

Energy-Aware Computing Systems

Energiebewusste Rechensysteme

V. Components and Subsystems

Timo Hönig

2019-05-23



EASY

Preface: The Parts vs. The Whole

- „The Whole is Greater Than The Sum of Its Parts” (Aristoteles)
 - synergy → working together
 - the purpose of individual *parts* (components) may be unrelated to the achieved *whole* (overall system)
- necessary preliminary work
 - construction of systems requires meaningful assembly of the individual parts
 - ...the sum of *parts* does not become a *greater whole* by accident...



Agenda

Preface

Terminology

Operating Domains

Scopes and Frontiers

Monitoring and Control

Components and Subsystems

Energy-Aware Processing Strategies

Data Processing and Computing (CPU)

Volatile Data (Uncore, Memory)

Summary



©thoenig EASY (ST 2018, Lecture 5) Preface

3 – 29

Abstract Concept: Components and Subsystems

■ **components** and subsystems

- component: constituent part or element
- **hardware** components
 - implementation of basic system functions
 - functional interactions between components
 - implement subsystems...



skoda-storyboard.com

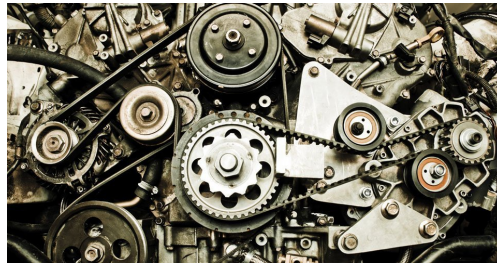


©thoenig EASY (ST 2018, Lecture 5) Terminology

6 – 29

■ components and **subsystems**

- overall systems are composed of subsystem
- **software** subsystems
 - hardware drivers and interaction → logic
 - local operation with a global scope
- **duty** and **high art** of computing
 - drive functionalities of hardware components
 - correct
 - efficient (i.e., performance characteristics)
 - with minimal effort (i.e., low energy demand)



multiscreensite.com

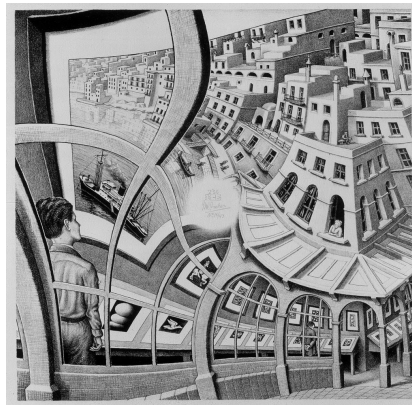
- considerations with regards to the impact and scope
- local and global **scope**
 - fast path to deep sleep state (i.e., without query towards higher level abstractions)
 - may (unnecessarily) stall other components when functionality is needed (e.g., ramp-up delay)
- time **frontier**
 - consider reordering of actions → keep quality of service (e.g., performance) but reduce energy demand?
 - runtime reordering (dynamic), programming reordering (static)



thestranger.com

Monitoring and Control

- higher level **monitoring**
 - software tracks (global) system state
 - operation states of components (i.e., active, idle, standby, sleep)
- diversified **control**
 - components have varying characteristics → different control mechanisms
 - subsystems that operate components are heterogeneous...

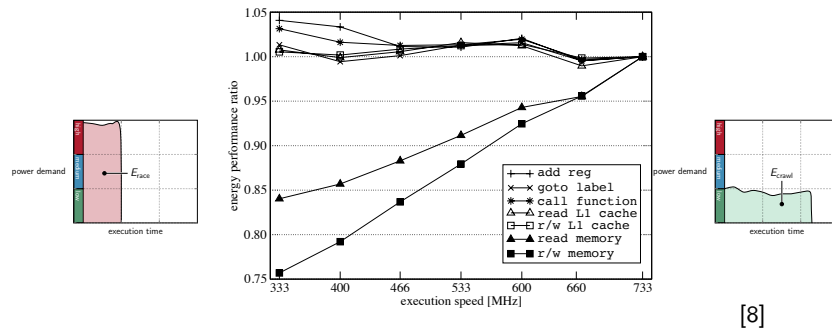


...and so are the energy-aware processing strategies.

Energy-Aware Processing Strategies

- all processing strategies depend on individual system **components** (→ hardware) and responsible **subsystems** (→ software)
1. data processing and computing → CPU
 - general purpose CPU cores as components
 - strategies to reduce energy demand under acceptance of moderate performance impacts
 2. volatile data → uncore, memory
 - uncore and memory as components
 - reduce energy demand of memory components under consideration of necessary performance (i.e., memory bandwidth)

- recap: **conflicting goals** for reducing the energy demand of computation-bound and memory-bound operations

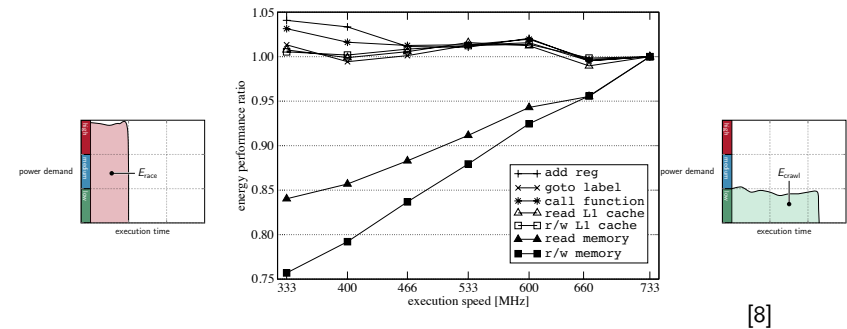


[8]

- naïve approach: run memory-bound and CPU-bound threads with low and high clock speed, respectively



- recap: **conflicting goals** for reducing the energy demand of computation-bound and memory-bound operations

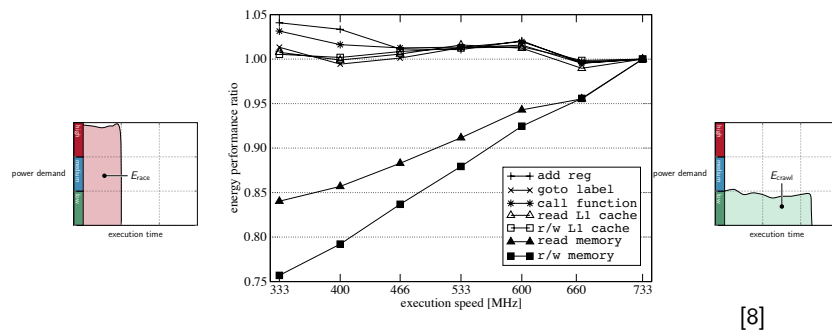


[8]

- considerations and problems of the naïve approach:
 - dynamic characteristics of workloads
 - simple system model (# cores, interlocked voltages, cache size)
 - input-dependent, variable size of working set
 - costs for frequency switching



- recap: **conflicting goals** for reducing the energy demand of computation-bound and memory-bound operations



[8]

- improved energy-aware processing strategies
 - memory-aware scheduling (combining strategy)
 - load/store and execute (sequencing strategy)
 - thread assignment to heterogeneous cores (assigning strategy)



- contention between cores as to resource demand (i.e., cache, memory)
- quad core processor (clock speed 1.6 GHz to 2.4 GHz)
- shared L2 cache by cores in pairs, memory shared by all cores

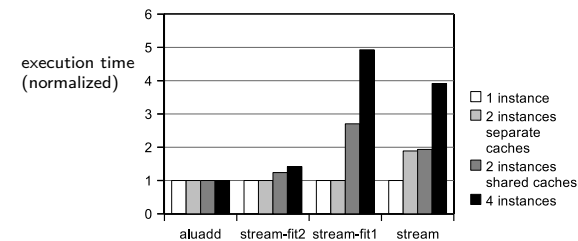


Figure 1. Normalized runtime of microbenchmarks running on the Core2 Quad

[4, 5]

- aluadd: compute-bound
- stream{-fit2,-fit1}: memory-bound, varying size of working set



- contention between cores as to resource demand (i.e., cache, memory)
- quad core processor (clock speed 1.6 GHz to 2.4 GHz)
- shared L2 cache by cores in pairs, memory shared by all cores

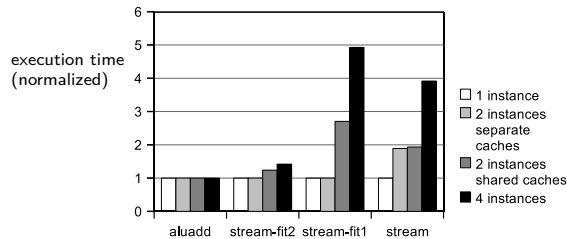


Figure 1. Normalized runtime of microbenchmarks running on the Core2 Quad

[4, 5]

- penalty depends on contention ← process characteristics
- identification of memory-bound process by number of memory transactions



- proposed strategy: **combined scheduling** to **reduce contention**
- co-scheduling of compute-bound and memory-bound processes, based on the concept of Gang scheduling [6]

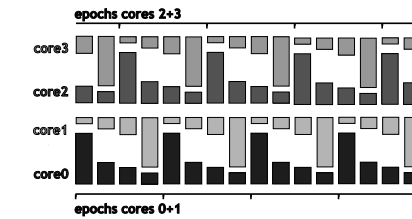


Figure 4. Sorted scheduling. Bars correspond to memory intensity. [4, 5]

- group CPU cores into pairs of two
- run processes with complementary resource demands on each pair



- proposed strategy: **combined scheduling** to **reduce contention**
- co-scheduling of compute-bound and memory-bound processes, based on the concept of Gang scheduling [6]

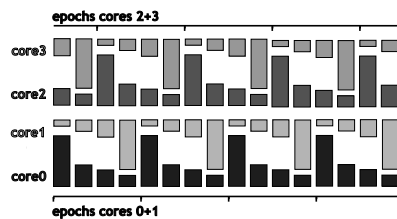


Figure 4. Sorted scheduling. Bars correspond to memory intensity. [4, 5]

- scale to lowest frequency if **no** compute-bound processes are ready → only memory-bound processes are ready
- scale to highest frequency if **at least one** compute-bound process is ready → best results (i.e., lowest EDP) [5]



- proposed strategy: **combined scheduling** to **reduce contention**
- co-scheduling of compute-bound and memory-bound processes, based on the concept of Gang scheduling [6]

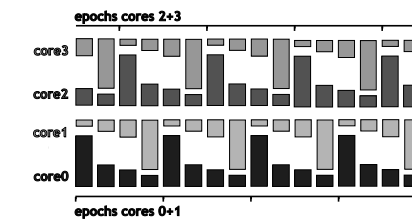


Figure 4. Sorted scheduling. Bars correspond to memory intensity. [4, 5]

- limitations and considerations
 - inferences with scheduling strategy → risk of priority inversion
 - scheduling policy on effective for specific sizes of working set
 - memory hierarchy and cache sizes must be considered



- proposed strategy: **sequenced execution** to **extend phases** of homogenous operations
- fundamental idea based on computer architecture which provides **performance improvements** with **decrease in complexity**

Decoupled Access/Execute
Computer Architectures
(Smith 1982, [7])

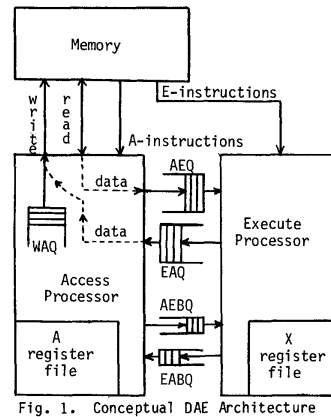


Fig. 1. Conceptual DAE Architecture

- proposed strategy: **sequenced execution** to **extend phases** of homogenous operations
- fundamental idea based on computer architecture which provides **performance improvements** with **decrease in complexity**

```

A7 ← -400
A2 ← 0
A3 ← 1
X2 ← r
X5 ← t
loop: X3 ← z + 10, A2
      X7 ← z + 11, A2 X4
      + X2 * f X3 X3 + X5
      * f X7 X7 + y, A2
      X6 ← X3 + f X4 X4 +
      X7 * f X6 A7 + A7 + 1
      x, A2 + X4
      A2 ← A2 + A3
      JAM loop
  
```

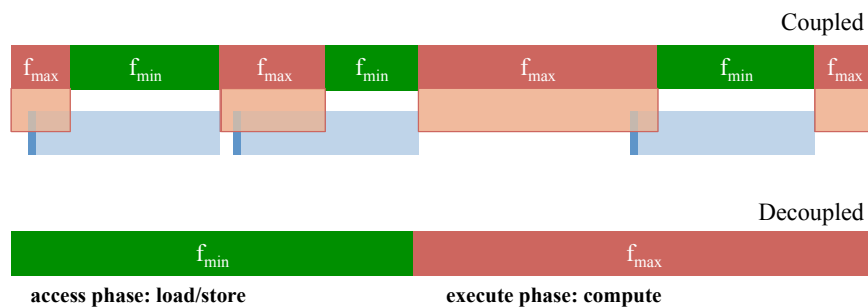
- negative loop count
- initialize index
- index increment
- load loop invariants into registers
- load z(k+10)
- load z(k+11)
- r * z(k+10) - f1.t.mul t.t *
- z(k+11)
- load y(k)
- r * z(x+10) + t * z(k+11))
- y(k) * (above)
- increment loop counter
- store into x(k)
- increment index
- Branch if A7 < 0

Fig. 2b. Compilation onto CRAY-1-like architecture

Access	Execute
.	.
.	.
AEQ ← z + 10, A2	X4 ← X2 * f AEQ
AEQ ← z + 11, A2	X3 ← X5 * f AEQ
AEQ ← y, A2	X6 ← X3 + f X4
A7 ← A7 + 1	EAQ ← AEQ * f X6
x, A2 ← EAQ	.
A2 ← A2 + A3	.
.	.
.	.

Fig. 2c. Access and execute programs for straight-line section of loop

- create two streams for operations of the same kind



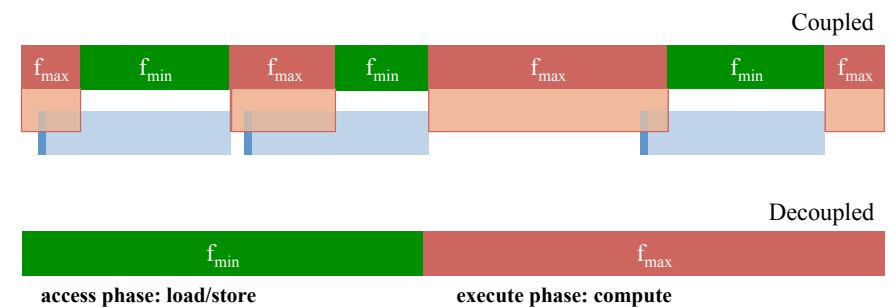
Access Phase

- prefetch data into caches, write intermediate results to memory
- run with low clock speed

Execute Phase

- execute operations on data in hot caches (i.e., computations)
- run with high clock speed

- create two streams for operations of the same kind

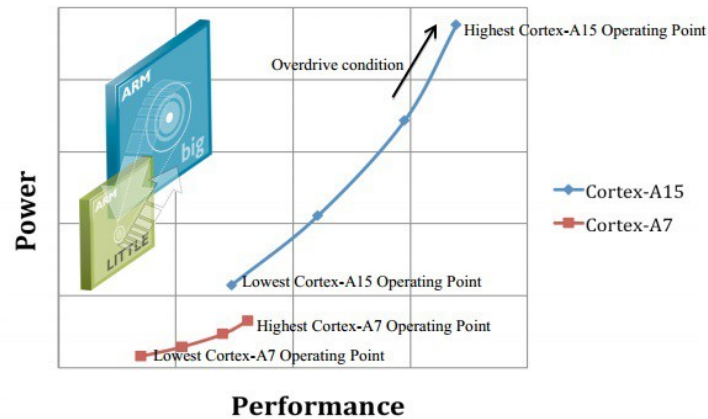


- gains and benefits (cf. [2])
 - reduce voltage and frequency thrashing
 - eliminate unnecessary CPU stalling and memory wait cycles
- limitations and considerations
 - compiler support → open target system and components
 - synchronization efforts (i.e., branches)

Thread Assignment to Heterogeneous Cores

CPU

- proposed strategy: **assigning** homogenous operations to **heterogeneous cores**
- exploit characteristics at the hardware level (i.e., heterogeneous cores)

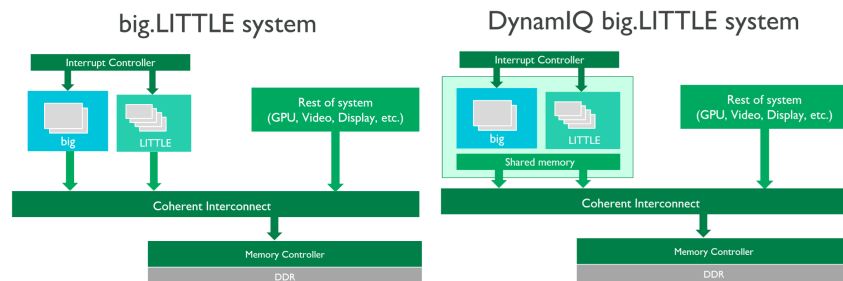


©thoenig EASY (ST 2018, Lecture 5) Components and Subsystems–Data Processing and Computing 20–29

Thread Assignment to Heterogeneous Cores

CPU

- proposed strategy: **assigning** homogenous operations to **heterogeneous cores**
- exploit characteristics at the hardware level (i.e., heterogeneous cores)



©thoenig EASY (ST 2018, Lecture 5) Components and Subsystems–Data Processing and Computing 20–29

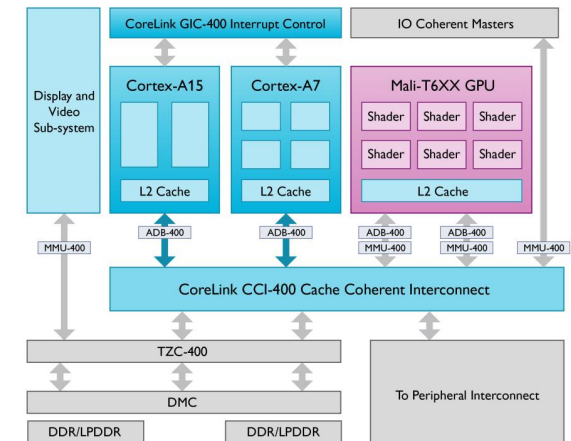
Thread Assignment to Heterogeneous Cores

CPU

- proposed strategy: **assigning** homogenous operations to **heterogeneous cores**
- exploit characteristics at the hardware level (i.e., heterogeneous cores)

- application of previously proposed strategies (i.e., combining, sequencing) depends on

- last level cache
- memory interconnect



©thoenig EASY (ST 2018, Lecture 5) Components and Subsystems–Data Processing and Computing 20–29

Volatile Data

Uncore, Memory

- CPU centric approaches (i.e., DVFS with general purpose CPU cores) influence **only parts** of a system's performance and energy demand
- fine-grained energy demand processing strategies must consider additional components
 - uncore (caches, memory and I/O controllers)
 - memory
 - (external) peripheral

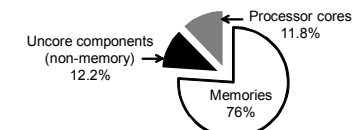
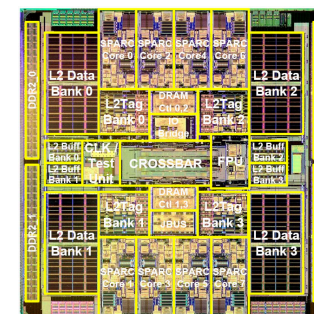


Figure 1. Area breakdown of the OpenSPARC T2 SoC.

[3]

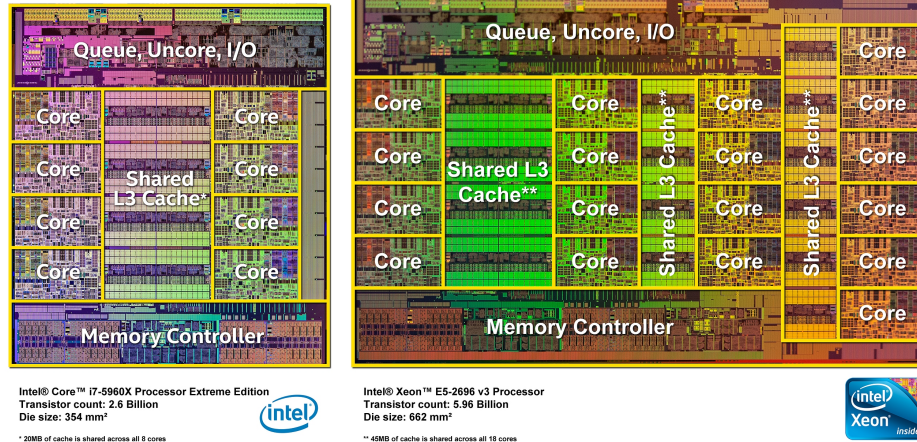


©thoenig EASY (ST 2018, Lecture 5) Components and Subsystems–Volatile Data

22-29

**8-Core Intel® Core™ i7-5960X
Processor Extreme Edition**

18-Core Intel® Xeon™ E5-2696 v3 Processor



- until SandyBridge: linked core and uncore voltages and frequencies
- since Haswell: individual core and uncore voltages and frequencies

Considerations and Caveats

- subsystem control hardware at component level
 - implementation of complex software mechanisms
 - influence on multiple components → multiple dimensions
- cross-component interferences
 - processor cores vs. uncore components vs. memory
 - ...plus external data paths (I/O, network)
- impact of strategies
 - overhead of energy-aware processing strategies
 - state monitoring
 - control algorithms
- upcoming challenges
 - non-volatile memory
 - power capping at component-level

- significant power demand of memory
- DDR memory can operate at multiple frequencies
- explore dynamic voltage and frequency scaling for memory
- apply *classic* DVFS approach
 - lower frequency directly reduces switching power
 - lower frequencies allow lower voltages

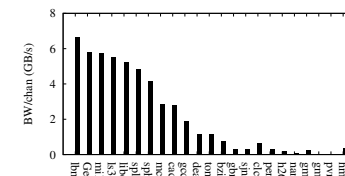


Figure 4: Memory bandwidth utilization per channel for SPEC CPU2006 with 1333MHz memory.

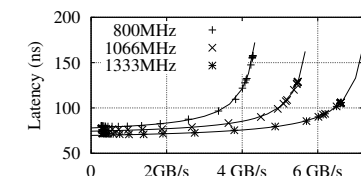


Figure 5: Memory latency in as a function of channel bandwidth demand.

[1]

Subject Matter

- hardware **components** must be controlled by software **subsystems**
- achieve **low energy demand** of the overall system without sacrificing **performance** (too much)
- **composition** of components and subsystem determines the benefit of the overall approach → „greater whole”
- reading list for Lecture 6:
 - ▶ Yuvraj Agarwal et al.
Occupancy-Driven Energy Management for Smart Building Automation
Proceedings of the ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys), 2010.

Reference List I

- [1] DAVID, H. ; FALLIN, C. ; GORBATOV, E. ; HANEPUTTE, U. R. ; MUTLU, O. :
Memory Power Management via Dynamic Voltage/Frequency Scaling.
In: *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*, 2011, S. 31–40
- [2] KOUKOS, K. ; BLACK-SCHAFER, D. ; SPILIOPOULOS, V. ; KAXIRAS, S. :
Towards More Efficient Execution: A Decoupled Access-execute Approach.
In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS'13)*, 2013, S. 253–262
- [3] LI, Y. ; MUTLU, O. ; GARDNER, D. S. ; MITRA, S. :
Concurrent Autonomous Self-test for Uncore Components in System-on-Chips.
In: *Proceedings of the 28th VLSI Test Symposium (VTS'10)* IEEE, 2010, S. 232–237
- [4] MERKEL, A. ; BELLOSA, F. :
Memory-aware Scheduling for Energy Efficiency on Multicore Processors.
In: *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower'08)*, 2008, S. 123–130
- [5] MERKEL, A. ; STOESS, J. ; BELLOSA, F. :
Resource-conscious Scheduling for Energy Efficiency on Multicore Processors.
In: *Proceedings of the 2010 ACM SIGOPS European Conference on Computer Systems (EuroSys'10)*, 2010, S. 153–166



Reference List II

- [6] OUSTERHOUT, J. K. u. a.:
Scheduling Techniques for Concurrent Systems.
In: *Proceedings of the 1982 International Conference on Distributed Computing Systems (ICDCS'82)* Bd. 82, 1982, S. 22–30
- [7] SMITH, J. E.:
Decoupled Access/Execute Computer Architectures.
In: *Proceedings of the 9th Annual Symposium on Computer Architecture (ISCA'82)*, 1982, S. 112–119
- [8] WEISSEL, A. ; BELLOSA, F. :
Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management.
In: *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'02)* ACM, 2002, S. 238–246

