's: `rts 1` and `rts 2` each representing 18.35% and 17.31% weight of the phase respectively. The application tuning parameter is yet to be implemented into ATM.

4 Configuration Switching Visualization

The configuration switching happens during both phases of the READEX methodology. During DTA, PTF runs experiments with different configurations to find the optimal configuration for each rts in the application, these are then stored in the tuning model. During RAT, for each rts, the optimal configuration from the tuning model is applied which requires configuration switching.

To enable the user to visualize the configuration switching for each region during DTA and during a production run, a visualization metric plugin has been implemented within RRL. The metric plugin adds each of the tuning parameters as a metric in the Vampir trace generated by Score-P. The user can select these metrics in Vampir and visualize the switching pattern for each metric.

Figure 6 shows the architecture of the READEX tool suite with the visualization metric plugin integrated. It uses the metric plugin interface provided in Score-P to add the tuning parameters as metrics in Vampir traces and to get the tuning parameter values from RRL.

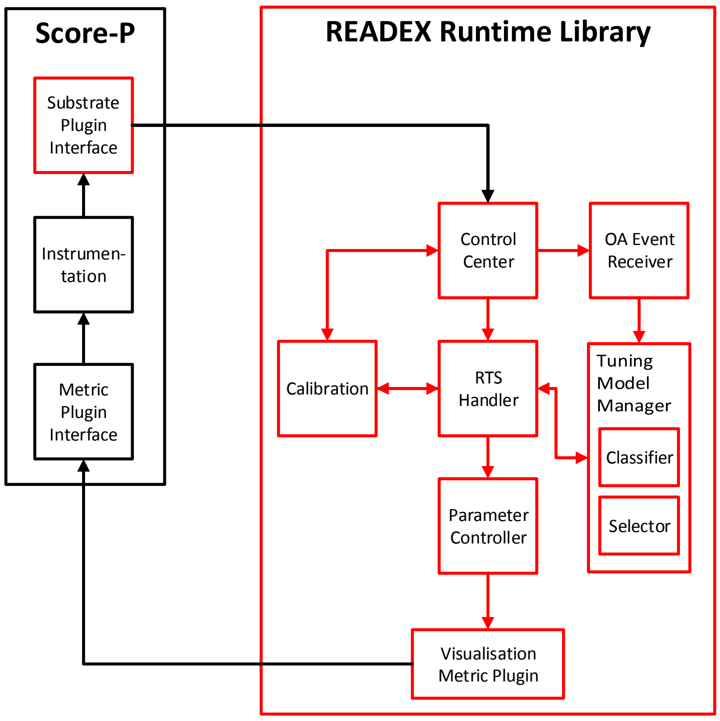


Figure 6: Overview of the READEX tool suite. The graph shows the information flow between the Score-P and the RRL. The "Visualization Metric Plugin" handles the visualization of configuration switching in Vampir traces.

The metric plugin has to be loaded using the Score-P environment variable SCOREP_METRIC_PLUGINS and the user can specify if all of the tuning parameters or only selected ones need to be added to the Vampir trace. The tuning parameters which the user wants to visualize can be specified using the Score-P environment variable "SCOREP_METRIC_SCOREP_SUBSTRATE_RRL". Any of the hardware, software and application tuning parameters can be chosen for visualization.

Figure 7 illustrates the switching of the CPU frequency and uncore frequency performed by RRL while tuning CPU frequency "CPU_FREQ" and uncore frequency "UNCORE_FREQ" for the Kripke benchmark. As can be seen in the Figure 7, the Vampir plots show the value of the tuning parameters for each region. The user can visulaize, for example, the configuration for the phase region by cliking on the phase region instances (found by the phase region name).

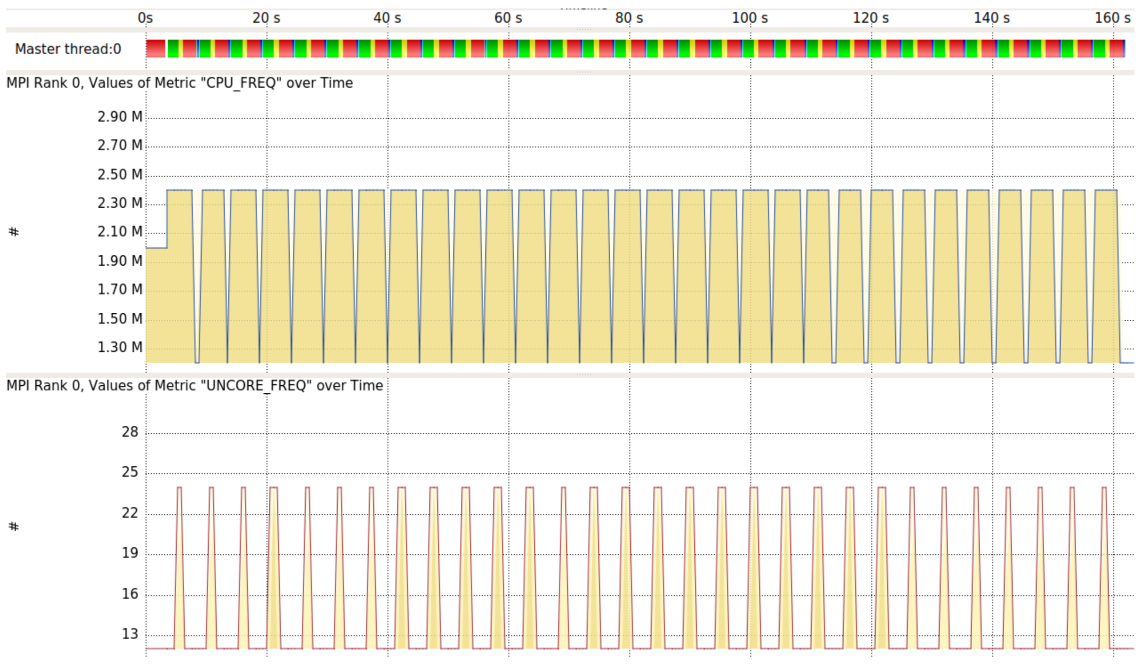


Figure 7: CPU_FREQ and UNCORE_FREQ switchings by RRL during Kripke run-time tuning

5 Summary

In this deliverable, different ways of visualizing the results produced during applying the READEX methodology have been presented. Every significant tool of the READEX tool suite has its own visualization feature. Before applying the READEX methodology, the application is analyzed for inter-phase and intra-phase dynamism using the `readex-dyn-detect` tool. If the tool reports the presence of inter- or intra-phase dynamism in the application, only then the READEX tuning methodology is applied.

A Python module has also been implemented, which provides the user a possibility to visualize the dynamism information detected by `readex-dyn-detect` in Vampir. The dynamism metrics *Execution_Time_Dynamism* and *Compute_Intensity_Dynamism* are added to the Vampir trace for visualization.

Visulaization of the results produced after DTA, the *tuning model* and the scenarios is made possible through the graphical tool based on the JavaScript library D3.js which generates a *Forced layout graph*. The graph shows the different scenarios and the *tuning model* as nodes. User can expand the nodes by clicking on them and can view the details of all scenarios and the generated *tuning model*.

Finally, a metric plugin has been implemented within RRL to allow the user to visualize the switching of tuning parameters configuration both during DTA and RAT.

References

[1] M. Bostock, V. Ogievetsky, and J. Heer. D3; data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec 2011.

[2] Dominic Eschweiler, Michael Wagner, Markus Geimer, Andreas Knpfer, Wolfgang E. Nagel, and Felix Gerd Eugen Wolf. Open Trace Format 2 - The Next Generation of Scalable Trace Formats and Support Libraries. In *Applications, Tools and Techniques on the Road to Exascale Computing : proceedings of the 14th biennial ParCo conference ; ParCo2011 ; held in Ghent, Belgium / Ed. by Koen De Bosschere ...*, volume 22 of *Advances in Parallel Computing*, pages 481–490, Amsterdam [u.a.], 2012. IOS Press. '... contains the proceedings of ParCo2011, the 14th biennial ParCo Conference, held from 31 August to 3 September 2011, in Ghent, Belgium'.

[3] Andreas Gocht, Zakaria Bendifallah, Umbreen Sabir Mian, and Othman Bouizi. D1.1: Hardware and system-software tuning plugins. Technical report, TUD, Intel, 2016.

[4] Per Gunnar Kjeldsberg, Michael Gerndt, Mohammed Sourouri, and Anamika Chowdhury. D2.1 analysis of tuning potential and scenario identification. Technical report, NTNU, TUM, 2016.

[5] Umbreen Sabir Mian and Zakaria Bendifalah. D1.2 final tuning plugins. Technical report, TUD, Intel, 2017.

[6] Lubomir Riha, Jan Zapletal, Martin Beseda, Ondřej Vysocký, and Vojtěch Nikl. D5.1 hardware and system-software tuning plugins. Technical report, IT4I-VSB, 2017.