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

EnA-HPC @ ISC'17

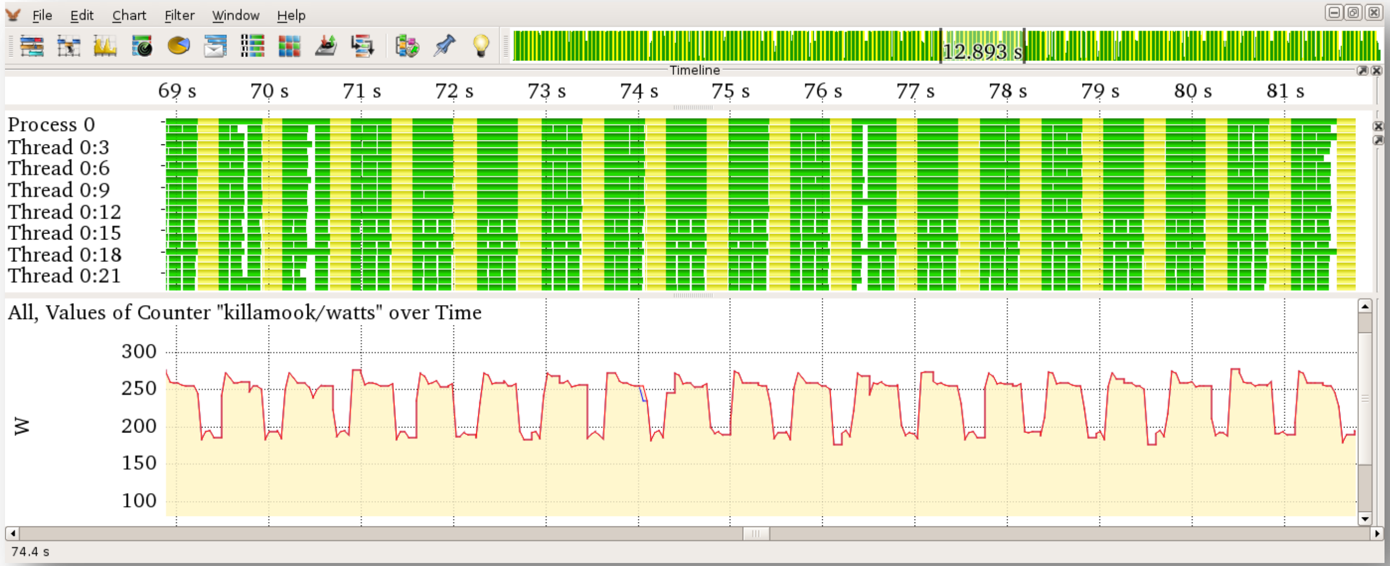
Robert Schöne – TUD

Horizon 2020
European Union funding
for Research & Innovation

Project Motivation

Applications exhibit dynamic behaviour

- Changing resource requirements
- Computational characteristics
- Changing load on processors over time



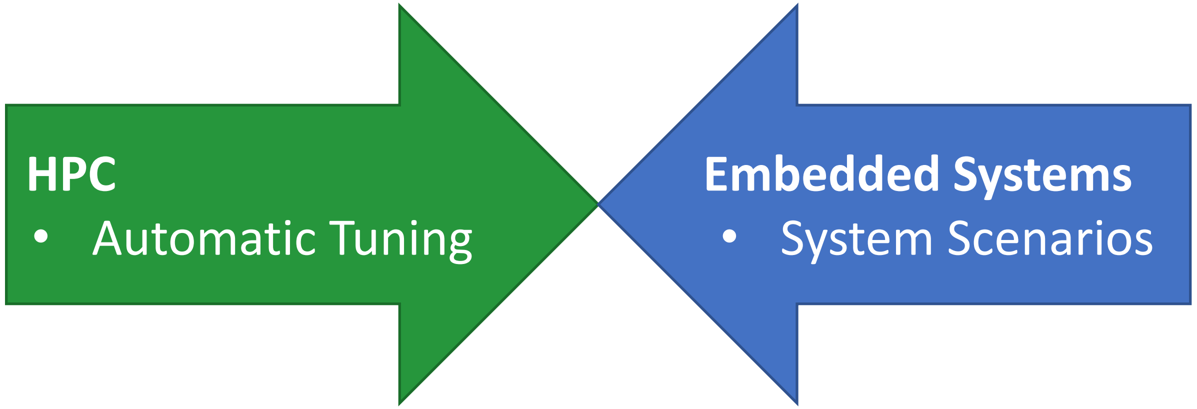
Overview

READEX creates a **tools-aided methodology for automatic tuning** of parallel applications

- Dynamically adjust system parameters to actual resource requirements

Join technologies from embedded systems and HPC

- HPC: PTF, Score-P, and HDEEM
- ES: System scenario methodology



Background and Project Partners

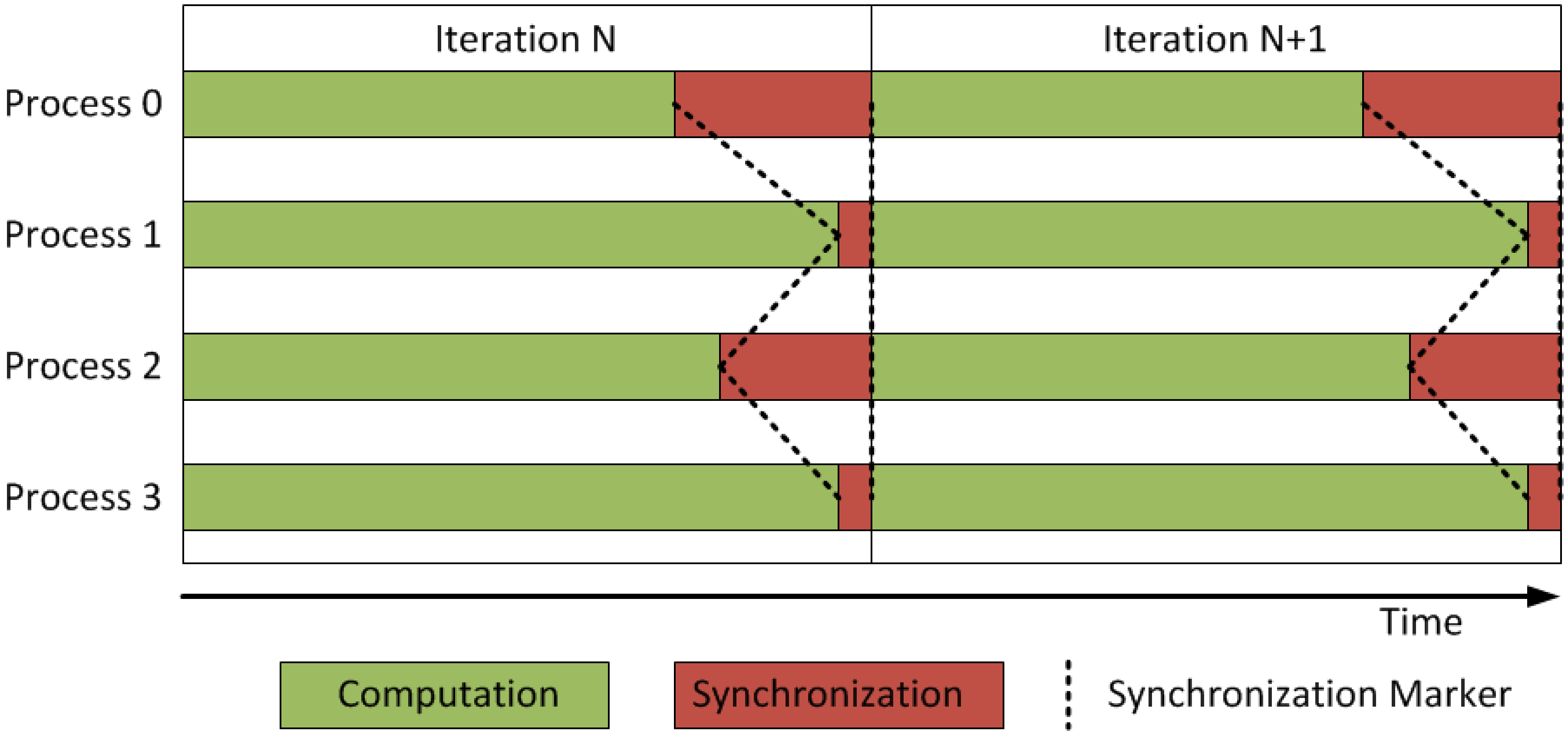
- Grant agreement No 671657
- Officially started September 1st, 2015

- Technische Universität Dresden/ZIH (Coordinator)
- Norwegian University of Science and Technology
- Technische Universität München
- IT4Innovations, VSB-Technical University of Ostrava
- NUI Galway, Irish Centre for High-End Computing
- Intel France
- Gesellschaft für numerische Simulation mbH

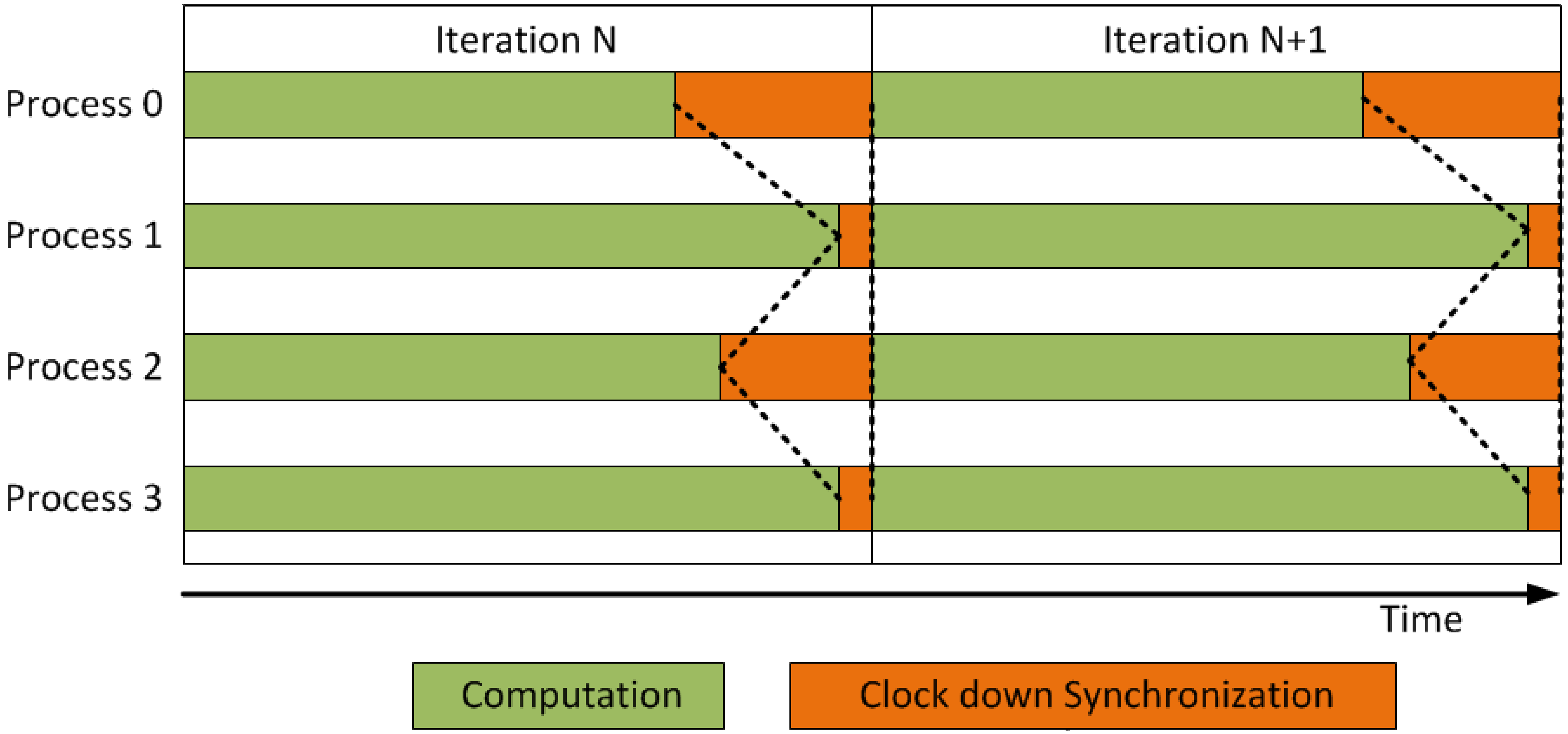
Energy Efficiency Tuning Types

1. Static or dynamic tuning?
   - Uniform or changing demand of programs
   - Dynamic: sampling or instrumentation
2. Reducing power or runtime?
   - Power:
     - Frequencies
     - C-States
     - Speculative execution (e.g., prefetchers)
   - Runtime:
     - Frequencies (Turbo, various resources share single power budget)
     - Select optimal code paths
     - Optimize code
3. Tackling regions or balancing?

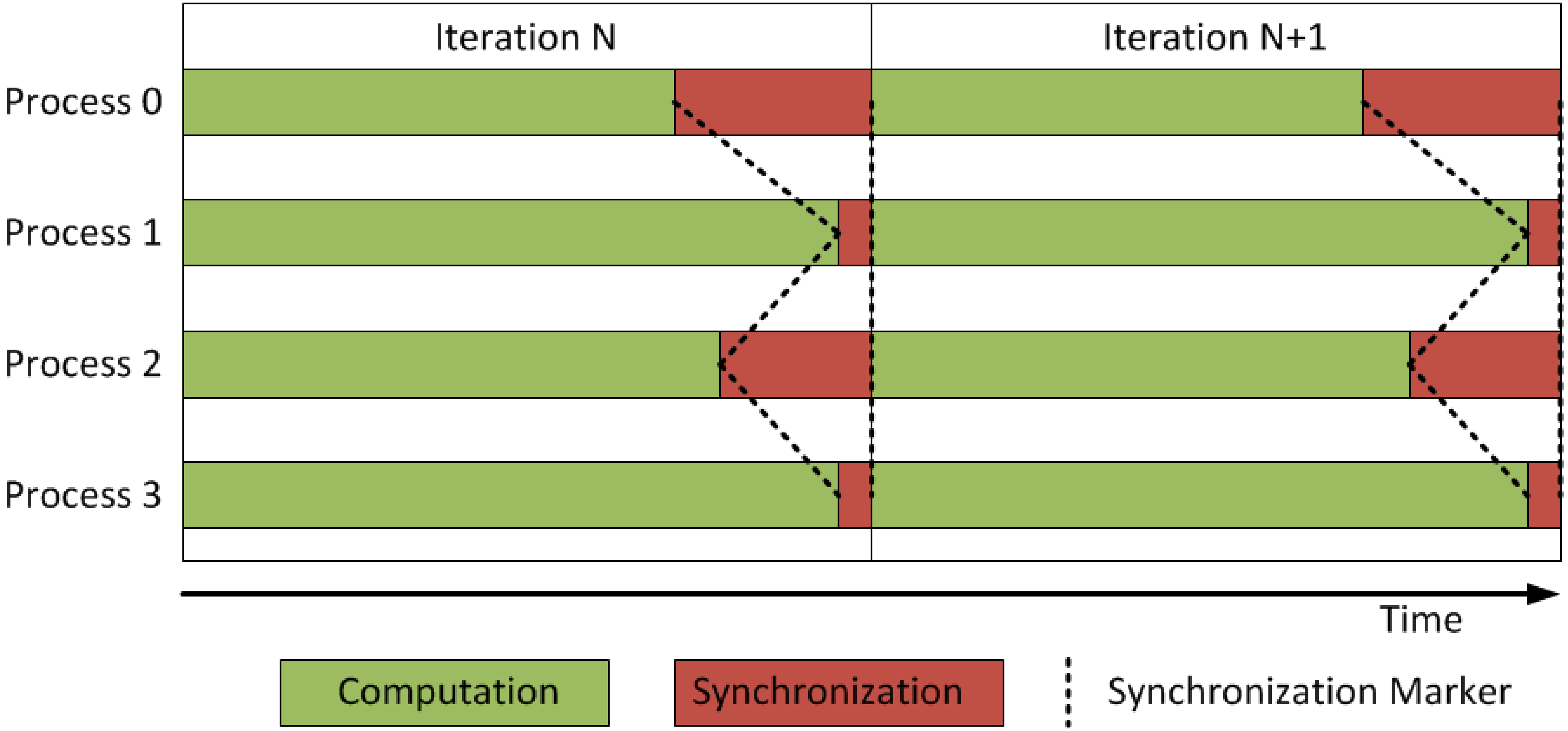
Energy Efficiency Tuning Types



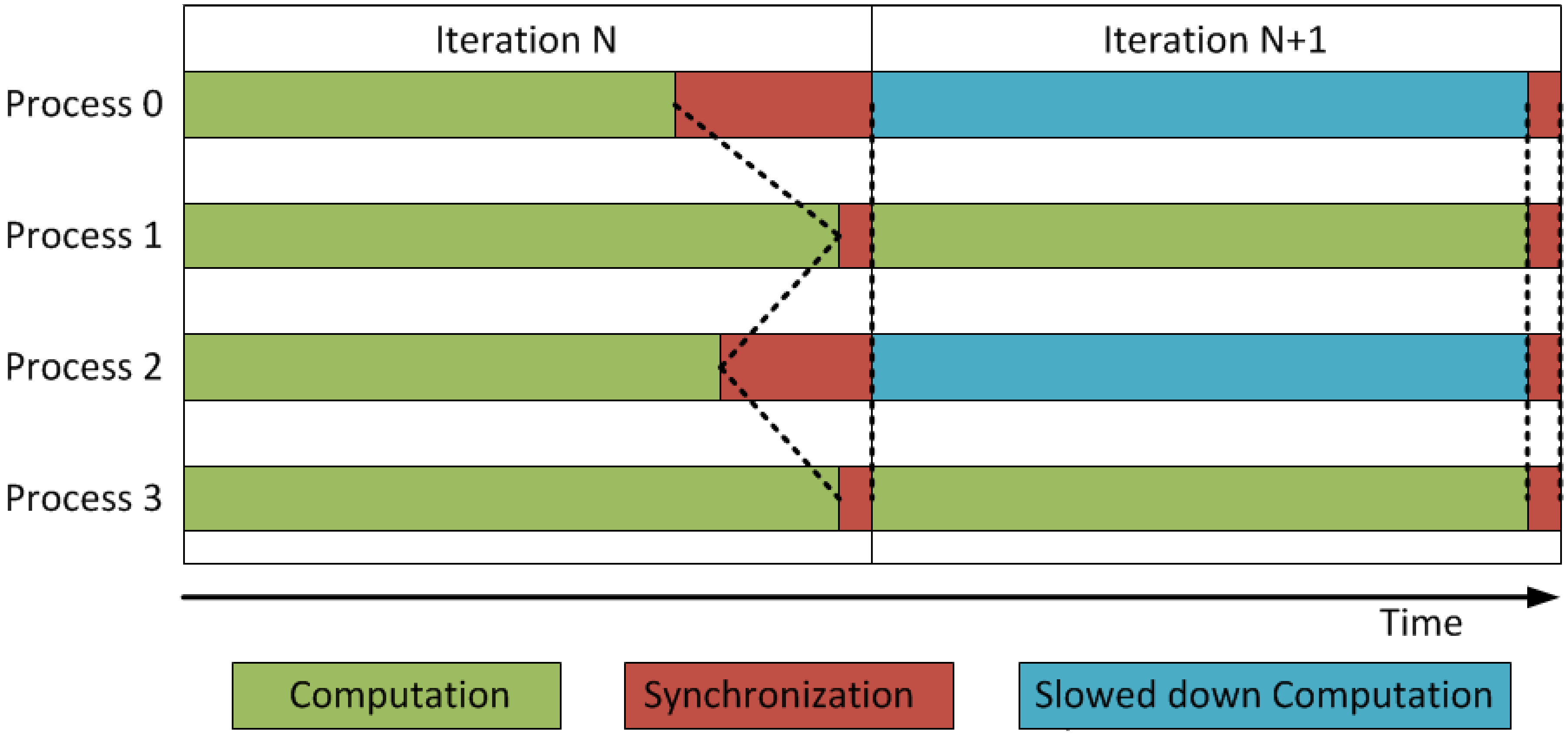
Energy Efficiency Tuning Types



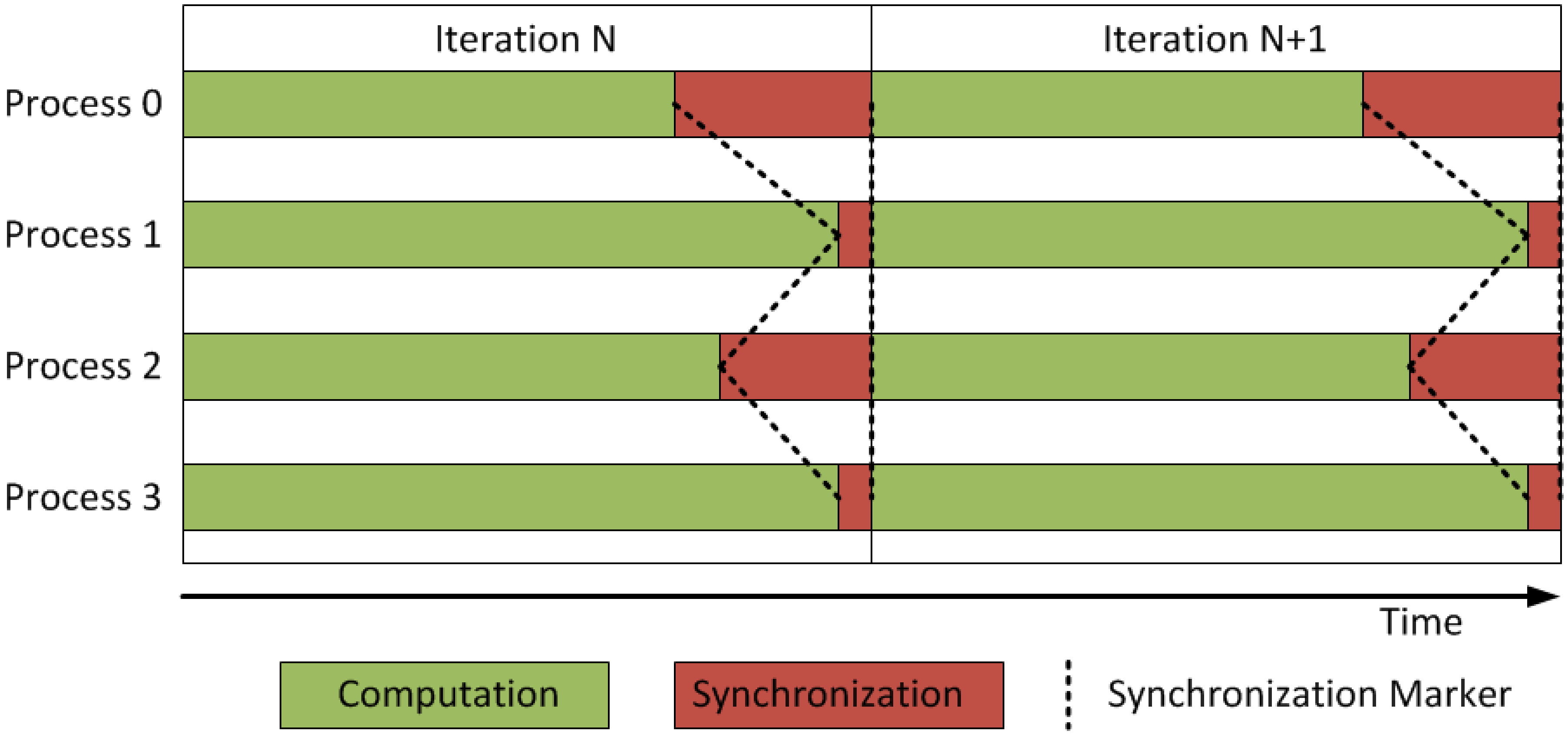
Energy Efficiency Tuning Types



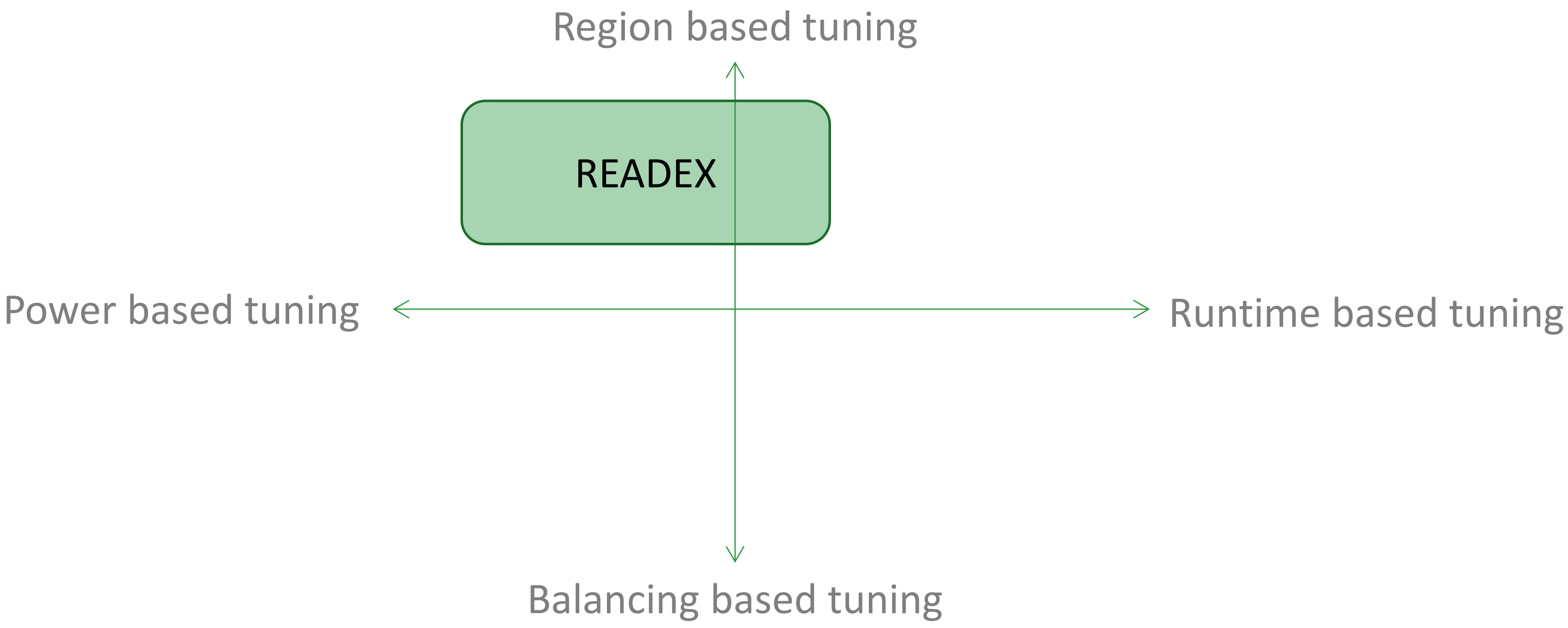
Energy Efficiency Tuning Types



Energy Efficiency Tuning Types



Energy Efficiency Tuning with READEX



Terminology: Region and Region Instance

```
int main(void) {

    // Initialize application
    // Initialize experiment variables

    int num_iterations = 2;
    for (int iter = 1; iter <= num_iterations; iter++) {
       // Start phase region
       // Read PhaseCharct
       laplace3D();              // significant region
       residue = reduction(); // insignificant region
       fftw_execute();           // significant region
       // End phase region
    }

    // Post-processing:
    // Write noise matrices to disk for visualization
    // Terminate application

    MPI_Finalize();
    return 0;
}
```

Phase region

Phase

Scenario

FREQ=2 GHz

FREQ=1.5 GHz

Significant region

Runtime situation

Workflow

1. Instrument application

   Score-P provides different kinds of instrumentation

2. Detect dynamism

   Check whether runtime situations could benefit from tuning

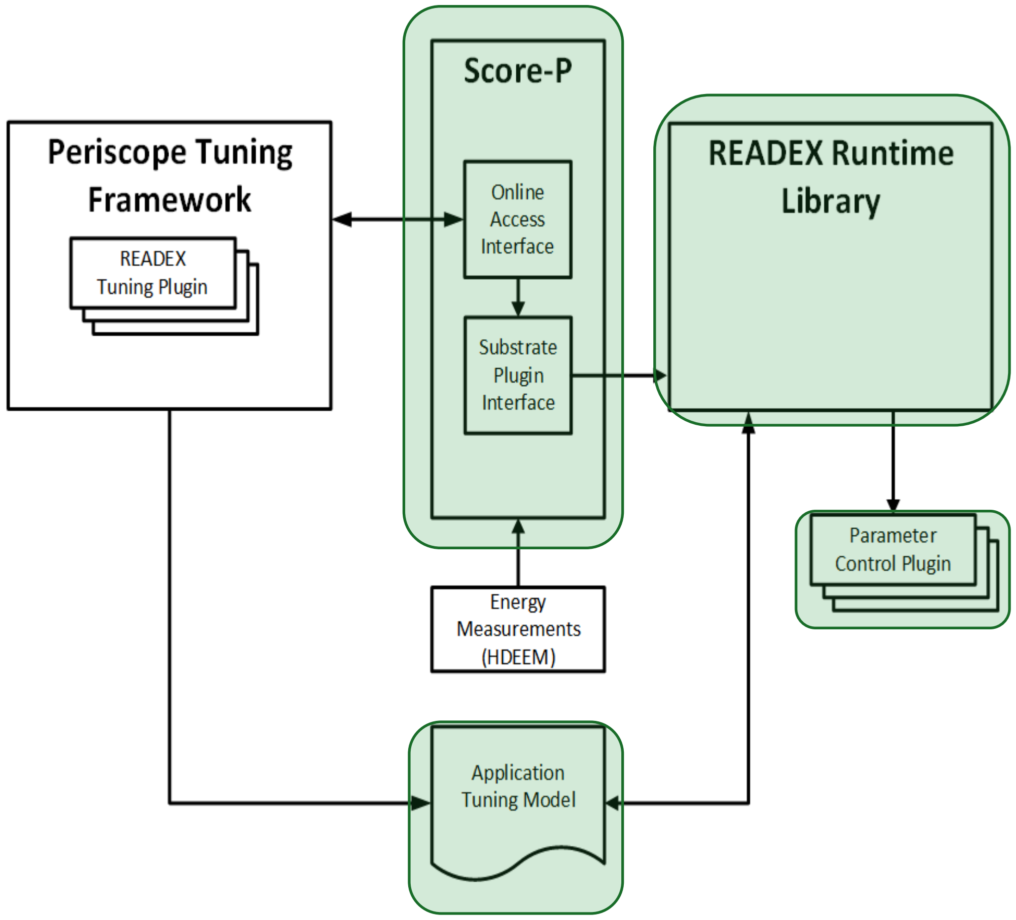
3. Detect energy saving potential and configurations (DTA)

   Use tuning plugin and power measure-ment infrastructure to search for opti-mal configuration

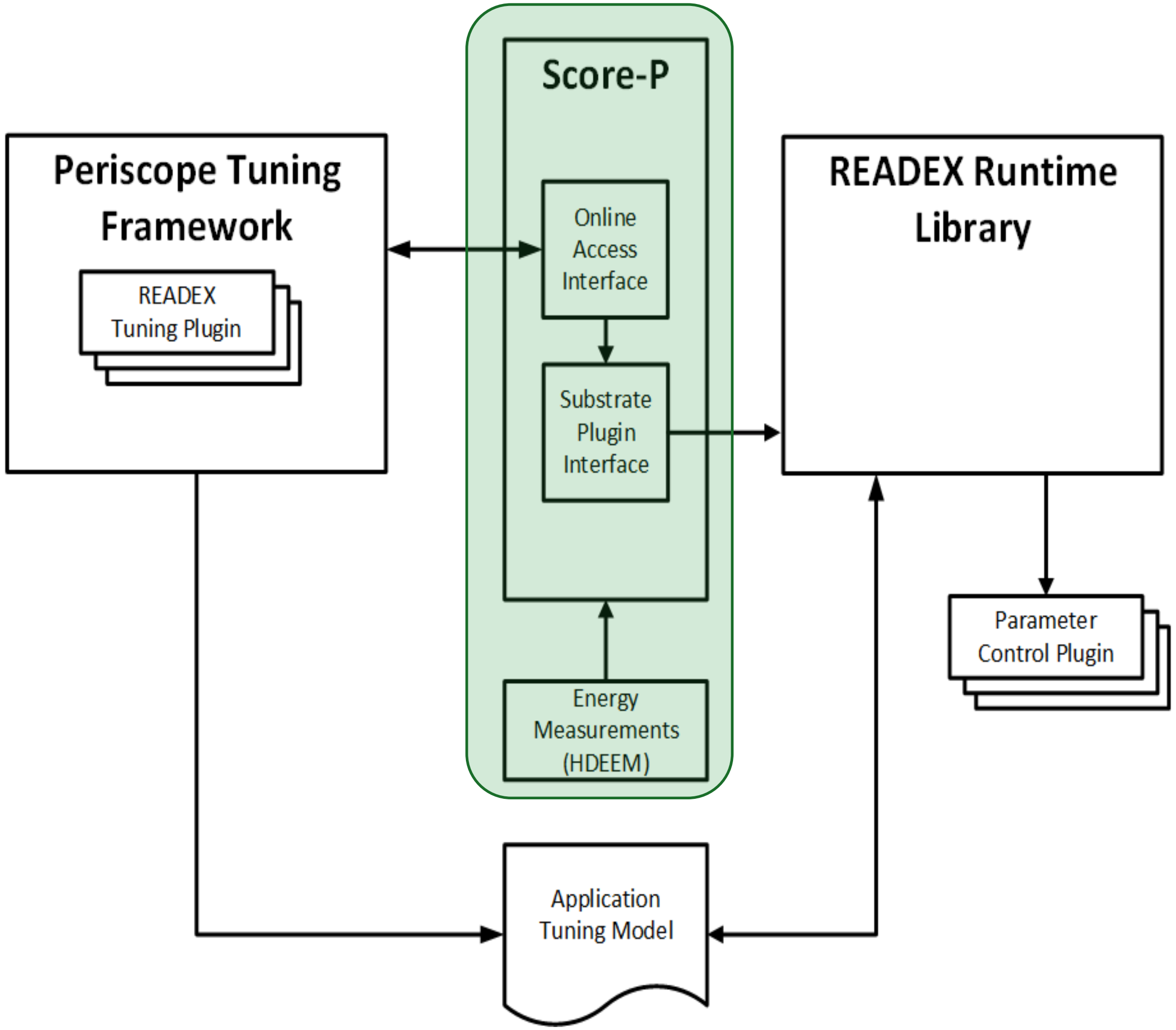
   Create tuning model

4. Runtime application tuning (RAT)

   Apply tuning model, use optimal configuration



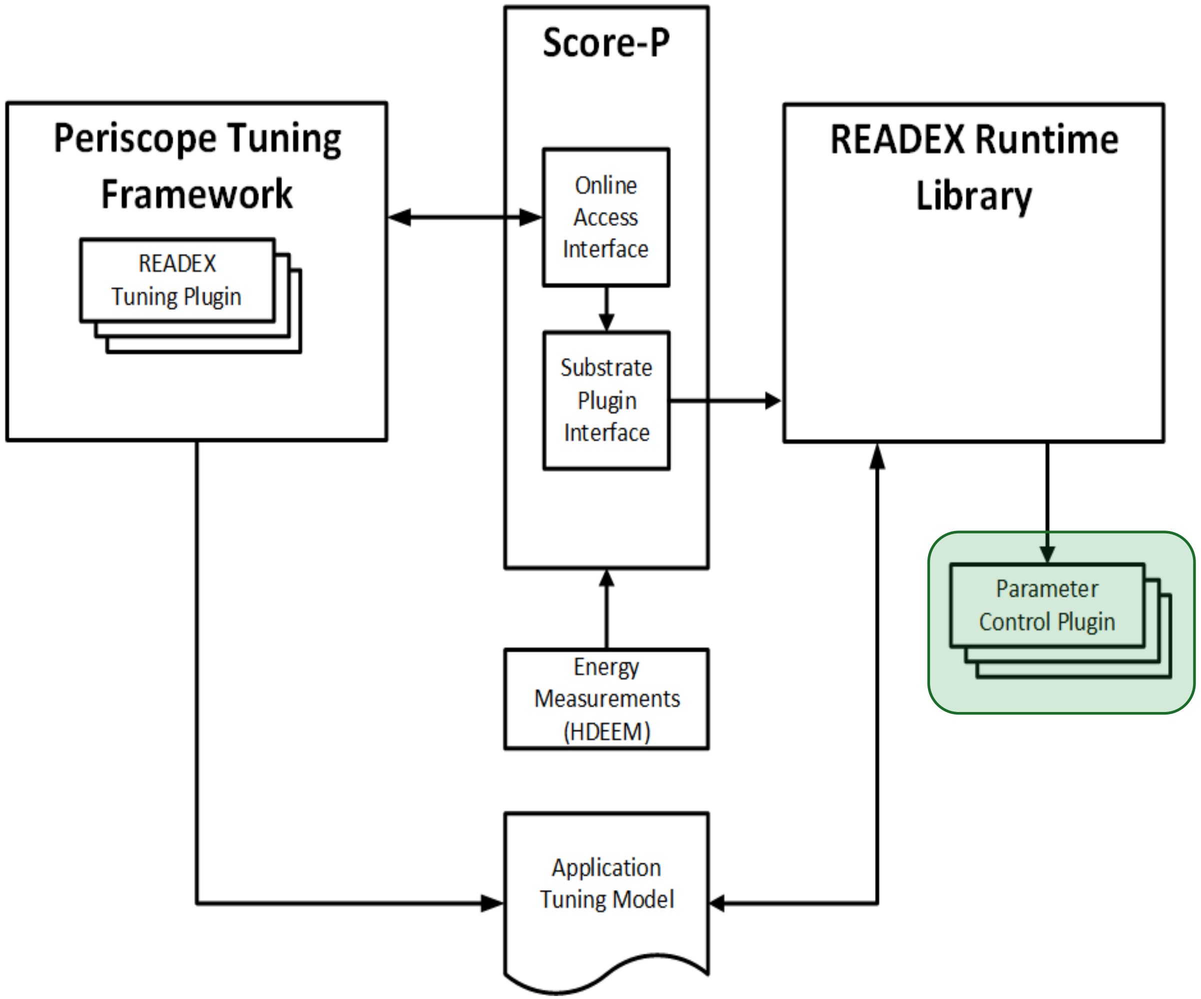
Instrumentation via Score-P



Instrumentation via Score-P

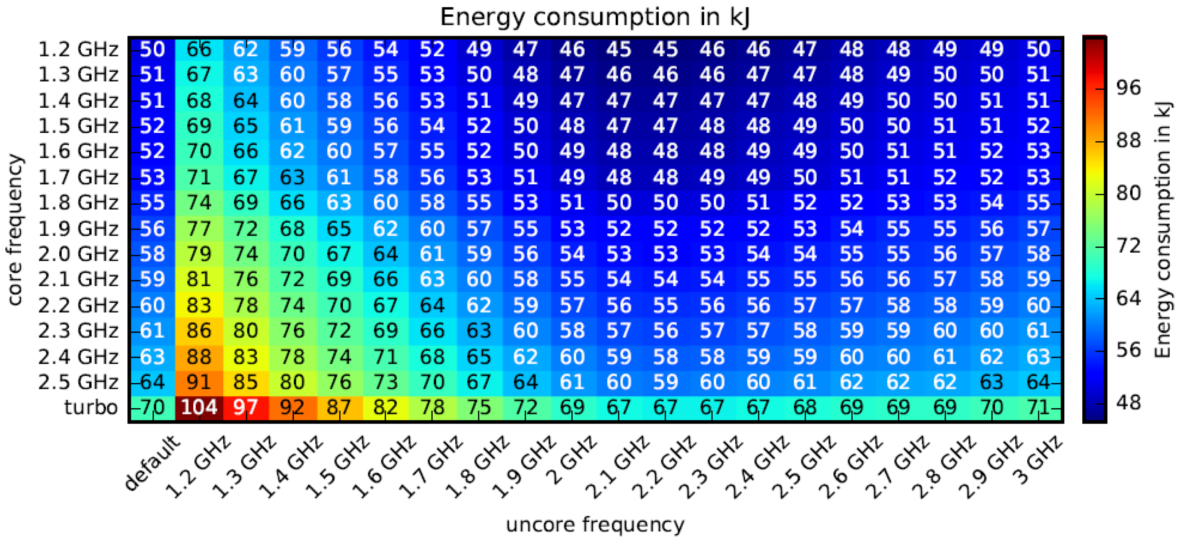
- HPC performance measurement infrastructure
  - Creates CUBEx profiles or OTF2 traces
  - Instrumentation and sampling
  - Supports most HPC programming paradigms
  - Mechanism for online usage of data – Periscope
  - Efficient implementation
  - Power measurement plugins (see talk by T. Ilsche)
- Re-use and extend existing infrastructure
  - Parse CUBEx profiles to find significant regions
  - Support tools to lower measurement overhead via filtering
  - Score-P Substrate Plugin Interface for alternative use-cases
  - READEX Runtime Library (RRL) to change parameters

Toggling Parameters



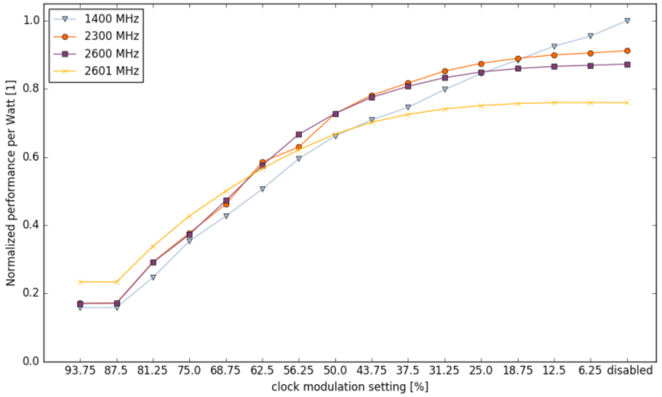
Toggling Parameters

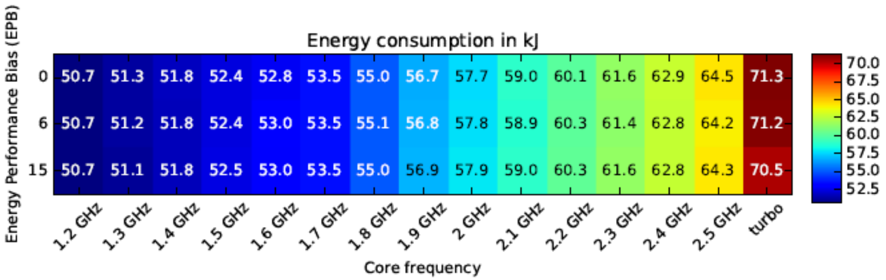
- Hardware parameters
  - Core frequency, uncore frequency

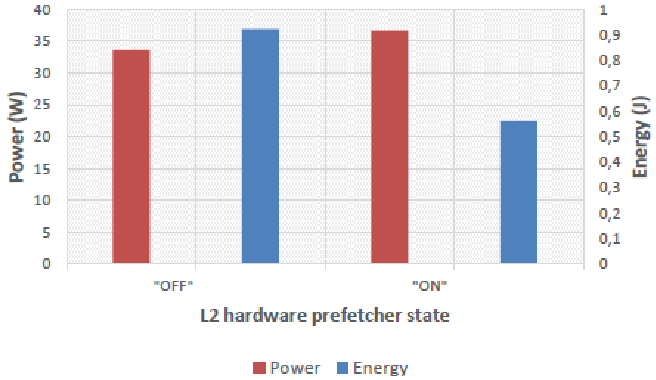


Toggling Parameters

- Hardware parameters
  - Core frequency, uncore frequency
  - Clock modulation, Energy Performance Bias, prefetchers

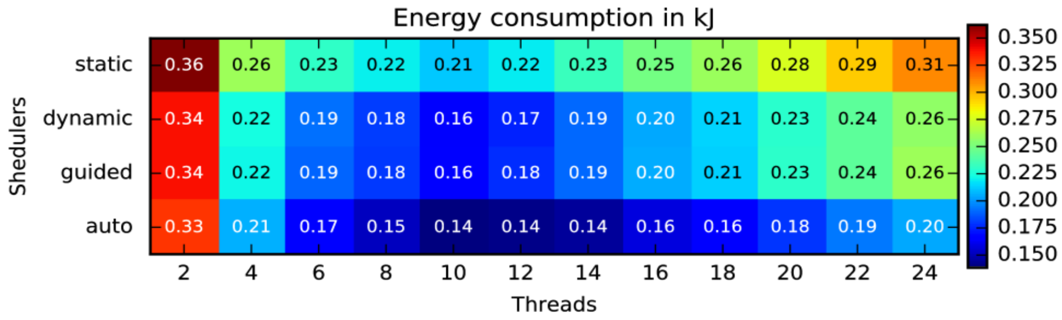






Toggling Parameters

- Hardware parameters
  - Core frequency, uncore frequency
  - Clock modulation, Energy Performance Bias, prefetchers
- Runtime parameters
  - Message Passing Interface, e.g., message size threshold
  - OpenMP parameters, e.g., loop scheduler, number of threads



Toggling Parameters

- Hardware parameters
  - Core frequency, uncore frequency
  - Clock modulation, Energy Performance Bias, prefetchers
- Runtime parameters
  - Message Passing Interface, e.g., message size threshold
  - OpenMP parameters, e.g., loop scheduler, number of threads
- Application tuning parameters

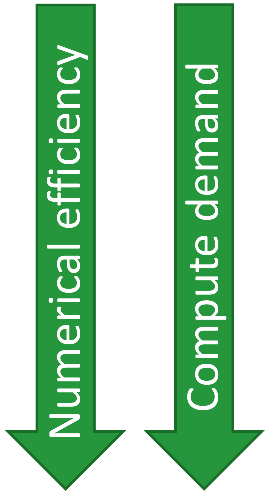
Application Tuning Parameters (Work in Progress)

```
// C example
// register parameters at READEX
ATP_PARAM_DECLARE("PARAMETER1", ATP_PARAM_TYPE_RANGE, 1, "Domain1");
// declare set of possible values for the parameter
ATP_PARAM_ADD_VALUES("PARAMETER1", values_array, num_values, "Domain1")
// getting parameter setting from READEX, store in variable app_param
ATP_PARAM_GET("PARAMETER1",&app_param,"Domain1")
// … usage of app_param, e.g., switch (app_param) {
```

Application Tuning Parameters (Work in Progress)

Application parameters example: different preconditioners in ESPRESO solver

- Full Dirichlet preconditioner is usually the preferred one (the best numerical properties)
- Depends on input dataset / problem that is solved
- All preconditioners have been evaluated with the optimal hardware parameter settings

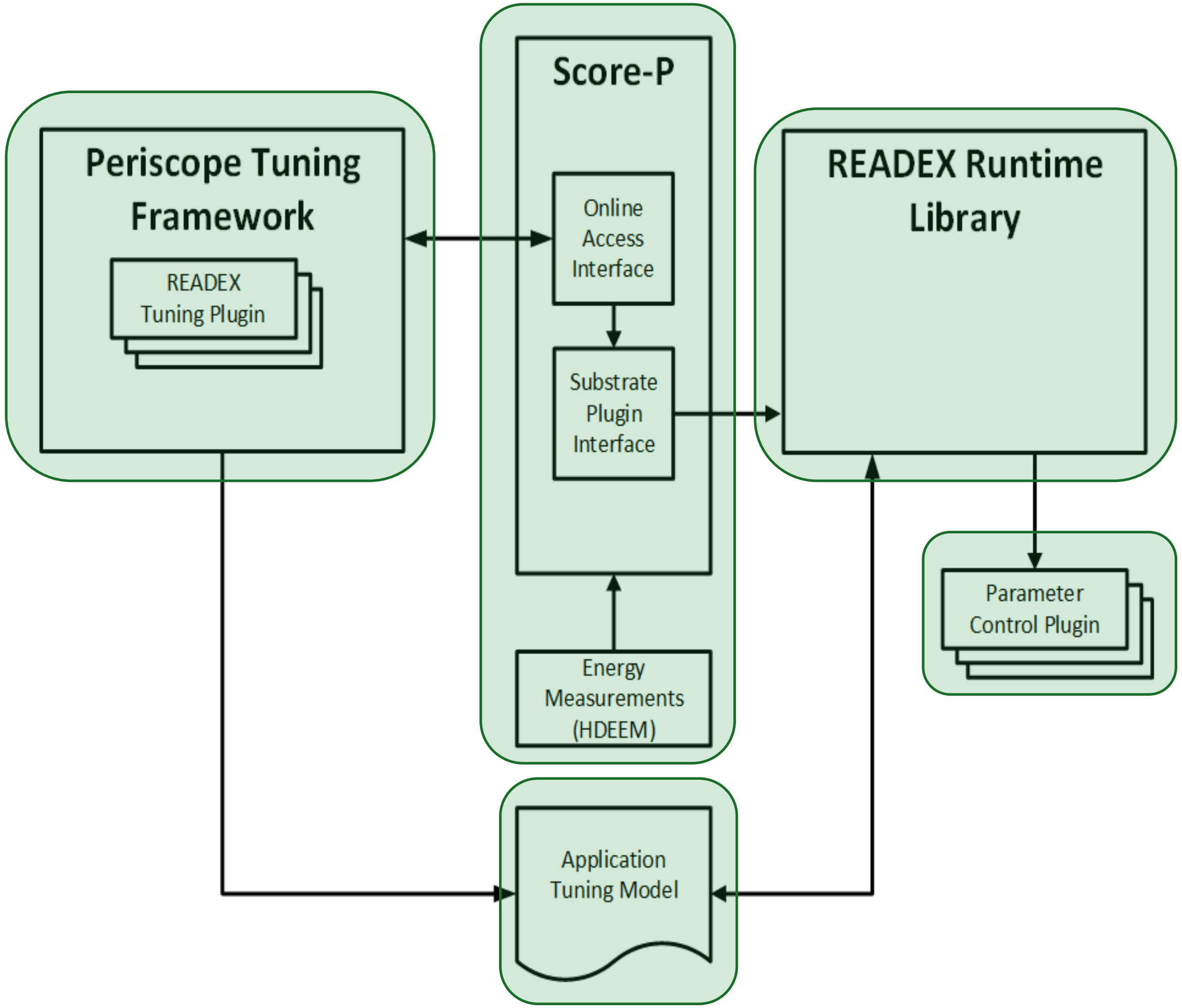


Preconditioner type	Number of iterations	Single iteration cost Time and energy		Total solution cost Time and energy	
No preconditioner	172	130 + 0 ms	32.3 + 0.00 J	21.4 s	5.50 kJ
Weight function	100	130 + 2 ms	32.3 + 0.53 J	12.9 s	3.28 kJ
Lumped	45	130 + 10 ms	32.3 + 3.86 J	6.3 s	1.64 kJ
Light Dirichlet	**39**	**130 + 10 ms**	**32.3 + 3.74 J**	**5.5 s**	**1.41 kJ**
Full Dirichlet (default)	**30**	**130 + 80 ms**	**32.3 + 20.6 J**	**6.3 s**	**1.59 kJ**

11.3% energy savings against the default full Dirichlet preconditioners

Note: 130 ms and 32.3 J – is a baseline for single iteration cost without preconditioner

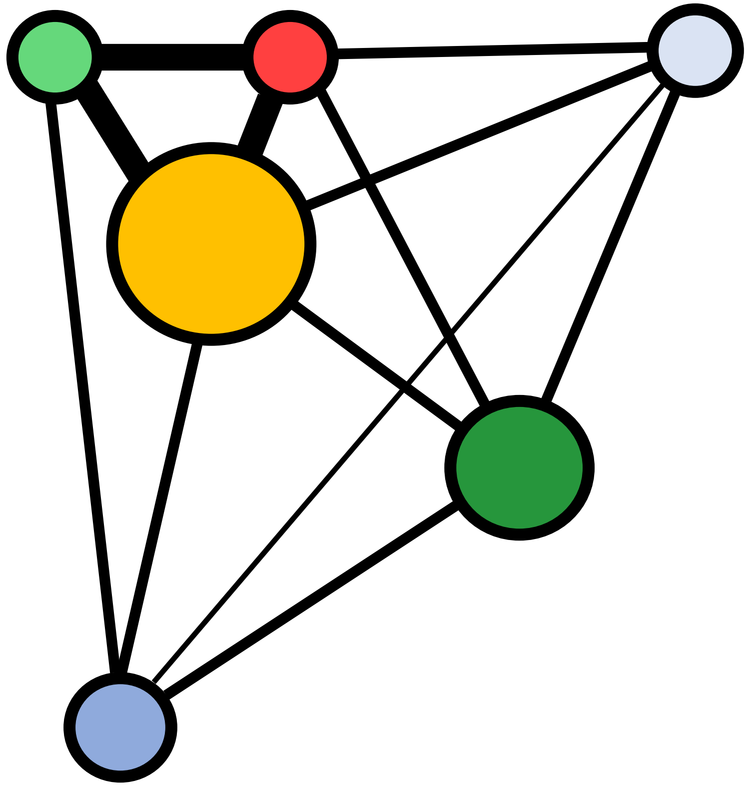
Design Time Analysis



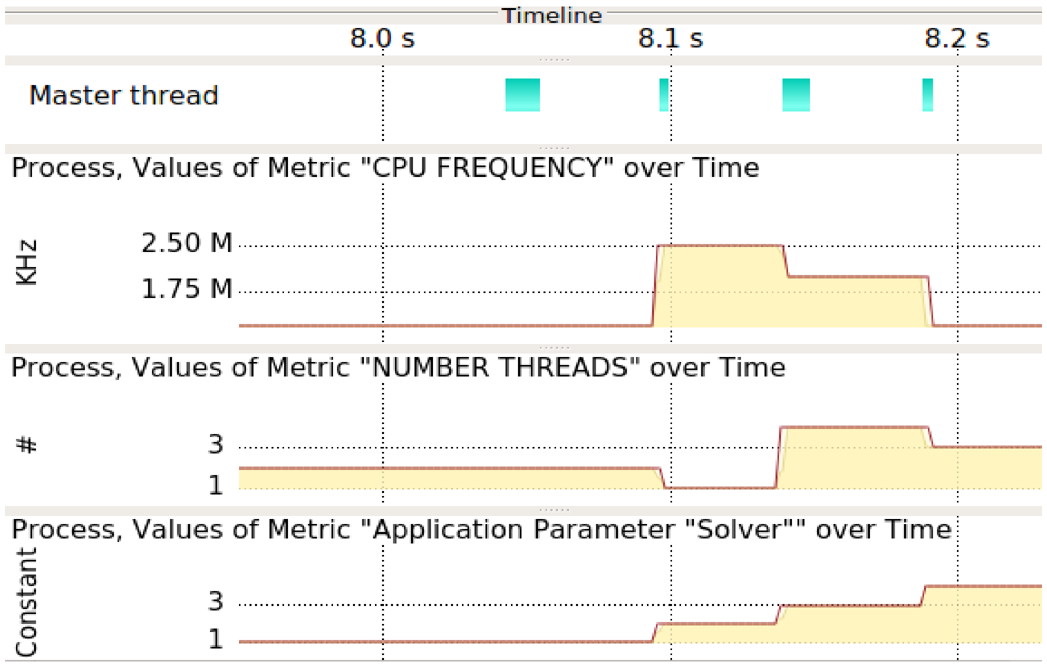
Design Time Analysis

- Periscope Tuning Framework
- Pre-computation of dynamicity and significant regions
- Different objectives (e.g., runtime, energy, EDP)
- Different search strategies (complete, random, genetic)
- Uses RRL to switch parameters per phase
- Determine intra- and inter-phase dynamism
- Determine best configuration for significant regions
- Current work in progress:
    - Cluster regions in scenarios
    - Application tuning parameters

Tuning Model Visualization

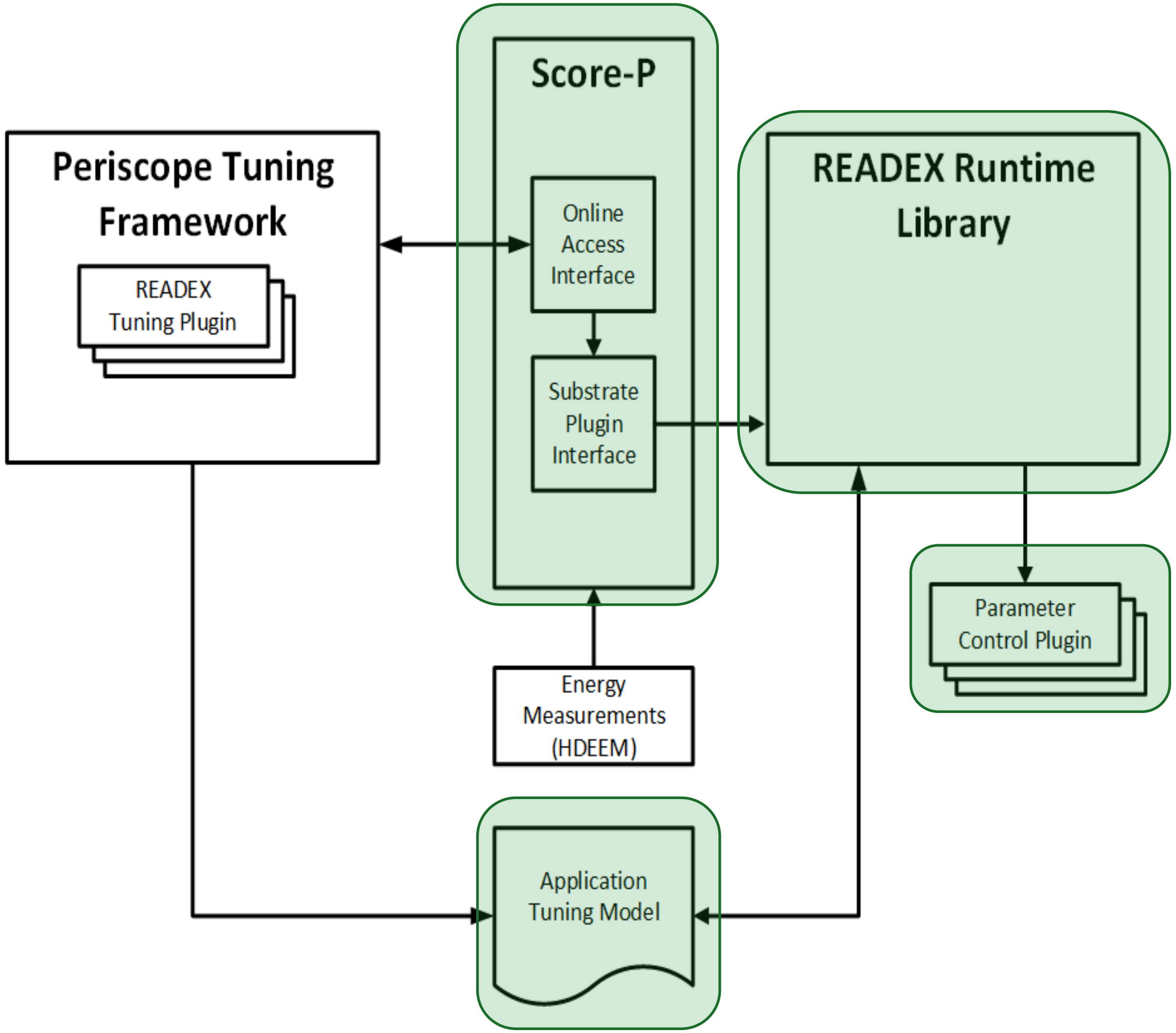


Force graph for scenarios



Vampir visualization of parameter changes

Runtime Tuning



Runtime Tuning

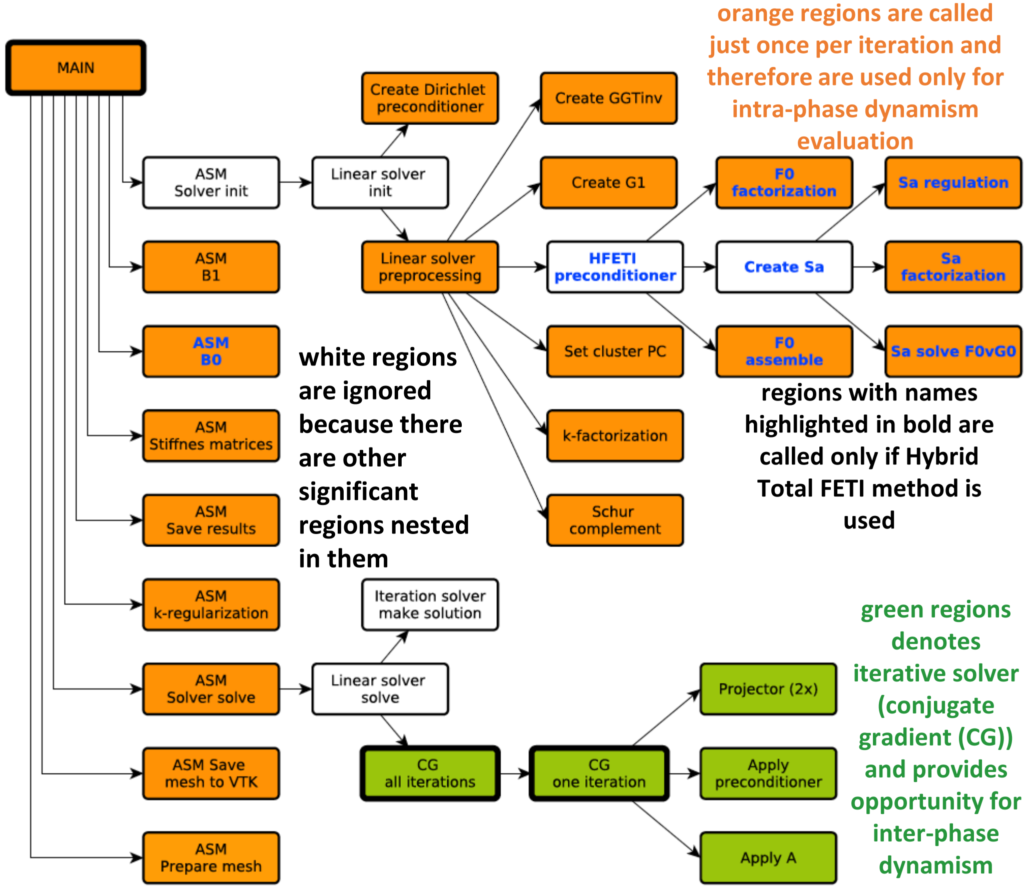
- READEX Runtime Library
- Reads and applies Tuning Model
- Sets and resets configuration at runtime
- Current work in progress:
    - Application tuning parameters
    - Online calibration mechanism
    - Advanced switching decision making

Tuning Potential

	static	dynamic	total
ESPRESO:	**12.3 %** +	**9.1 %**	= **20.3 %**

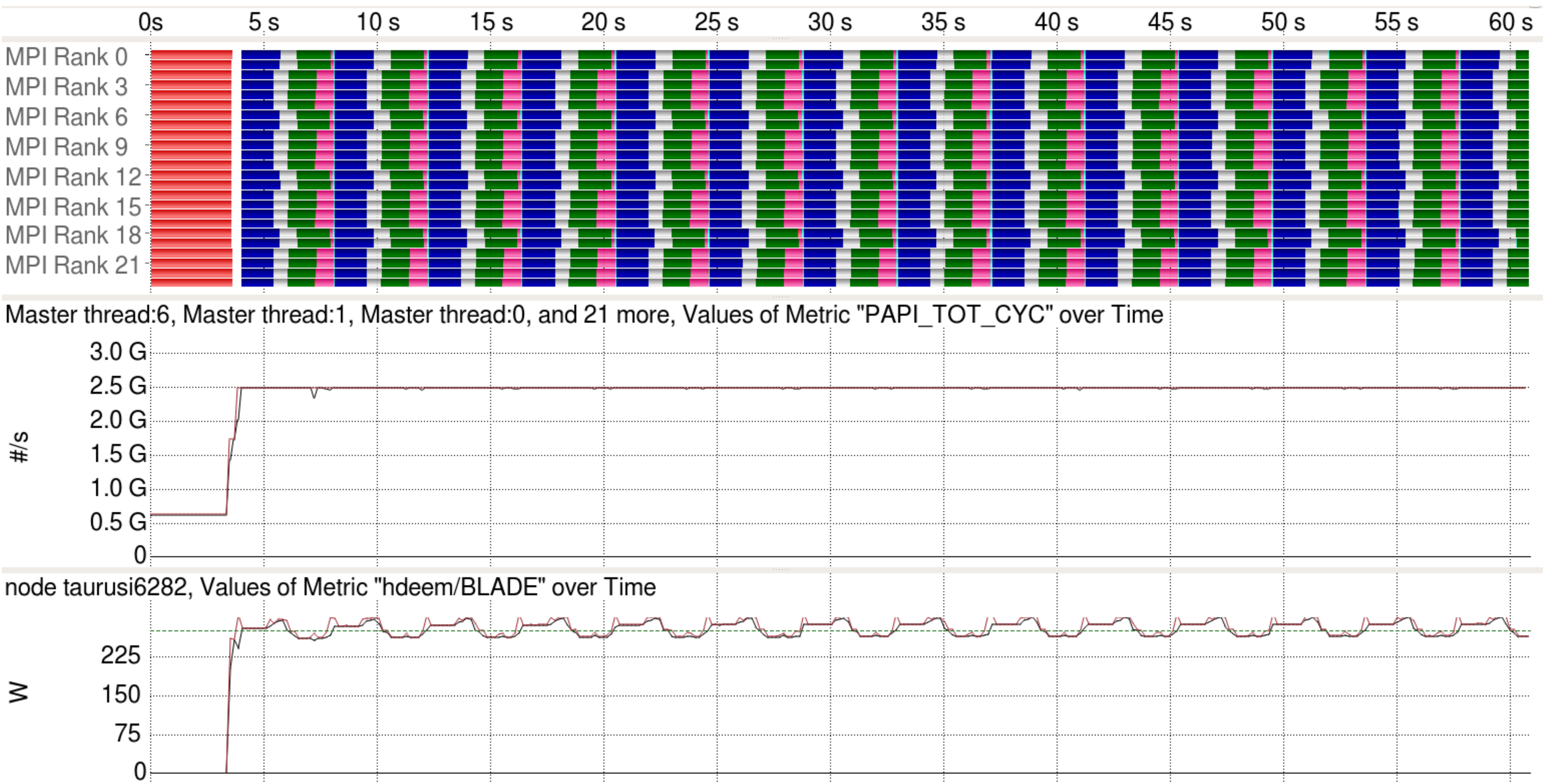
- Structural mechanics code
- Finite element + sparse FETI solver

Region	% of 1 phase	Best static configuration	Value	Best dynamic configuration	Value	Dynamic savings
Assembler–AssembleStiffMat	14.32	18 threads, 1.8 GHz UCF, 2.5 GHz CF	733.73 J	20 threads, 2.0 GHz UCF, 2.5 GHz CF	731.22 J	2.51 J (0.34%)
Assembler–Assemble-B1	2.23	18 threads, 1.8 GHz UCF, 2.5 GHz CF	114.30 J	2 threads, 2.2 GHz UCF, 2.5 GHz CF	94.15 J	20.15 J (17.63%)
Cluster–CreateF0-FactF0	0.17	18 threads, 1.8 GHz UCF, 2.5 GHz CF	8.71 J	6 threads, 1.6 GHz UCF, 2.5 GHz CF	6.90 J	1.80 J (20.73%)
Assembler–SaveResults	3.10	18 threads, 1.8 GHz UCF, 2.5 GHz CF	158.81 J	2 threads, 1.2 GHz UCF, 2.5 GHz CF	147.66 J	11.16 J (7.03%)
			...			
Cluster–CreateSa-SaReg	0.17	18 threads, 1.8 GHz UCF, 2.5 GHz CF	8.59 J	8 threads, 2.0 GHz UCF, 2.5 GHz CF	7.03 J	1.56 J (18.15%)
Total value for static tuning for significant regions			733.73 + 114.30 + 8.71 + 158.81 + 278.39 + 113.87 + 14.23 + 658.07 + 325.69 + 99.93 + 74.70 + 641.88 + 1578.06 + 13.28 + 24.20 + 278.22 + 8.59 = 5124.66 J			
Total savings for dynamic tuning for significant regions			2.51 + 20.15 + 1.80 + 11.16 + 47.01 + 16.41 + 5.31 + 28.45 + 29.03 + 19.08 + 0.16 + 2.49 + 288.21 + 0.77 + 1.88 + 23.24 + 1.56 = 499.22 J of 5124.66 J (9.74 %)			
Dynamic savings for application runtime			499.22 J of 5493.55 J (9.09 %)			
Total value after savings			4994.33 J (79.72 % of 6265.18 J)			

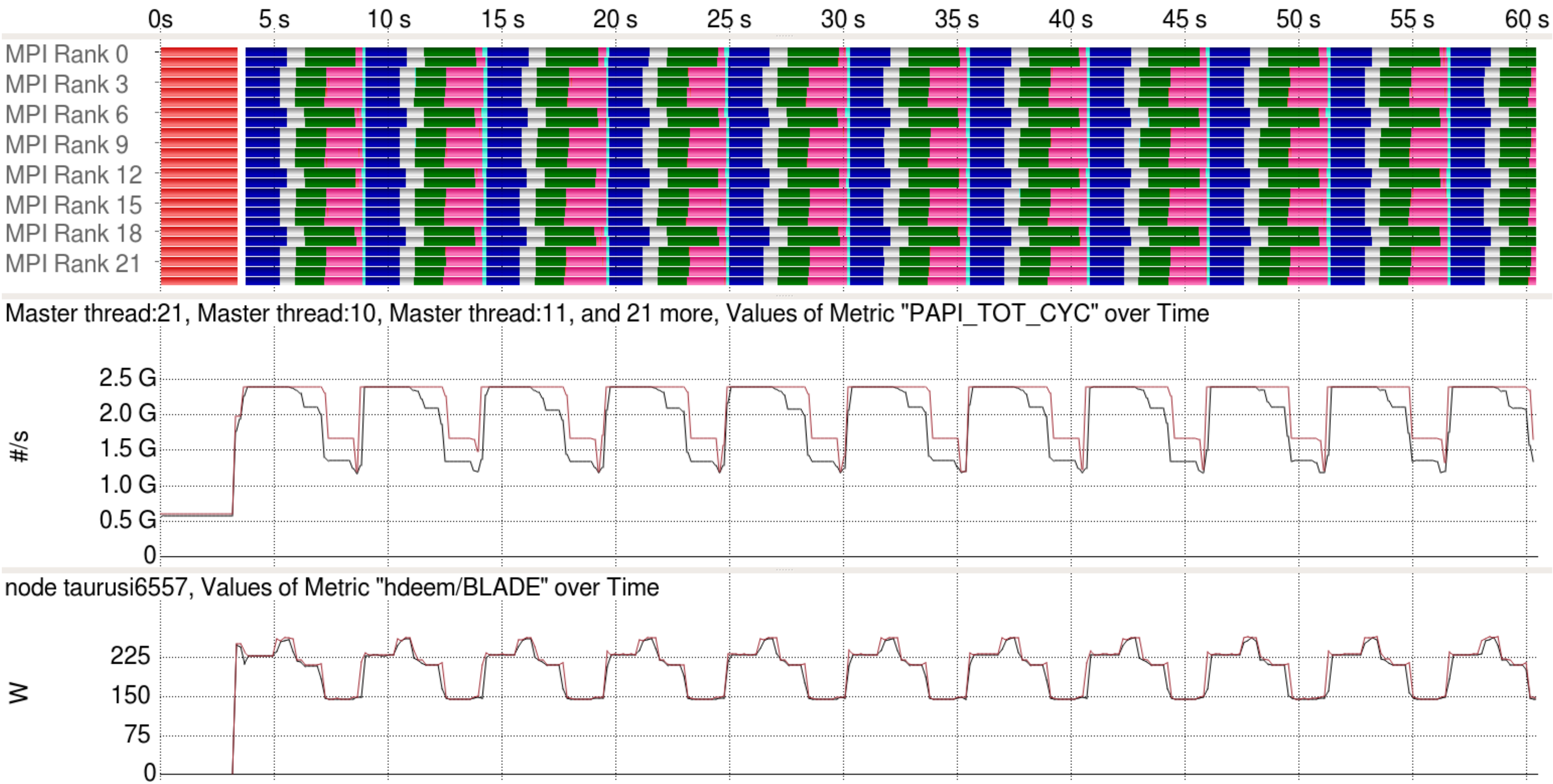


	Default settings	Default values	Best static configuration	Static Savings	Dynamic Savings
Energy consumption [J] , Blade summary	24 threads, 3.0 GHz UCF, 2.5 GHz CF	6265.18 J	18 threads, 1.8 GHz UCF, 2.5 GHz CF	771.63 J (12.32%)	499.2 J of 5493.6 J (9.09 %)
Runtime of function [s], Job info - hdeem	24 threads, 3.0 GHz UCF, 2.5 GHz CF	29.55 s	22 threads, 3.0 GHz UCF, 2.5 GHz CF	0.01 s (0.04%)	0.82 s of 29.54 s (2.76 %)

Alpha Prototype Results – Kripke



Alpha Prototype Results – Kripke



Summary

- Region-based offline and online energy efficiency tuning
- Four steps:
  - Instrumentation
  - Preparation
  - Design Time Analysis
  - Runtime Tuning
- Alpha Prototype available
- Current work:
  - Application Tuning Parameters
  - Online Calibration

Discussion


READEX
application
tuning
system
systems
applications
energy-efficiency
performance
HPC
energy
dynamic
auto-tuning
results
dynamism
knowledge
research
methodology
European
analysis
run-time
partners
scenario
used
data
user
parameters
configurations
manual
framework
automatic
Exascale
users
scenarios
based
well
community
information
technologies
process
potential
developed
computing
activities
future
provide
tools
industry
Suite
also
Tool
new
impact
improve
approach
develop
time
using