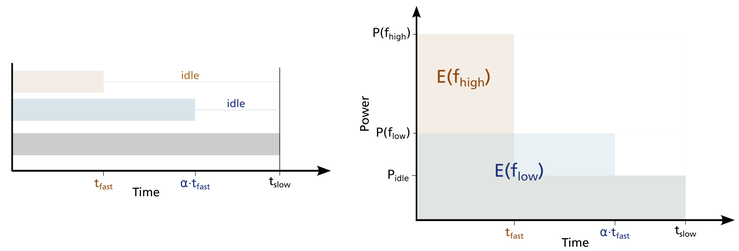
Energy-Efficient Coupled Simulations with Dynamic Voltage and Frequency Scaling

Valentina Schüller[1], Filip Vaverka[2], Ondřej Vysocký[2], Philipp Birken[1], Roman Iakymchuk[3]

[1]Lund University, Lund, Sweden, [2]VSB - Technical University of Ostrava, Ostrava, Czech Republic, [3]Uppsala University, Uppsala, Sweden

1 Idea

Coupled PDE solvers: Codes synchronize at regular intervals (time windows) to exchange data → **Inter-solver load imbalance!**
How much energy can one save by reducing CPU frequency of the fast solver(s) **at runtime**?



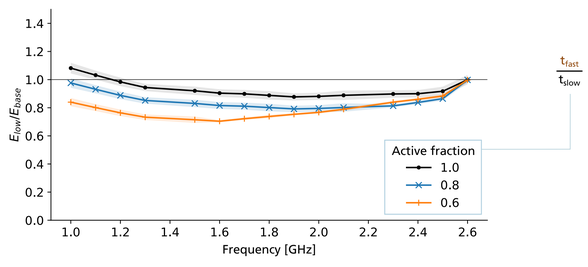
Lower CPU frequency: **Tradeoff** between
1. Time penalty α ↔ arithmetic intensity of code
2. Reduced power consumption: $P_{dyn} \propto V(f)^3$

2 Energy model: Estimate savings potential

$$E_{\text{base}} = P(f_{\text{high}})\, t_{\text{fast}} + P_{\text{idle}}\,(t_{\text{slow}} - t_{\text{fast}})$$
$$E_{\text{low}} = P(f_{\text{low}})\, \alpha t_{\text{fast}} + P_{\text{idle}}\,(t_{\text{slow}} - \alpha t_{\text{fast}})$$

measure for a given solver

Example: Heat equation solver on Barbora (cf. case study)



→ Optimal choice depends on **solver**, **processor**, and **imbalance**.

3 Toolchain: How do you do this in practice?

You need:
- **access to the solvers** you want to adjust
- a tool to **report inter-solver load balance** at runtime
- an **algorithm** to choose new CPU frequencies
- a tool to **adjust** CPU **frequency** at runtime
- a tool to **measure energy consumption**

At the end of each time window:

PDE Solver
Modular toolbox for solving PDEs.
Generates MPI- & OpenMP-parallel C++ code.
High-level Python interface (DUNE-FEM)
UFL-based problem formulation.
`www.dune-project.org`

↓ coupling data exchange

preCICE
Coupling library for multiphysics simulations.
Handles communication & data exchange.
Inter-solver communication via TCP/IP sockets.
Active fraction easy to monitor.
`www.precice.org`

↓ **report** solver active fraction

Jitter [1]
Deterministic algorithm to adjust frequency.
Originally developed for dynamic load balancing of MPI codes, easy to implement & test.
$\frac{t_{fast}}{t_{slow}}$ ↦ New frequency

↓ **choose** new frequency

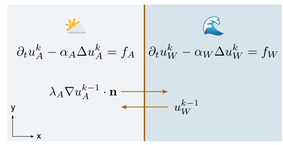
MERIC
Library for energy evaluation & runtime tuning.
Measurements & frequency scaling on various CPU & GPU architectures via a range of interfaces possible.
We use: RAPL counters & msr-safe.
`code.it4i.cz/energy-efficiency/meric-suite`

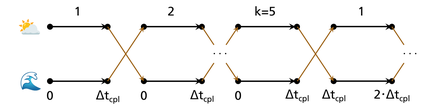
↓ **set** new frequency

\+ **Monitor** energy consumption and inter-solver load balance during simulation with MERIC and preCICE.

4 Case study: 2D Coupled heat equations

`gitlab.maths.lu.se/valentinaschueller/timeadaptiveqnwr`





Heat equations for air & water: t ∈ [0, 30], (x,y) ∈ [0,2] × [0,1].
Partitioned approach, iterative coupling [2]: Δt_{cpl} =10.
Linear finite elements in space, implicit Euler in time.
Linear solver: GMRES with SOR preconditioning.

Δx = Δy	1/500	1/600
Δt	1/10	1/10

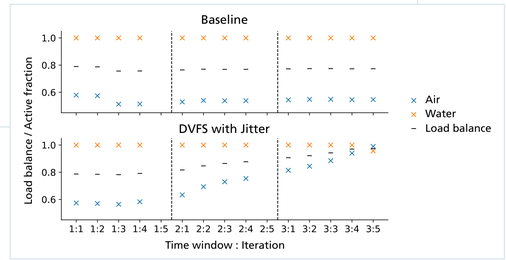
It can work:

Barbora	Time to solution [s]	⌀ Load Balance	Energy to solution [kJ] (air solver)	Energy to solution [kJ] (total)
Baseline (2.6GHz)	180 ± 1	0.82	37.0 ± 0.2	77.6 ± 0.3
DVFS with Jitter	**181** ± 2	**0.87**	**26.6** ± 0.2	**67.4** ± 0.5
	↗ <1%	↗ 6%	↘ 28%	↘ 13%

Intel Cascade Lake. Per solver: 1 node, 18 MPI procs, 2 OpenMP threads.

Load balance ↗ Energy to solution ↘ without compromising time to solution.

Faster air solver is slowed down ✔



But not always:

Karolina	Time to solution [s]	⌀ Load Balance	Energy to solution [kJ] (air solver)	Energy to solution [kJ] (total)
Baseline (3.3GHz)	386 ± 5	0.92	110 ± 2	223 ± 4
DVFS with Jitter	**468** ± 1	**0.89**	**138** ± 0	**279** ± 1
	↗ 18%	↘ 3%	↗ 25%	↗ 25%

AMD Zen2. Per solver: 1 node, 64 MPI procs, 2 OpenMP threads.

Jitter overshoots, imbalance switches, DVFS creates overhead!

Selected References.
[1] V. W. Freeh et al. (2008): "Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs," J. Parallel Distrib. Comput.
[2] Rüth et al. (2021): "Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications," Int J. Numer. Methods Eng.

valentina.schuller@math.lu.se · valentinaschueller.github.io

Acknowledgements. This work was partially supported by the Swedish e-Science (eSSENCE) research program. The compute resources were provided by the EuroHPC Development Access grant, EHPC-DEV-2025D02-126. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254).