

# Energy-Aware Computing Systems

*Energiebewusste Rechensysteme*

## IX. Energy-Aware Programming

Timo Hönig

July 16, 2020



# Agenda

---

Preface and Terminology

System Activities and Energy Demand

Cross-Layer Considerations

Retrospective vs. Prospective

Energy-Aware Programming

HEAL, ROAM

Paper Discussion

Summary



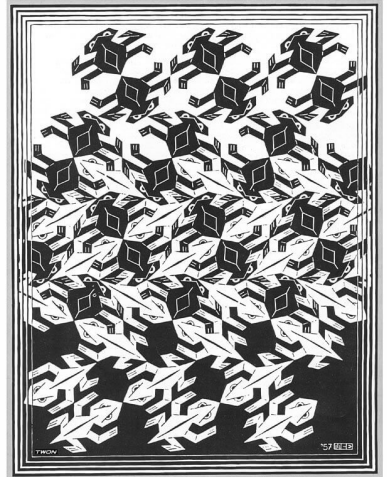
# Energy-Aware Programming

## ■ motivation

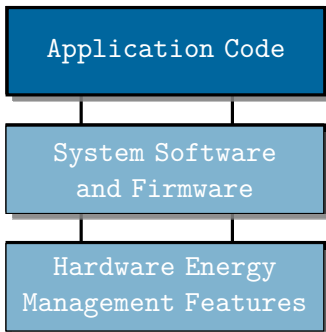
- knowledge transfer:  
development → execution phase
- reduction of work to  
the necessary minimum
- carry out the remaining work in the  
most efficient way

## ■ operational goals

- reduce guesswork by lower system  
levels (i.e., system software,  
firmware, and hardware)
- interweave static aspects (→ ahead  
of run time) with dynamic aspects  
(→ at run time)



# Cross-Layer Considerations



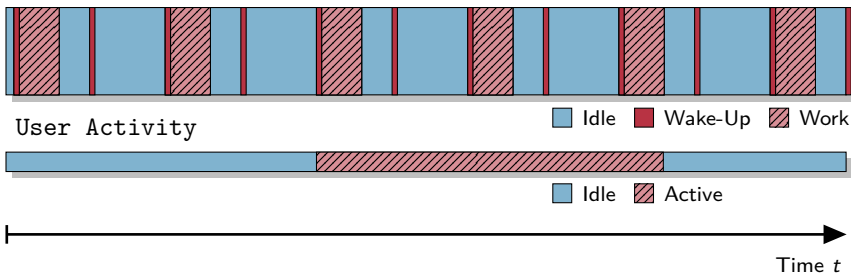
- compiler optimization (e.g., loop optimizations, aligned RAM access)
- tracing and profiling tools (e.g., PowerTOP)
- energy management stack
- latency hiding, race/crawl to sleep
- dynamic voltage and frequency scaling (DVFS)
- sleep states (e.g., CPU C-states, device-specific power saving features)



# Retrospective vs. Prospective: Analysis

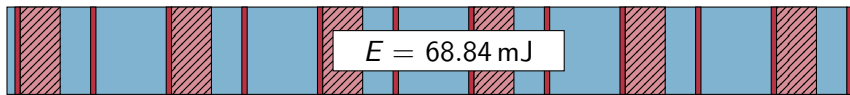
- statistics at process level (e.g., PowerTOP), unit of measurement is *wake-ups per second*
- wake-ups cause the CPU to return from C-state, subsequent activities (e.g., I/O) are likely to follow
- less wake-ups  $\rightarrow$  lower energy demand

## Process Activity

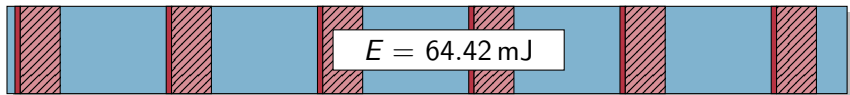


# Retrospective vs. Prospective: Revisions and Impact

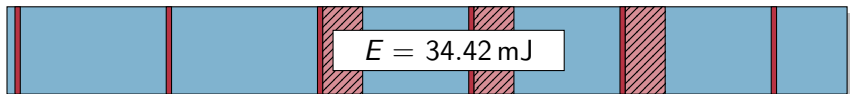
Process Activity 1



Process Activity 1' (1 + adjusted periodic wakeups)



Process Activity 1'' (1' + exclude idle times)

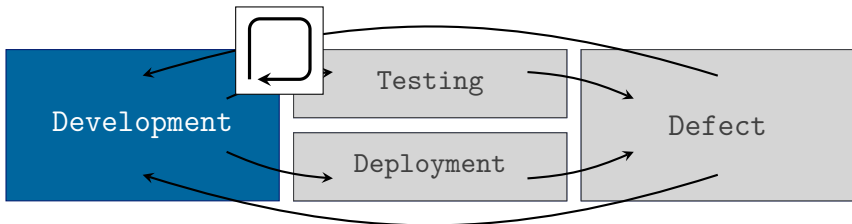
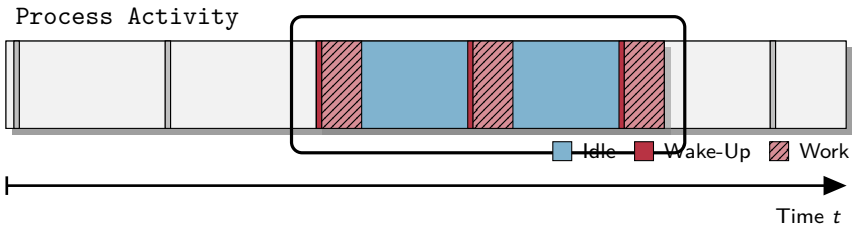


Idle Wake-Up Work

Time  $t$

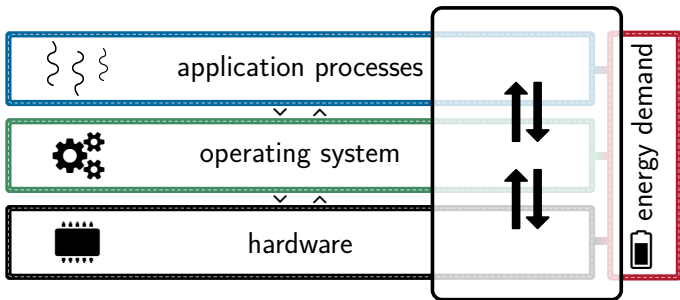


# Retrospective vs. Prospective: Forward-Looking

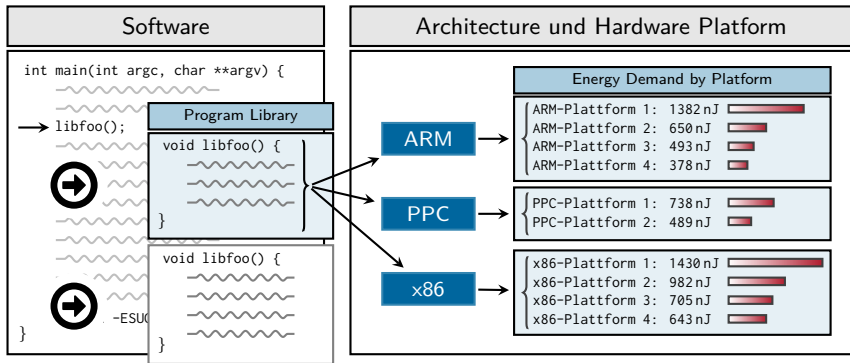


# Energy-Aware Programming

- proactive energy-aware computing
  - cross-layer und cross-phase (positioning and momentum)
  - focus: single-chip computing systems and HPC
- holistic analysis and evaluation of software components with regard to their impact on the energy demand of the systems



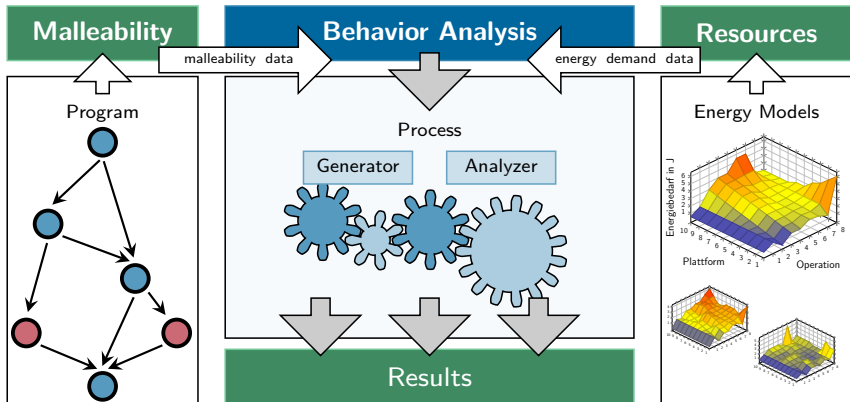
# HEAL: Energy-Aware Programming



- making energy demand estimates at the function level available during development
- basis for energy-aware programming decisions



# HEAL: Architecture and Implementation



- determine malleability by program analysis
- behavioral analysis with process execution and evaluation
- resource-demand analysis using energy models



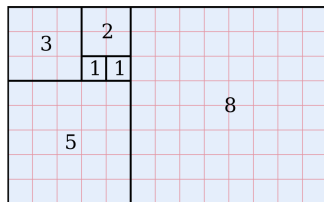
# HEAL: Program Example Fibonacci Sequence

## Program:

```
1 # modes: (l)ookup, static
2 #         (c)alculate, dynamic
3 #         (m)emoisation, dynamic
4
5 def main():
6     mode = sys.argv[1]
7     fnum = 42
8
9     if mode == 'l':
10        fib_lookup(fnum) Ⓞ
11    elif mode == 'c':
12        fib_calc(fnum) Ⓞ
13    elif mode == 'm':
14        fib_calc_mem(fnum) Ⓞ
15
16 if __name__ == "__main__":
17    main()
```

## HEAL:

1. path exploration (argv[1]: symbolic)
2. generate program with concrete input  
e.g., argv[1]: 'c', fnum: 42
3. program execution and evaluation  
→ energy demand estimate



# HEAL: Results and Open Questions

Malleability

Behavior Analysis

Resources

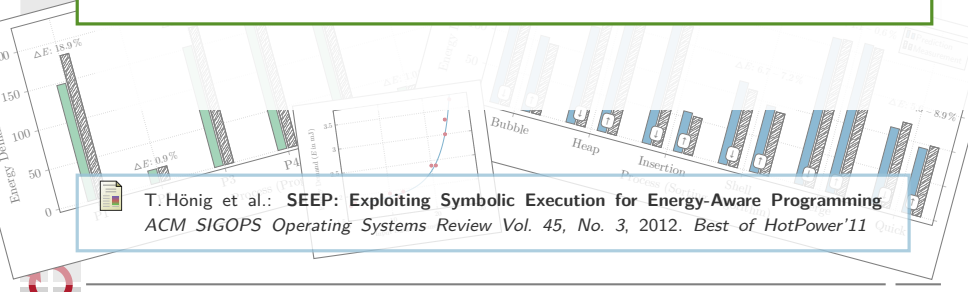
- i** energy demand estimates deviate on average by less than 9.1% compared to energy measurements
- i** the evaluation shows that the energy demand of functionally identical processes deviate up to 3.9 times



T. Hönig et al.: **SEEP: Exploiting Symbolic Execution for Energy-Aware Programming**  
*ACM SIGOPS Operating Systems Review* Vol. 45, No. 3, 2012. *Best of HotPower'11*

# HEAL: Results and Open Questions

- ✓ comparison of (functionally identical) programs as to their different non-functional properties
- ✓ energy-demand analysis tightly integrated with the development process of software



T. Hönig et al.: **SEEP: Exploiting Symbolic Execution for Energy-Aware Programming**  
*ACM SIGOPS Operating Systems Review* Vol. 45, No. 3, 2012. *Best of HotPower'11*

# HEAL: Results and Open Questions

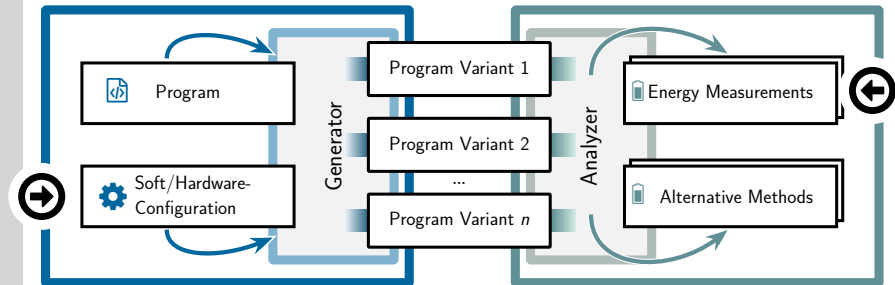


- ✗ missing and inaccurate energy models for hardware components are the rule
- ✗ unused potential to further reduce energy demand by pre-analysis of runtime energy-saving mechanisms



T. Hönig et al.: **SEEP: Exploiting Symbolic Execution for Energy-Aware Programming**  
*ACM SIGOPS Operating Systems Review* Vol. 45, No. 3, 2012. *Best of HotPower'11*

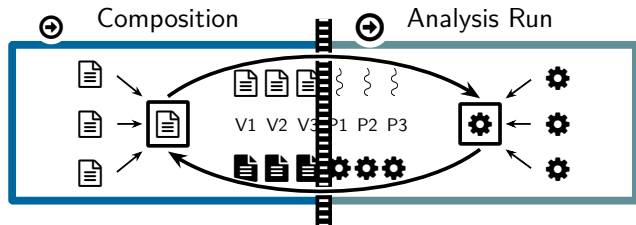
# ROAM: Program Variant Generator and Analysis



- generate program variants: programs with different software/hardware configurations
- energy measurements with a measuring circuit which is based on a current mirror for determining the energy demand



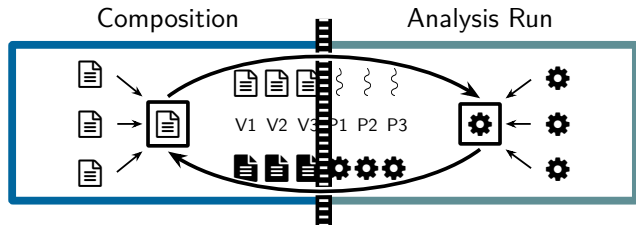
# ROAM: Architecture and Implementation



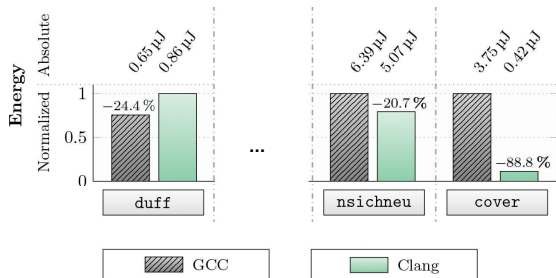
- composition: static preparation for testing
  - heterogeneous hardware settings (e.g., energy saving features)
  - different software settings (e.g., compiler settings)
- analysis run: dynamic evaluation
  - execution of program variants on different hardware platforms
  - determination of execution time and energy demand by measurement



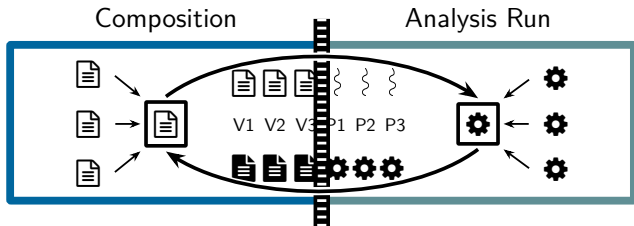
# ROAM: Experiments and Results



First experiment<sup>1</sup>: comparison of interface-compatible compilers



# ROAM: Experiments and Results

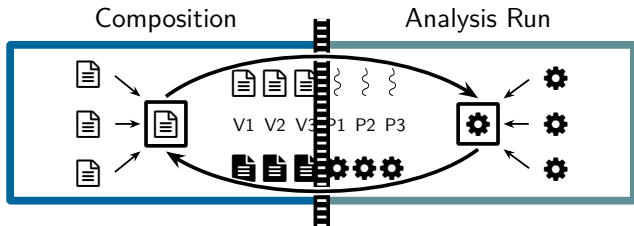


First experiment<sup>1</sup>: comparison of interface-compatible compilers

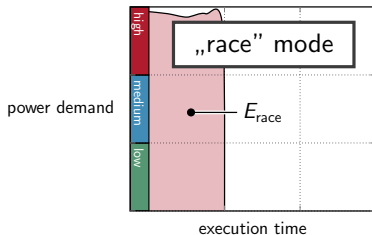
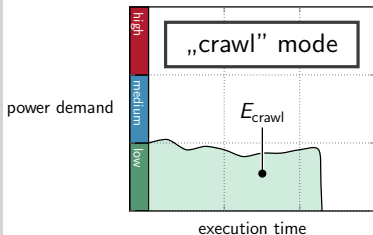
- GCC vs. Clang**
- in 80 % of the cases, GCC generates more energy-efficient program variants (up to a quarter lower energy demand)
  - one program variant of Clang is approx. 10x more energy-efficient than the corresponding variant of GCC
- energy vs. time**
- no causal relationship between process energy demand and execution time in 10 % of the program analyses

<sup>1</sup>Software: GNU GCC 4.8, LLVM Clang 3.4, Hardware: ARM Cortex-M0+ (Kinetis KL02)

# ROAM: Experiments and Results

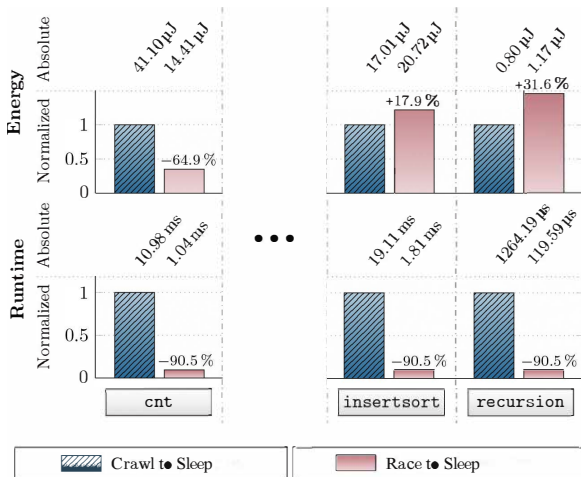


Second experiment<sup>2</sup>: scaling of operating voltage and clock frequency

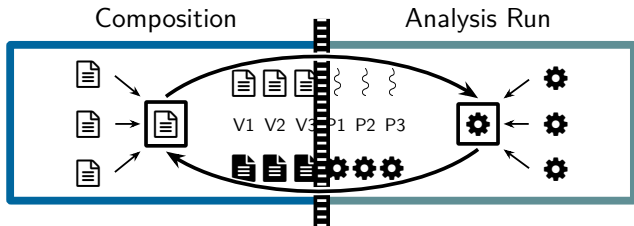


<sup>2</sup>Software: GNU GCC 4.8, Hardware: ARM Cortex-M0+ (Kinetis KL02, RUN/VLPR)

# ROAM: Experiments and Results



# ROAM: Experiments and Results



Second experiment<sup>2</sup>: scaling of operating voltage and clock frequency

- race vs. crawl** ■ „race” mode is commonly preferred to maximize idle time (→ exploit sleep modes)
- expected increase in performance occurs in all test cases (i.e., shortening of the execution time)

**energy vs. time** ■ however, no causal relationship between process energy demand and execution time in 20% of the program analyses

<sup>2</sup>Software: GNU GCC 4.8, Hardware: ARM Cortex-M0+ (Kinetis KL02, RUN/VLPR)

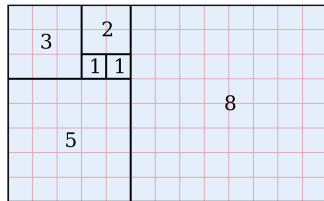
# ROAM: Program Example Fibonacci Sequence (II)

Program:

```
1 # modes: (l)ookup, static
2 #         (c)alculate, dynamic
3 #         (m)emoisation, dynamic
4
5 def main():
6     hwop = roam_fetch_hwops()
7     mode = sys.argv[1]
8     fnum = 42
9
10    sw_hardware_mode(hwop);
11    if mode == 'l':
12        fib_lookup(fnum)
13    elif mode == 'c': ⚡ race/crawl
14        fib_calc(fnum)
15    elif mode == 'm': ⚡ race
16        fib_calc_mem(fnum)
17    reset_hardware_mode(); ⚡ crawl
18
19    if __name__ == "__main__":
20        main()
```

ROAM:

1. generate software and hardware settings to be used
2. generate program variants
3. process execution and evaluation
  - energy demand measurements
  - results evaluation



# ROAM: Results

Composition

Analysis Run

- i** configuration settings made by ROAM reduce hardware energy demand by between 18 % and 65 %
- i** choosing the right compiler infrastructure can reduce the energy demand by a factor of 10



T. Hönig et al.: Proactive Energy-Aware Programming with PEEK  
USENIX TRIOS '14

# ROAM: Results

Composition

Analysis Run

- ✓ pre-analysis generates necessary a priori knowledge for suitable hardware settings at process execution time
- ✓ energy measurement during analysis addresses unavailability of energy models



T. Hönig et al.: Proactive Energy-Aware Programming with PEEK  
USENIX TRIOS '14

# Agenda

---

Preface and Terminology

System Activities and Energy Demand

Cross-Layer Considerations

Retrospective vs. Prospective

Energy-Aware Programming

HEAL, ROAM

Paper Discussion

Summary



- paper discussion

- ▶ R. Pereira et al.

- Energy efficiency across programming languages: how do energy, time, and memory relate?**

- Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE'17), 2017.*



- energy-aware programming connects **static** (ahead of run time) with **dynamic** (at run time) analysis
- use **cross-layer considerations** to reduce energy demand
- pinpoint relevant program code sections for **extended analysis** and manual labor
- reading list for Lecture 10:
  - ▶ X. Fan et al.  
**Power provisioning for a warehouse-sized computer**  
*Proceedings of the 34th International Symposium on Computer architecture (ISCA'07), 2007.*



- [1] HÖNIG, T. ; EIBEL, C. ; KAPITZA, R. ; SCHRÖDER-PREIKSCHAT, W. :  
SEEP: exploiting symbolic execution for energy-aware programming.  
In: *Proceedings of the 2011 Workshop on Power-Aware Computing and Systems (HotPower '11)* ACM, 2011, S. 17–22. –  
Best of HotPower 2011 Award.
  
- [2] HÖNIG, T. ; JANKER, H. ; EIBEL, C. ; MIHELIC, O. ; KAPITZA, R. ;  
SCHRÖDER-PREIKSCHAT, W. :  
Proactive Energy-Aware Programming with PEEK.  
In: *Proceedings of the 2014 Conference on Timely Results in Operating Systems (TRIOS '14)* USENIX, 2014, S. 1–14

