Optimizing the Energy Efficiency of High Performance Scientific Computing Software

Kai Diethelm

GNS Gesellschaft für numerische Simulation mbH
Braunschweig, Germany

GAMM CSE Workshop 2016
Universität Kassel, 08–09 September 2016

Acknowledgement

The work described in the talk is part of the projects READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) and Score-E.

The project Score-E is financially supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IH13001.

SPONSORED BY THE Federal Ministry of Education and Research

The project READEX has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 671657.

European Commission | Horizon 2020 European Union funding for Research & Innovation

Partners


JÜLICH
FORSCHUNGSZENTRUM

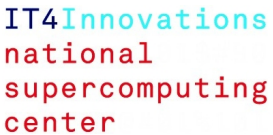

RWTH AACHEN
UNIVERSITY


TECHNISCHE
UNIVERSITÄT
DRESDEN


intel


TECHNISCHE
UNIVERSITÄT
DARMSTADT


OÉ Gaillimh
NUI Galway

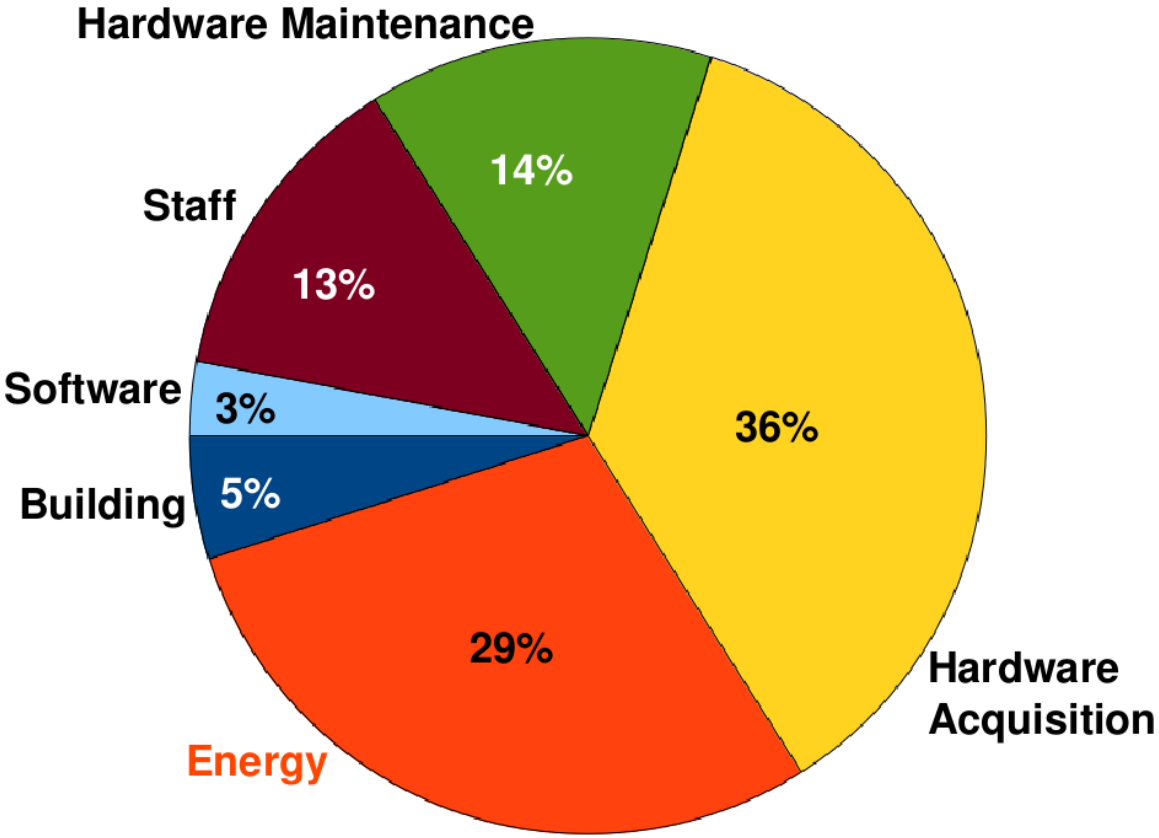

IT4Innovations
national
supercomputing
center


NTNU

Outline

Initial Observation

Distribution of Total Cost of Ownership of HPC System



Energy cost is already ∼30% of total cost, with rising tendency.

Source: Survey conducted by RWTH Aachen (2012)

Challenges

- HPC centers
  - are forced to develop strategies for energy efficient modes of operation,
  - will ask users (e.g., the scientific computing community) to run their codes in an energy efficient manner.

Challenges

- HPC centers
  - are forced to develop strategies for energy efficient modes of operation,
  - will ask users (e.g., the scientific computing community) to run their codes in an energy efficient manner.

- HPC users
  - are likely to be billed for energy instead of CPU time in the future,
  - must find out how much energy their codes need to run,
  - need to optimize their codes with respect to energy.

Possible Approach

LRZ München's Strategy for SuperMUC

Set default CPU frequency significantly lower than maximum. Allow use of higher frequency only if users can demonstrate that it pays off.

Possible Approach

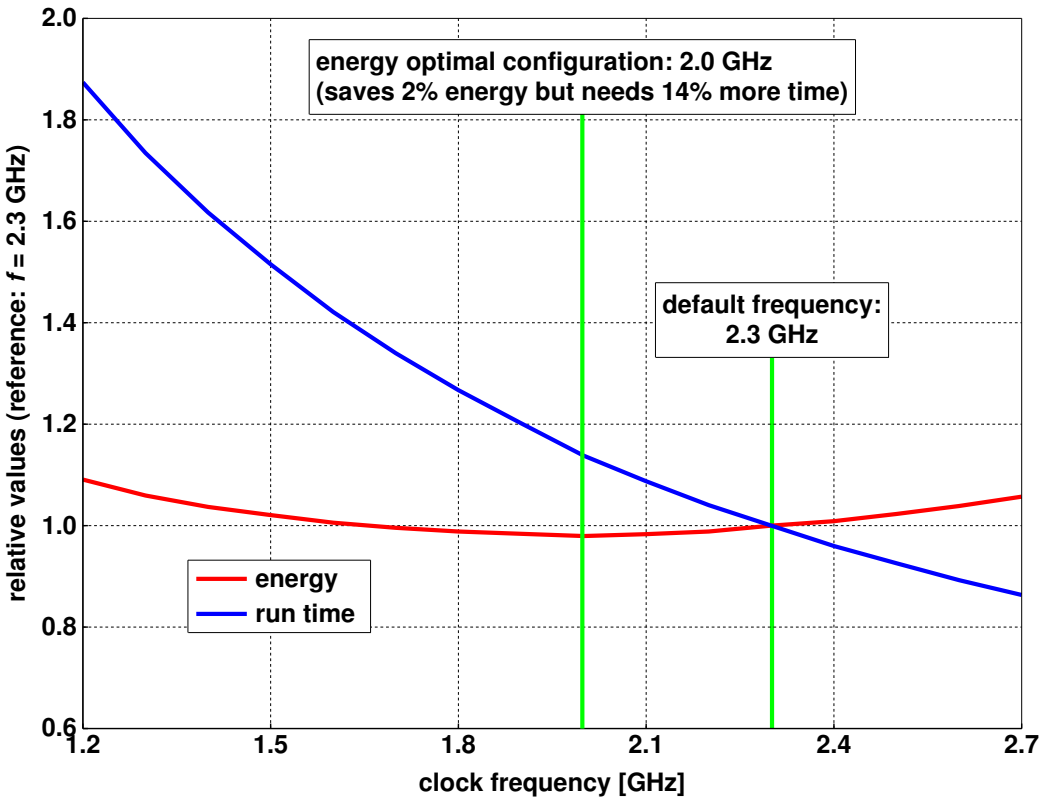
LRZ München's Strategy for SuperMUC

Set default CPU frequency significantly lower than maximum. Allow use of higher frequency only if users can demonstrate that it pays off.

- Model: $E = \int_0^T (P_{\text{static}}(t) + P_{\text{dynamic}}(t))dt$
- P_{static} is constant (idle power)
- P_{dynamic} depends on frequency f and voltage U: $P_{\text{dynamic}} \sim U^2 \cdot f$
- Runtime T is non-increasing function of frequency f

Observations

- When running an application with higher frequency . . .
  - static energy usually decreases (due to shorter runtime)
  - dynamic energy usually increases
- Each application has its own optimal frequency!



When Does an Increased Frequency Pay Off?

Consequences of frequency reduction:

- Energy requirements are reduced
- Runtime is increased
- Throughput of system is decreased
- Capabilities of hardware are not fully utilized
- More investment in hardware necessary to satisfy demand

Frequency increase pays off if runtime decreases significantly

Pure focus on energy is not appropriate.

Elementary Approach

Energy-delay product:

$$\text{EDP} = E \cdot T^w$$

- E: Energy used by the application run
- T: Runtime used by the application run
- Parameter w is used to prioritize runtime
- Usually $w \in \{1, 2, 3\}$ (EDP1, EDP2 and EDP3)
- *Single* metric to optimize both energy and hardware cost
  $\Rightarrow$ useful for our purpose in spite of physical implausibility

Tools Landscape

Analysis tools for performance

- CUBE: Profiling
- Scalasca: Automatic trace analysis
- Vampir: Interactive trace analysis
- TAU: Profiling and tracing
- Periscope Tuning Framework:
  On-line analysis and tuning

Tools Landscape

Analysis tools for performance

- CUBE: Profiling
- Scalasca: Automatic trace analysis
- Vampir: Interactive trace analysis
- TAU: Profiling and tracing
- Periscope Tuning Framework:
  On-line analysis and tuning

Unified measurement infrastructure

Score-P (`http://www.score-p.org`)

Tools Landscape

Analysis tools for performance and energy requirements

- CUBE: Profiling
- Scalasca: Automatic trace analysis
- Vampir: Interactive trace analysis
- TAU: Profiling and tracing

new: visualization & analysis capabilities for energy related metrics

- Periscope Tuning Framework: On-line analysis and tuning

new: energy tuning plugins (PCAP, DVFS, MPIProcs, . . .)

Unified measurement infrastructure

Score-P (`http://www.score-p.org`)

new: interface to energy measurement hardware

Outline

Static Tuning

- Run code with a few typical example data sets and different frequencies
- Measure runtimes and energy requirements
- Select frequency with best EDP for production runs

Static Tuning

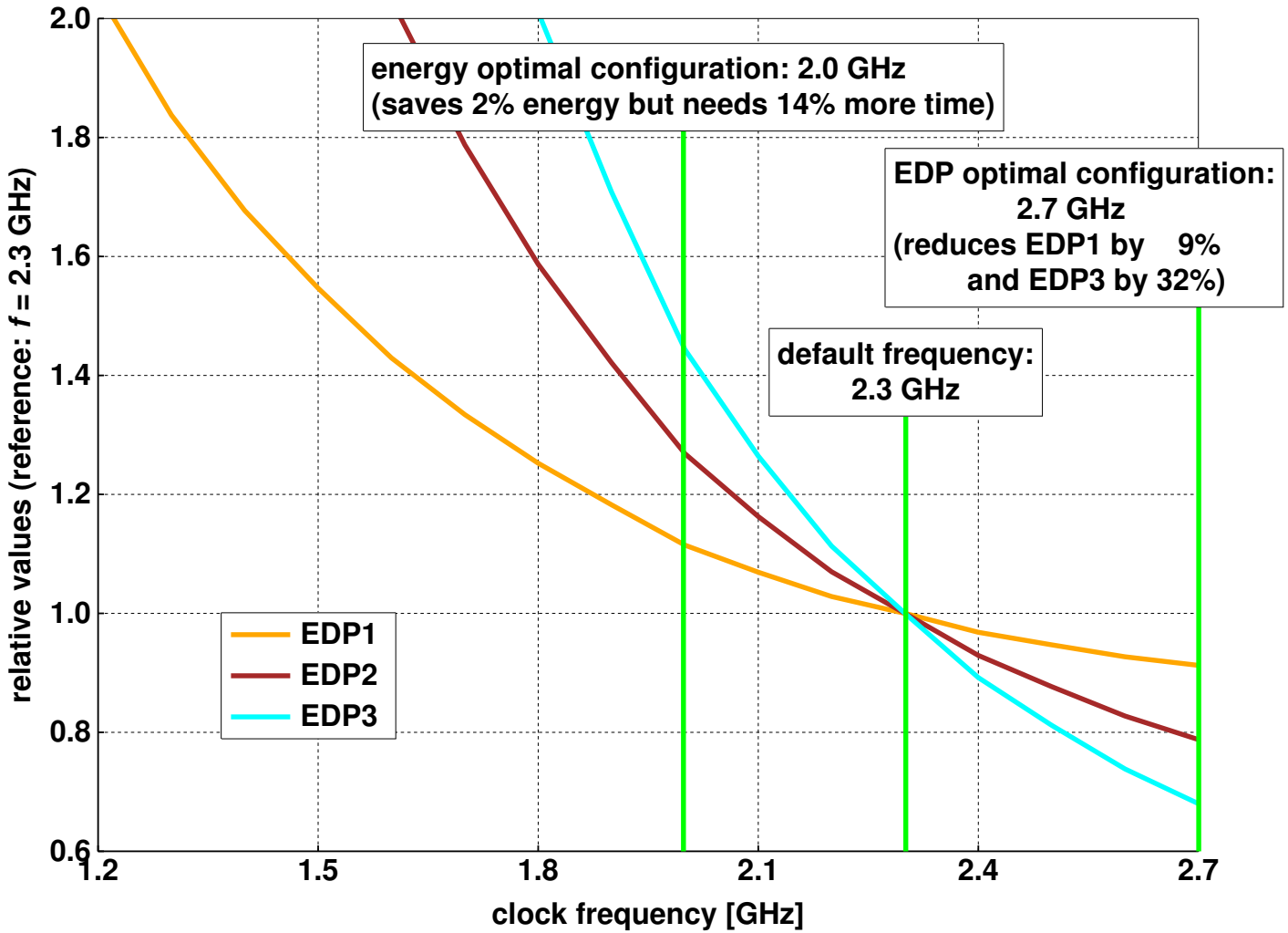
- Run code with a few typical example data sets and different frequencies
- Measure runtimes and energy requirements
- Select frequency with best EDP for production runs

Same approach can be followed for other tuning parameters:

- # of MPI tasks
- # of OpenMP threads
- …

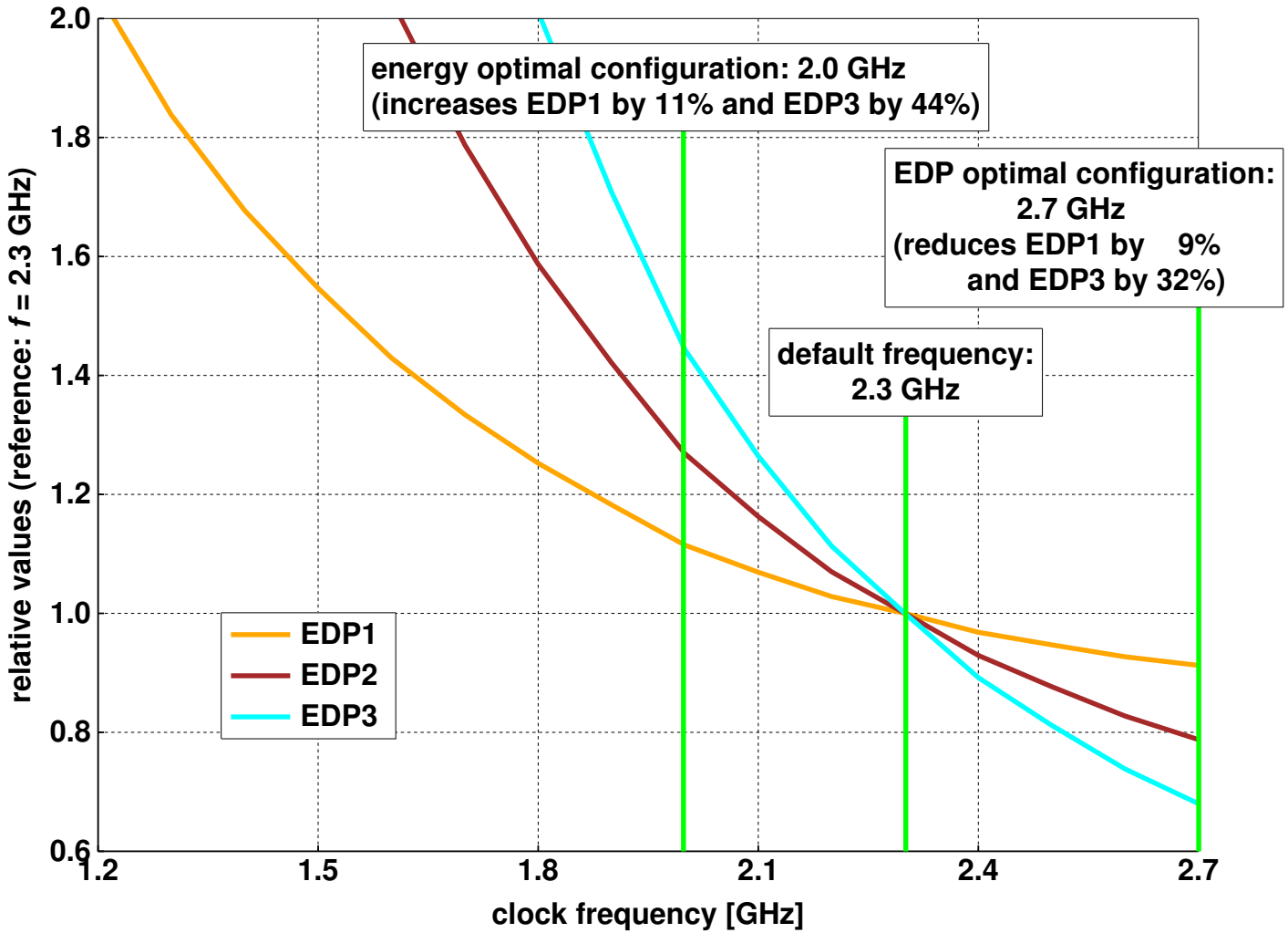
Static Tuning

Example: Indeed (FE code for sheet metal forming simulation)



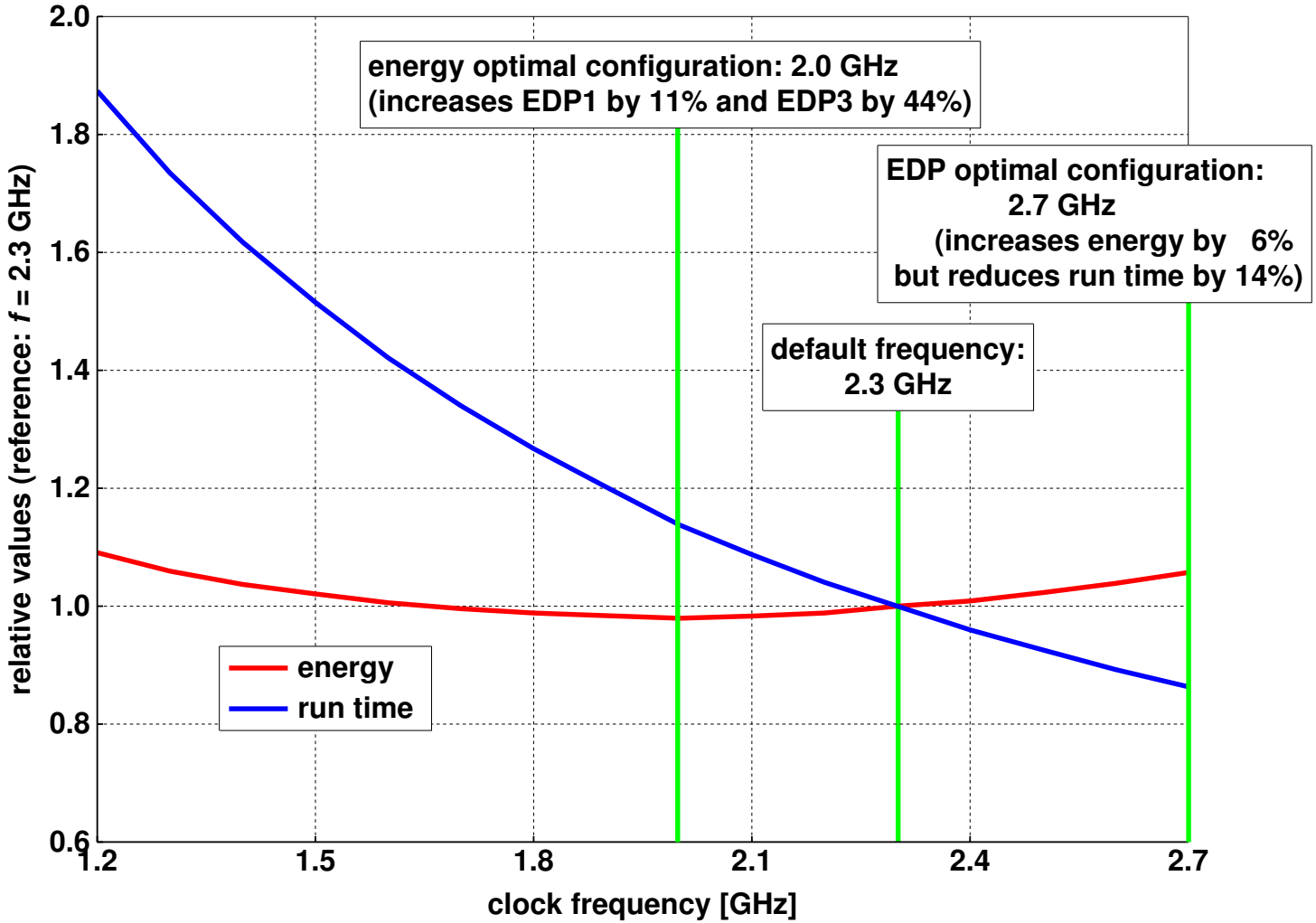
Static Tuning

Example: Indeed (FE code for sheet metal forming simulation)



Static Tuning

Example: Indeed (FE code for sheet metal forming simulation)

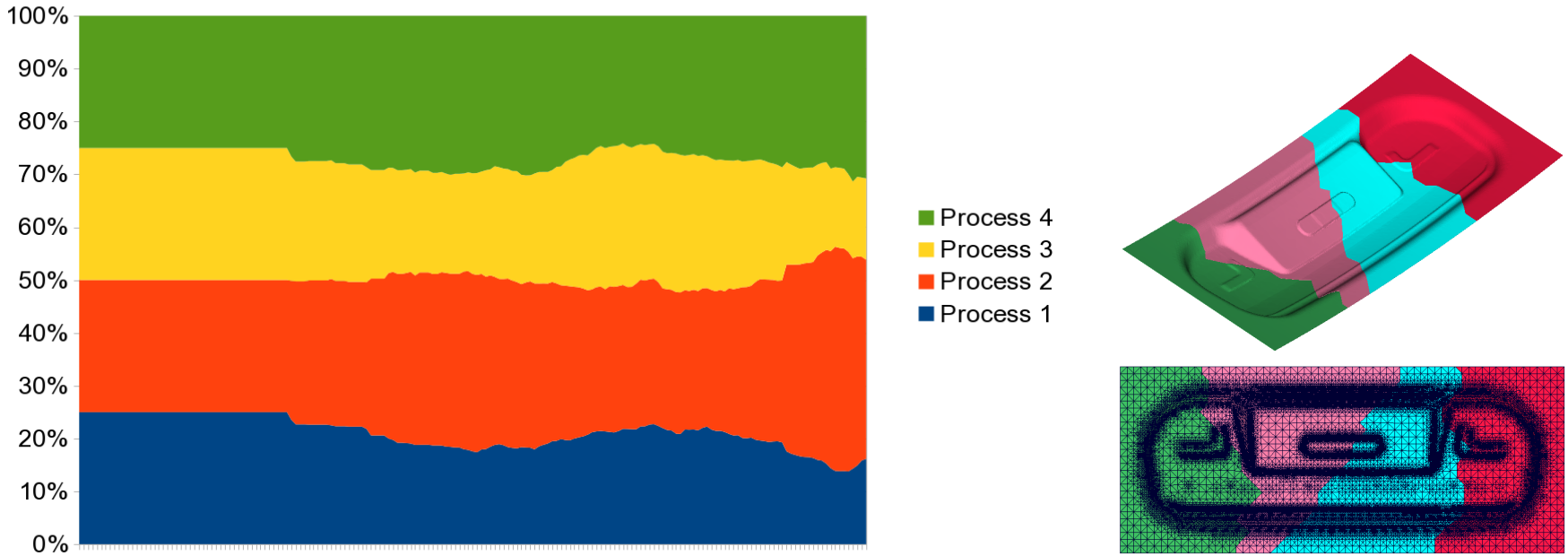


Dynamic Tuning

Fundamental Property

Many HPC codes exhibit dynamic behaviour.

Example: Distribution of workload in Indeed run (4 MPI tasks)



Dynamic Tuning

Exploit dynamic behaviour at runtime:

- reduce CPU frequency for I/O bound or memory bound parts of code
- increase CPU frequency for compute bound parts of code
- reduce # of OpenMP threads in case of lock contention
- reduce # of MPI tasks if problem size is small or if algorithm scales poorly
- ...

Dynamic Tuning

Two approaches:

Exploit inter-phase dynamism

Change parameters at each time step

Dynamic Tuning

Two approaches:

Exploit inter-phase dynamism

Change parameters at each time step

Exploit intra-phase dynamism

Split up each time step
into code regions with different requirements
and change parameters when moving from one region to next

Dynamic Tuning

Two approaches:

Exploit inter-phase dynamism

Change parameters at each time step

Exploit intra-phase dynamism

Split up each time step
into code regions with different requirements
and change parameters when moving from one region to next

Idea:

Develop tools for automatic tuning to relieve software developer from burden

Outline

The READEX Workflow

During application development (design time)

- Application instrumentation
- Application pre-analysis:
  - detect dynamism
  - find significant regions
- Identification of scenarios
- Derivation of tuning model

(based on Periscope Tuning Framework and Score-P)

During production runs (runtime)

- Run-time application tuning

(uses READEX Run-time Library)

READEX Design Time Analysis

1. Select tuning parameters
   - # of OpenMP threads
   - # of MPI processes
   - CPU frequency
   - different code paths
   - ...

READEX Design Time Analysis

1. Select tuning parameters
2. Instrument complete code

via energy related functionality of Score-P measurement infrastructure developed in Score-E project

READEX Design Time Analysis

1. Select tuning parameters
2. Instrument complete code
3. Filter fine grained code regions to reduce measurement overhead; automatic tool has been developed in READEX project

READEX Design Time Analysis

1. Select tuning parameters
2. Instrument complete code
3. Filter fine grained code regions
4. Detect regions with dynamism

and thus identify potential switching points for tuning parameters (automatic tool from READEX project)

READEX Design Time Analysis

1. Select tuning parameters
2. Instrument complete code
3. Filter fine grained code regions
4. Detect regions with dynamism
5. Identify optimal parameters for relevant runtime situations

by means of measurements via Periscope Tuning Framework

READEX Design Time Analysis

1. Select tuning parameters
2. Instrument complete code
3. Filter fine grained code regions
4. Detect regions with dynamism
5. Identify optimal parameters for relevant runtime situations
6. Cluster runtime situations with similar optimal configurations in scenarios and store them in database for exploitation during production runs

READEX Runtime Application Tuning (in progress)

At switching points identified in design time analysis phase:

- Check current runtime situation (call path, input data) for proximity to scenario in database
- Automatically switch tuning parameters to values found to be optimal for currently encountered scenario
  (via READEX runtime library)

READEX Runtime Application Tuning (in progress)

At switching points identified in design time analysis phase:

- Check current runtime situation (call path, input data) for proximity to scenario in database
- Automatically switch tuning parameters to values found to be optimal for currently encountered scenario
  (via READEX runtime library)

Advantages:

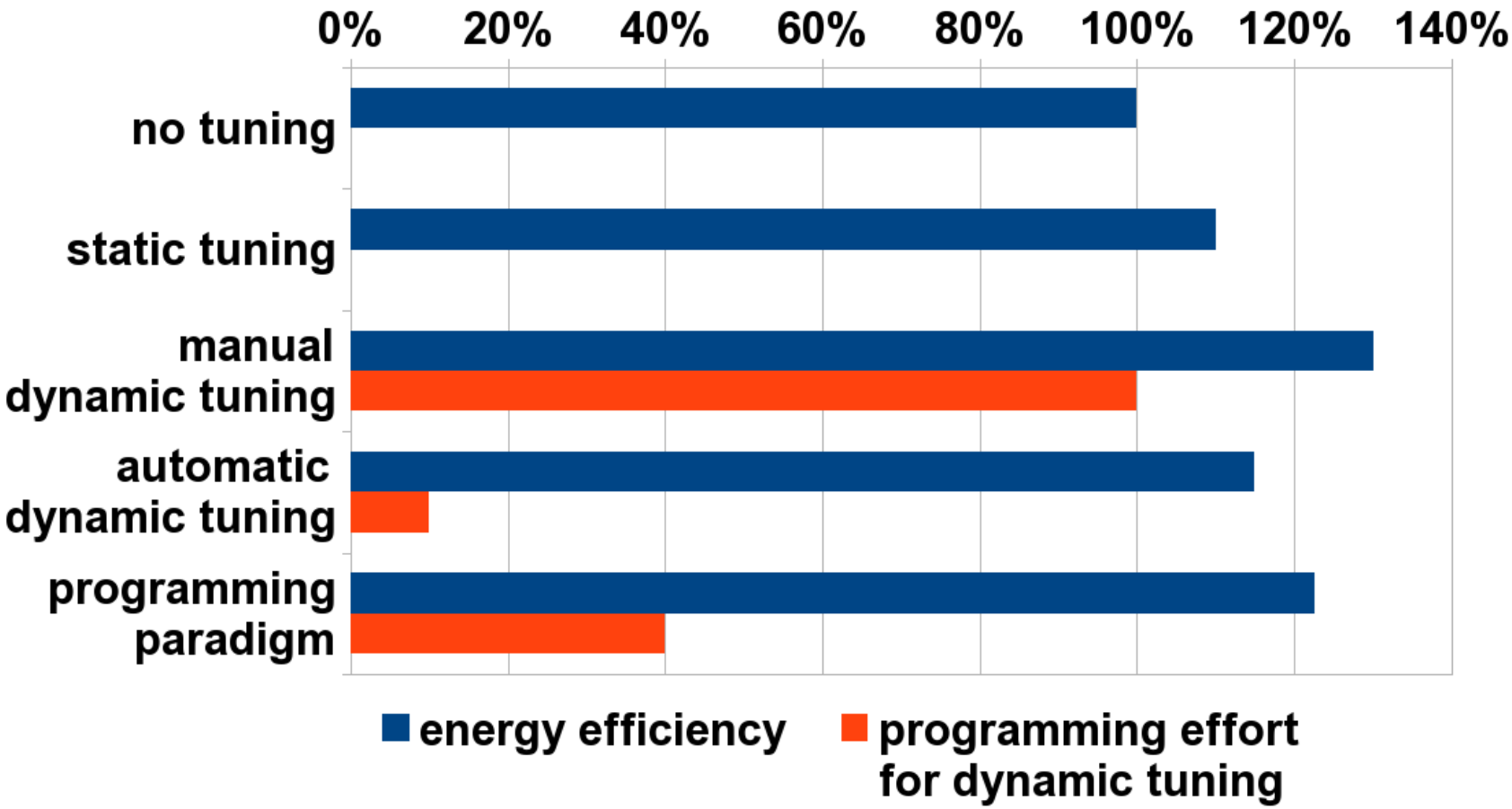
- Platform-independent software development process
- User friendliness

READEX Programming Paradigm

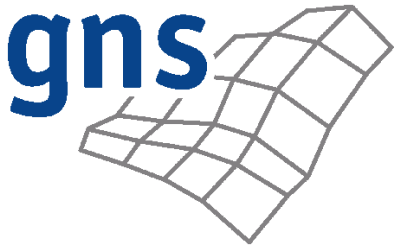
Future work:

- Development of programming paradigm for expressing dynamism
  Goal: further improvement of automatic dynamic tuning

Expected Outcome



Thank you for your attention!



Further information:
`http://www.vi-hps.org/projects/score-e`
`http://www.score-p.org`
`http://www.readex.eu`

Contact: **diethelm@gns-mbh.com**