High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

NETCOM

Optimizing the Energy Consumption of Computation

Tobias Fischbach
University of Luxembourg

Pascal Bouvry
University of Luxembourg

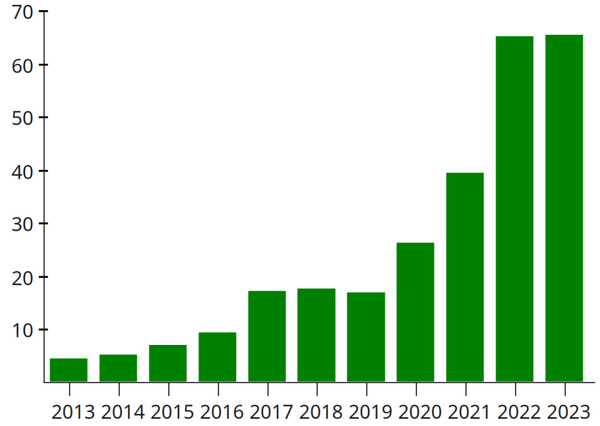
UNIVERSITÉ DU
LUXEMBOURG

Introduction

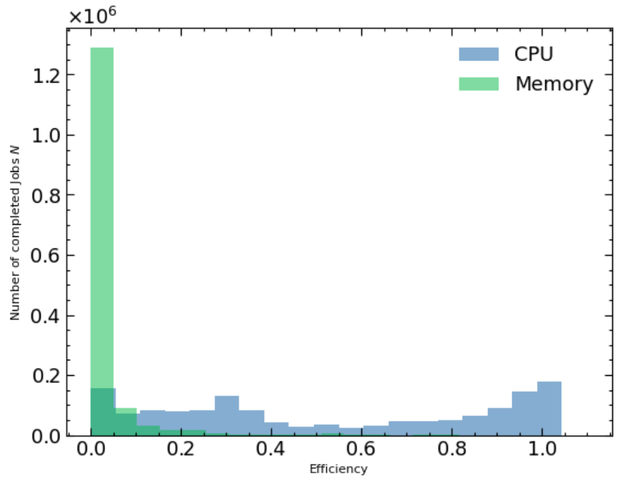
Motivation

Context

- ICT sector responsible for \(\approx 3-4\%\) of world wide power consumption[1]
- \(\approx 20\%\) due to data centers
- continues research to improve energy efficiency of hardware (eg. Green500)[2]
- concernes are raised regarding the power demand of HPC-code in the era exascale computing
- focus on accelerators and mpi applications [3][4]
- energy-aware scheduling successfull in retrospects or tight bounds [5][6]
- Software optimization is another tool to improve energy efficiency
- Average memory efficiency: \(2.675\%\)
- Average cpu efficiency: \(52.460\%\)



\(\left[\frac{\text{GFlops}}{\text{W}}\right]\) of the Green500 winners per year[2].



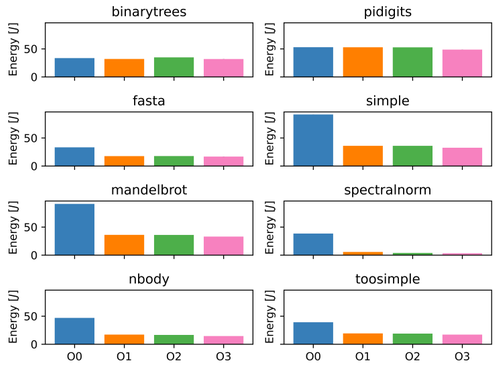
CPU and memory efficiency of all completed jobs on Iris (Feb. 2018 - Sep. 2023).

The case for Software Optimization

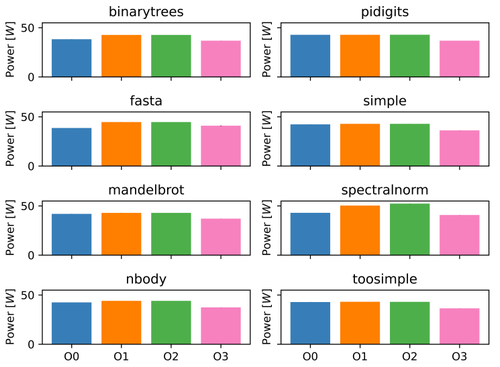
State of the Art

- faster software not necessarily more energy efficient
- cache misses and branching seem to be dominant drivers
- less optimized code can consume less power and / or less energy
- different runtime characteristics \(\rightarrow\) software categories

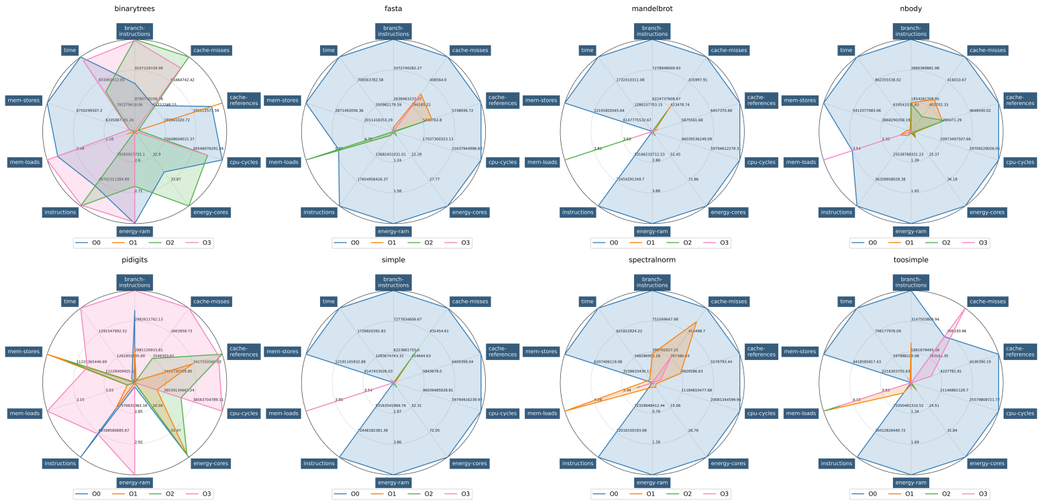
\(\Rightarrow\) What is the optimization pipeline that that minimizes the total energy consumption?



Total energy consumption of Clang for different opt. levels.



Average power consumption of Clang for different opt. levels.



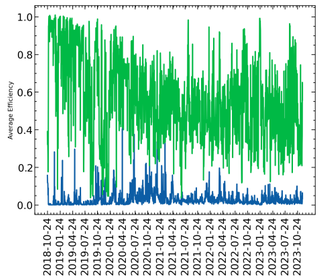
Runtime characteristics of Clang based opt. levels.

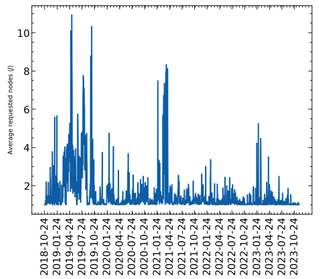
UNIVERSITÉ DU LUXEMBOURG

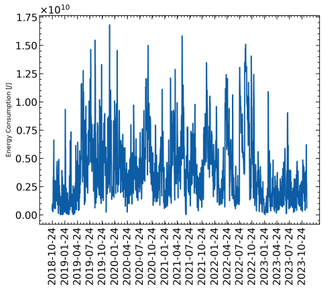
Roadmap - PhD Sandwich

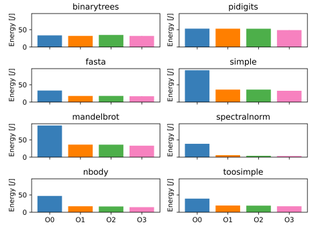
1. Top View: HPC Job Analysis

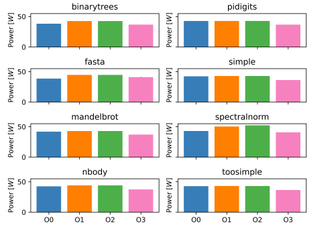
- environmental impact of HPC scheduling (PhD course project)
- clustering (done, but not promising)
- identify inefficient jobs upon submission











2. Middle View: Surrogate Software Optimization

- generate real-word and reproducible model
- provide an energy efficient pass pipeline for different software categories
- write own LLVM-Pass
- compare different CPU architecture

3. Bottom View:

- generate explanations
- guidelines for green code
- CPU architecture comparison

UNIVERSITÉ DU LUXEMBOURG

Methodology

Power measurements

Software

- IPMI
- Intel RAPL (Linux, x86):
  - Likwid
  - PowerTOP
  - PowerStat
  - perf
- Syspower (macOS)

Hardware

- Wattmeter
- PDUs
- Curent sensors

Challenges

- ARM and RISC-V require vendor support or extra hardware to monitor the power consumption
- Intel RAPL only supported by x86 and educated guessing

Intel RAPL

Usage

- access counters with perf:

```
/proc/sys/kernel/perf_event_paranoid < -1
perf --event="power/energy-pkg/,power/energy-ram/" \
     -j ./hal-9000.sh
```

- alternative direct access at “/dev/cpu/??/msr”
- available counters[7]:
  - energy-pkg := complete CPU socket
  - energy-cores := all CPU cores
  - energy-ram := DRAM
  - energy-psys := eDRAM and I/0 Handling
- \(f_{\text{update}} = 1 \text{ , } \left[\text{kHz} \right]\)

Limitations

- no on-die circuits since Haswell-E \(\rightarrow\) application of CPU-specific offsets to historic data
- power management dynamics not properly captured
- package based readings (core based readings in new AMD Power Management driver)
- systemwide readings \(\rightarrow\) measurements include os and other running software (**measurement script**)
- scheduling effects
- thermal effects
- discrepancy in:
  - low DRAM usage[8]
  - dynamic CPU usage[9]
  - short Applications[9]

Reducing the noise

System changes

- power management \(\rightarrow\) force c/p-state
- CPU scheduler \(\rightarrow\) exclude all cores but one
- thermal effects \(\rightarrow\) limit CPU frequency
- measurement script effects \(\rightarrow\) limit perf to \(f_{\text{update}}=100 \text{ , } \left[\text{Hz} \right]\)

\(\Rightarrow\) **Reproducible measurements (within \(<5\%\)) but lack of real-world connection**

Reproducible vs Real-World

- reproducible configuration for:
  - CPU Architecture comparisons
  - surrogate model used for explanation generation
- real-world configuration:
  - software optimization using a surrogate model

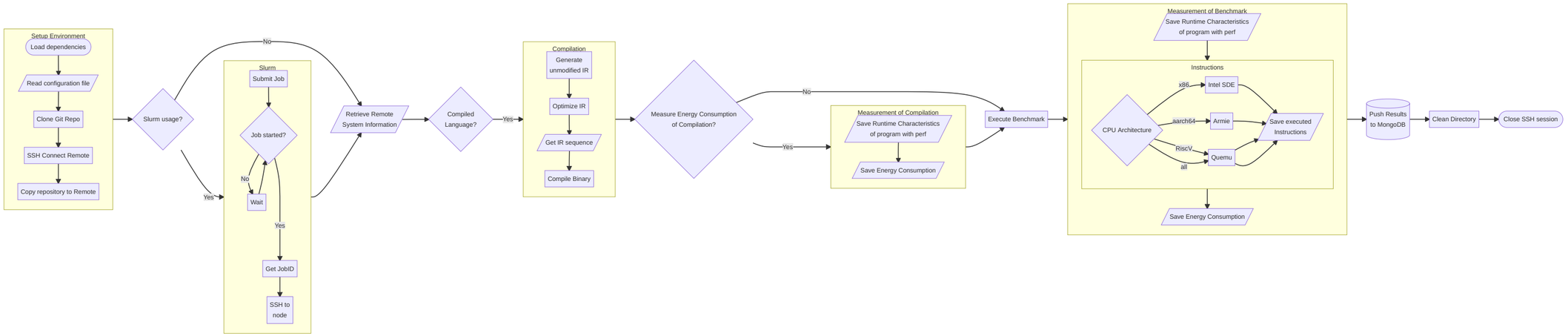
Example Configuration

```
boot = {
    kernelParams = [
      "processor.max_cstate=1"
      "intel_idle.max_cstate=0"
      "intel_pstate=passive"
      "intel_pstate=no_hwp"
      "intel_pstate.no_turbo=1"
      "isolcpus=1,2,3,4,5,6,7,8,9,10,11"
      "mitigations=off"
    ];
    kernel.sysctl."kernel.perf_event_paranoid" = -1;
    kernel.sysctl."kernel.kptr_restrict" = 0;
};
systemd.extraConfig = ''
    CPUAffinity=0
'';
systemd.services.set-cpu-freq = {
    wantedBy = [ "multi-user.target" ];
```

Example Task Spawner

```
perf stat -j --event="power/energy-cores/" \
    taskset -c 1-11 <command>
# or
perf stat -j --event="power/energy-cores/" \
    numactl -physcpubind=1-11 <command>
```

Experimental Pipeline - Overview



- reduce measurement overhead by separating experiment and data processing
- nix is used to create a reproducible software environment
- mongoDB for data management
- extract executed instructions Qemu, Intel SDE and Armie
- saves LLVM-IR by default
- modular (replace measurement method, integrate with optimization pipeline, …)

UNIVERSITÉ DU LUXEMBOURG

Experimental Pipeline - Usage

Usage

1. Define building, jumping (opt.) and destination machine
2. Setup language specifics, including custom compiler passes
3. Specify how environments should be started (here nix)
4. mongoDB database and column setup
5. Define how programs should be called and how to measure

Limitations

- Retrieval of executed Instructions through emulation slow
- Capturing runtime Characteristics with perf:
  - introduces overhead
  - depends on the argument order
- Relying on a running SSH service (extra overhead)

Example Configuration

```
{
    "dic_dest": {
        "server": "<destination-adress>",
        "username": "<username>",
        "password": "<password-or-key>",
        "port": <port>
    },
    "dic_jump": {
        <jumphost-config>
    },
    "dic_local": {
        <localhost-config>
    },
    "remote_dir": "<remote-dir>",
    "language": "<language>",
    "opt": "<custom-pass-pipeline>",
    "env_loader_build": "<shell-script-or-command-to-load-build-env
    "env_loader_exp": "nix-shell -Q <path-to-experiment-env>/shell.
```

Example Call

```
$ python driver.py my-python-experiment.conf
[2023-11-11 17:54:22.678456]:   Setup /home/tobias/src/python/bin/r
[2023-11-11 17:54:23.007983]:   Enqueue runs.
[2023-11-11 17:54:23.008093]:   Execute experiment.
[elapsed time: 0:02:13]:|****************************************
[2023-11-11 17:56:36.868175]:    Setup experiment for language pyth
[2023-11-11 17:56:38.224519]:   Start remote build.
[2023-11-11 17:57:12.296006]:   Copy /tmp/build to /home/tobias on
```

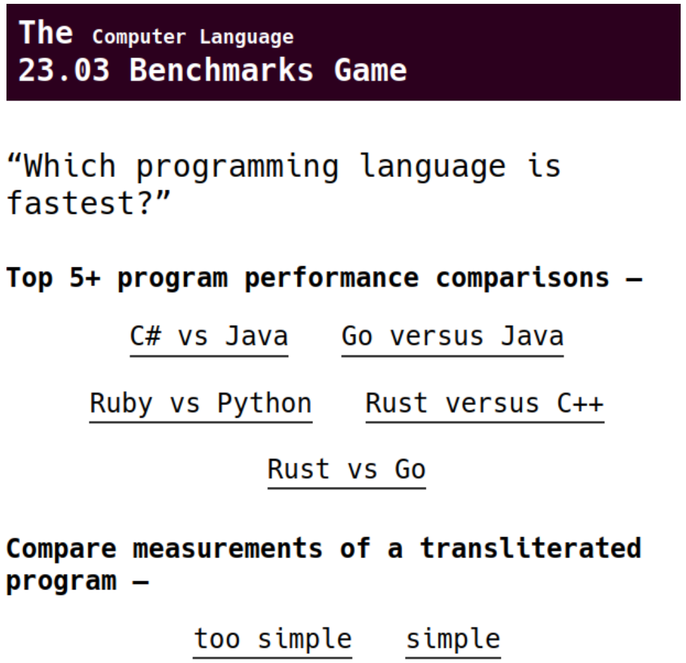
How can we trust our measurements? \(\Rightarrow\) Validation Experiment

Programming Language Benchmark

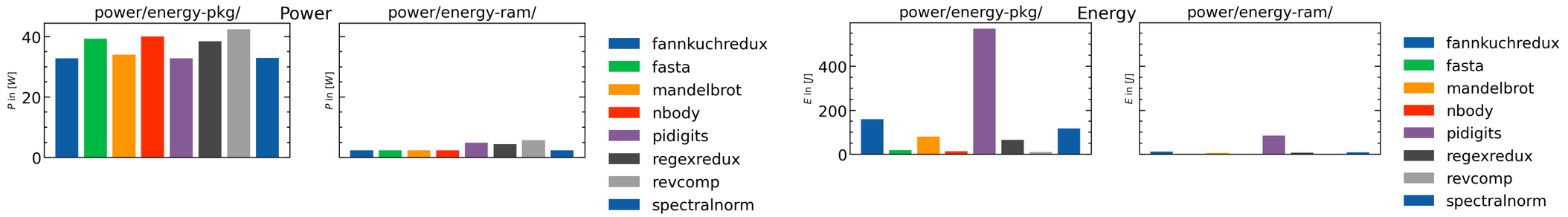
Game Rules

- no external libraries (eg. no numpy)
- code from the computer language benchmarks game[10]
- same input parameters
- consider consumed power by compilation and subtract baseline
- $$\begin{aligned} E^{\text{bench}} = & \left< E^{\text{bench}}_{i}\right> + \left< E^{\text{comp}}_{i}\right> - 2*\left< E^{\text{base}}_{i}\right> \text{ , }\left[\text{J}\right]\\ P^{\text{bench}} = & \frac{\left< E^{\text{bench}}_{i}\right> + \left< E^{\text{comp}}_{i}\right> - 2*\left< E^{\text{base}}_{i}\right>}{\left< t_{i}^{\text{bench}}\right>} \text{ , }\left[\frac{\text{J}}{\text{s}}\right]\\ \end{aligned}$$

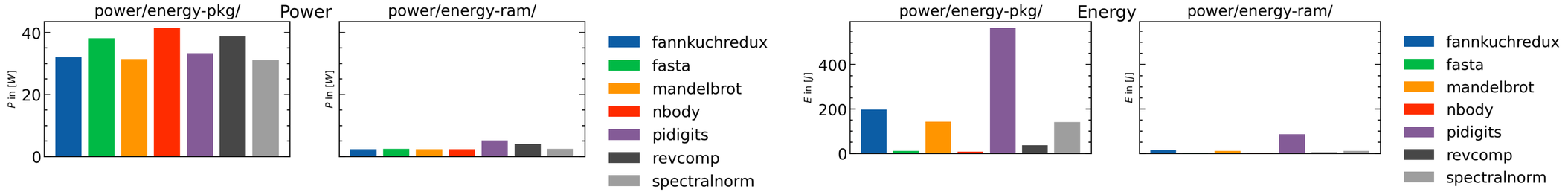
$\Rightarrow$ reproduces[11] and[12] $\rightarrow$ **experimental pipeline is sound**



C/C++

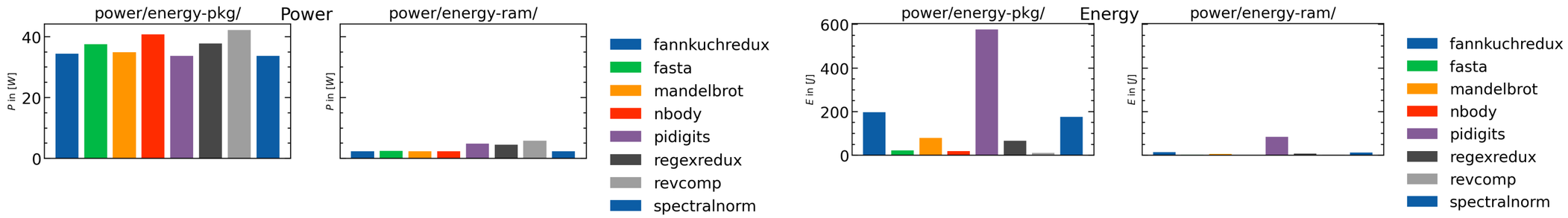


Clang 16

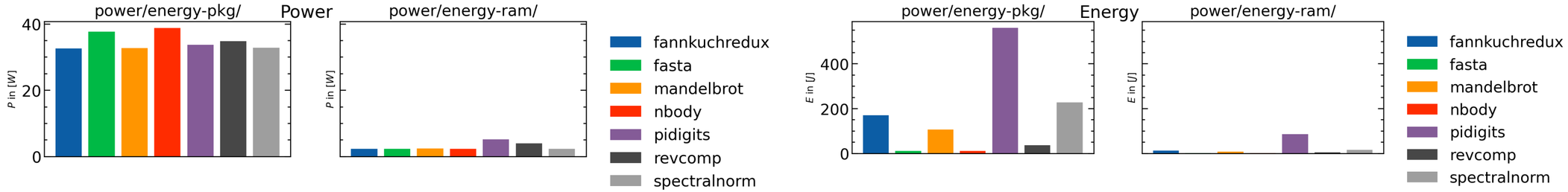


Clang++ 16

C/C++

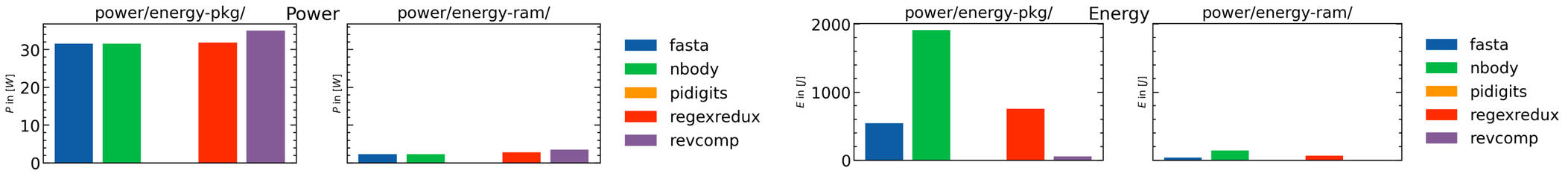

power/energy-pkg/
Power
power/energy-ram/
P in [W]
0
20
40
power/energy-pkg/
Energy
power/energy-ram/
E in [J]
0
200
400
600
fannkuchredux
fasta
mandelbrot
nbody
pidigits
regexredux
revcomp
spectralnorm

GCC 12

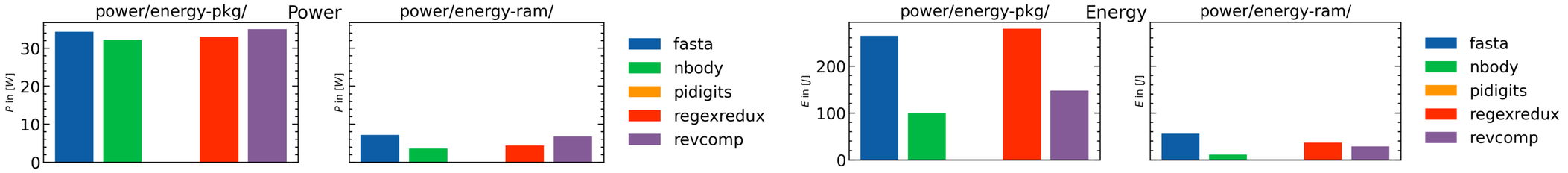

power/energy-pkg/
Power
power/energy-ram/
P in [W]
0
20
40
power/energy-pkg/
Energy
power/energy-ram/
E in [J]
0
200
400
fannkuchredux
fasta
mandelbrot
nbody
pidigits
revcomp
spectralnorm

G++ 12

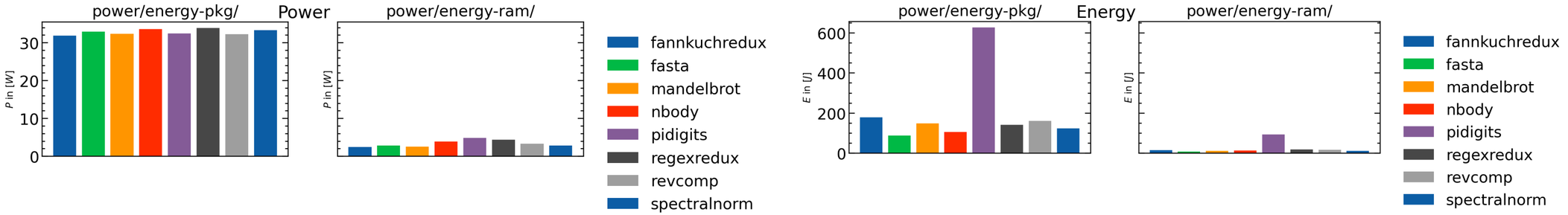
Interpreted Languages



Python 3.10



PyPy 7.3.8



Julia 1.9

UNIVERSITÉ DU LUXEMBOURG

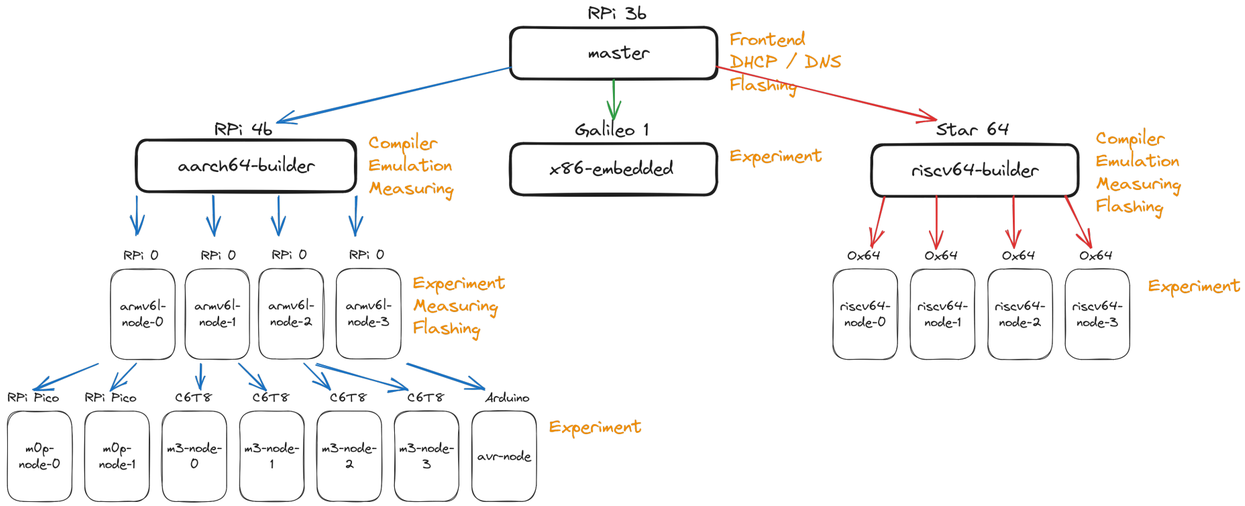
Programming Language Benchmark: Whats next?

Computer Architecture

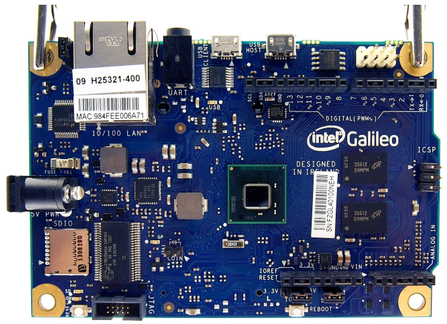
- compare energy consumption for different cpu architecturess
- currently missing:
  - Slurm setup on master
  - small code to interact with current sensor via GPIO

Parallel Computing

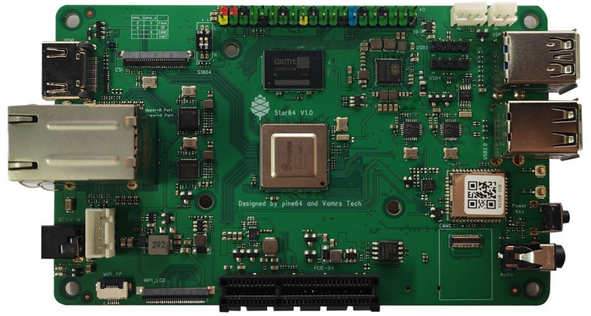
- compare the energy efficiency scaling from single-core to multi-core to multi-nodes
- using matrix-multiplication
- currently implementing



Tree Diagram of a cluster that is used for several experiments. Different x86, ARM, RISC-V and microcontrollers are supported. The energy consumption is measured with external current sensors and is always system-wide.



Galileo is the x86 X1000 32-bit “microcontroller” from Intel (no SIMD like MMX or SSE). Comparable to i586.



Star64 is a RISC-V board based on JH7110. No vector extension support. Qemu is supports this sock.

Software Optimization using Surrogate models

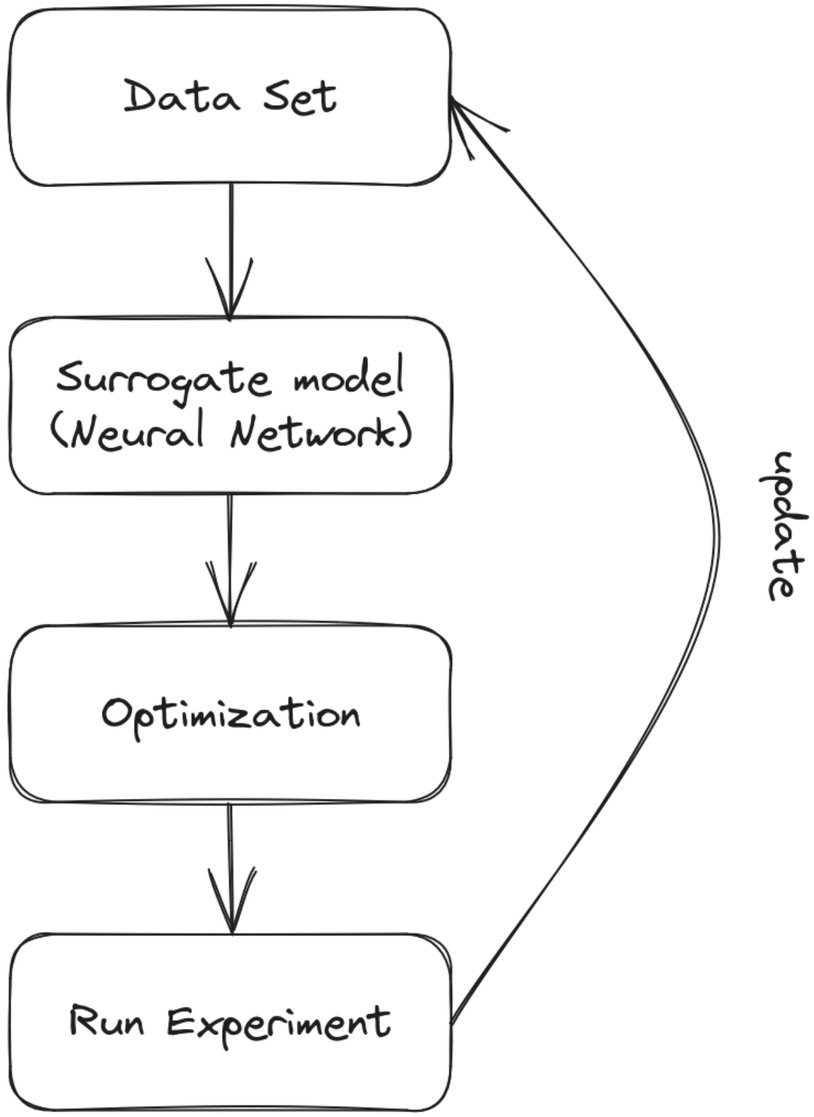
Goal

- find the transformation pass pipeline that minimizes the energy consumption for different software types
- target IR instead of individual programming languages
- Approach detailed in: “Challenges in Automatic Software Optimization: the Energy Efficiency Case”, Fischbach, Kieffer, Bouvry (2023)

Why Surrogate Optimization?

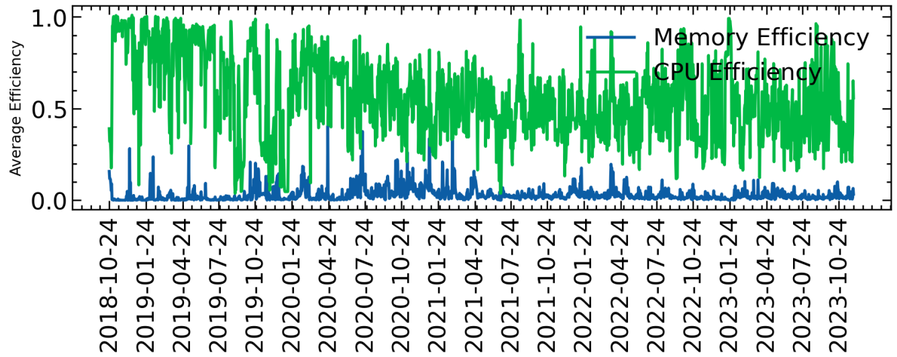
- time consuming evaluation
- blackbox (neural network based)
- model can be used to generate explanations

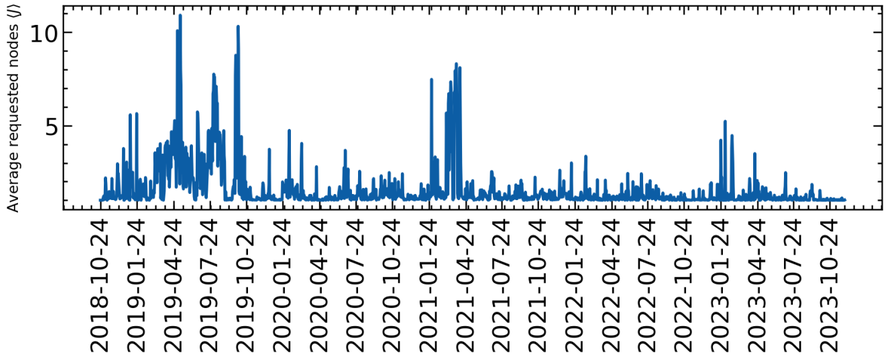
\(\Rightarrow\) Next step is the integration with the experimental pipeline.

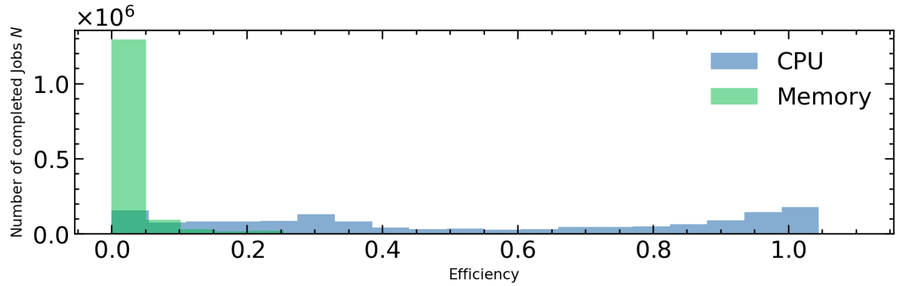


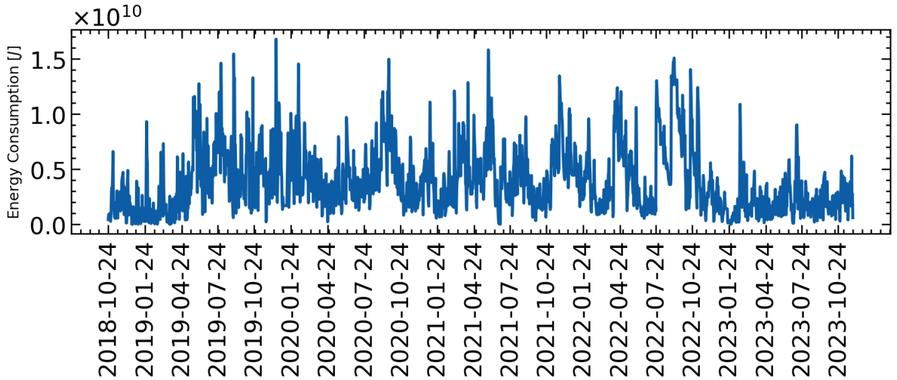
Job Efficiency

Iris









- CPU efficiency: \(C_{\text{eff}}=\frac{t_{\text{TotalCPU}}}{t_{\text{Elapsed}}*N_{\text{requested}}}\)
- Memory Efficiency \(M_{\text{eff}}=\frac{\text{MaxRSS}}{\text{Mem}_{\text{requested}}*N_{\text{requested}}}\)

Outlook

Summary

- HPC users are better in utilizing CPUs than memory
- most of the requested memory is unused
- modular framework to capture runtime dynamics and energy consumption
- introduction of programming language benchmark
- next surrogate optimization of LLVM pass-pipeline

Thank you for your attention



Bibliography

1. The Beating Heart of the World's First Exascale Supercomputer - IEEE Spectrum.
2. Green500 TOP500.
3. Rountree, B. *et al.* Bounding energy consumption in large-scale MPI programs. in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing* 1–9 (ACM, 2007). doi:10.1145/1362622.1362688.
4. Mittal, S. & Vetter, J. S. A Survey of Methods for Analyzing and Improving GPU Energy Efficiency. *ACM Computing Surveys* **47**, 19:1–19:23 (2014).
5. Garg, S. K., Yeo, C. S., Anandasivam, A. & Buyya, R. Energy-Efficient Scheduling of HPC Applications in Cloud Computing Environments. *arXiv.org* (2009).
6. Kocot, B., Czarnul, P. & Proficz, J. Energy-Aware Scheduling for High-Performance Computing Systems: A Survey. *Energies* **16**, 890 (2023).
7. Intel® 64 and IA-32 Architectures Software Developer Manuals. *Intel*.
8. Desrochers, S., Paradis, C. & Weaver, V. M. A Validation of DRAM RAPL Power Measurements. in *Proceedings of the Second International Symposium on Memory Systems* 455–470 (Association for Computing Machinery, 2016). doi:10.1145/2989081.2989088.
9. Zhang, H. & Hoffmann, H. A Quantitative Evaluation of the RAPL Power Control System.
10. Which programming language is fastest? (Benchmarks Game).
11. Pereira, R. *et al.* Energy efficiency across programming languages: How do energy, time, and memory relate? in *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering* 256–267 (Association for Computing Machinery, 2017). doi:10.1145/3136014.3136031.
12. Pereira, R. *et al.* Ranking programming languages by energy efficiency. *Science of Computer Programming* **205**, 102609 (2021).