Energy-Aware Computing Systems

Energiebewusste Rechensysteme

V. Components and Subsystems

Timo Hönig

2019-05-23

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

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...

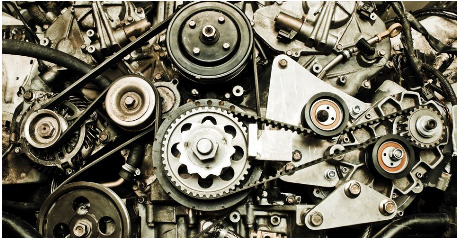
Abstract Concept: Components and Subsystems

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

skoda-storyboard.com

Abstract Concept: Components and Subsystems

- 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

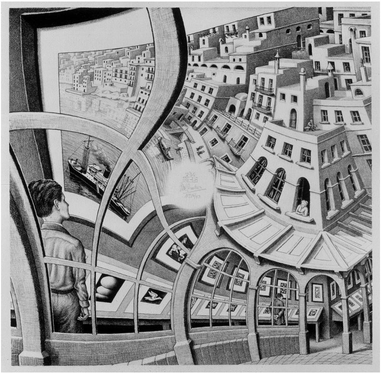
Scopes and Frontiers

- 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.

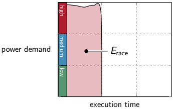
Energy-Aware Processing Strategies

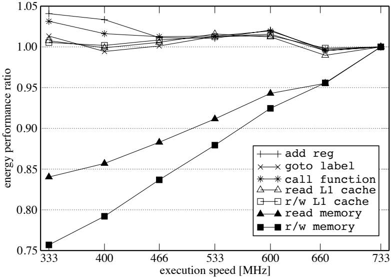
- all processing strategies depend on individual system **components** (→ hardware) and responsible **subsystems** (→ software)

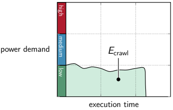
1. data processing and computing → CPU
   - general purpose CPU cores as components
   - strategies to reduce energy demand under acceptance of moderate performance impacts
2. volatile data → uncore, memory
   - uncore and memory as components
   - reduce energy demand of memory components under consideration of necessary performance (i.e., memory bandwidth)

Data Processing and Computing

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





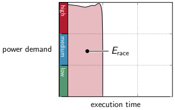


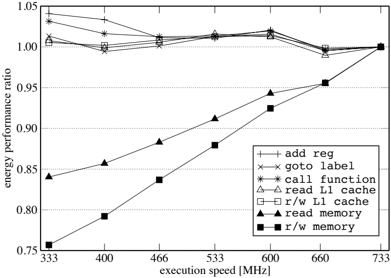
[8]

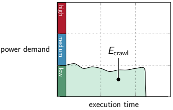
- naïve approach: run memory-bound and CPU-bound threads with low and high clock speed, respectively

Data Processing and Computing

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





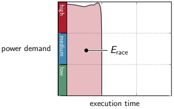


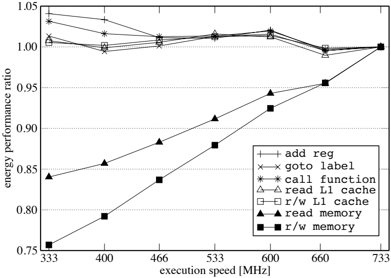
[8]

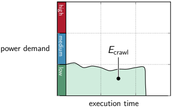
- considerations and problems of the naïve approach:
  - dynamic characteristics of workloads
  - simple system model (# cores, interlocked voltages, cache size)
  - input-depended, variable size of working set
  - costs for frequency switching

Data Processing and Computing

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







[8]

- improved energy-aware processing strategies

1. memory-aware scheduling (combining strategy)
2. load/store and execute (sequencing strategy)
3. thread assignment to heterogeneous cores (assigning strategy)

Memory-aware Scheduling (Combining)

- 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

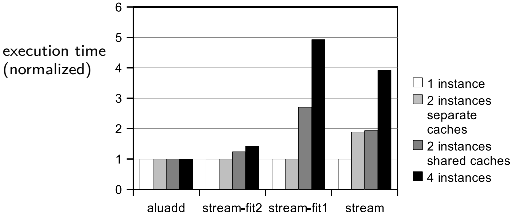


Figure 1. Normalized runtime of microbenchmarks running on the Core2 Quad

[4, 5]

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

Memory-aware Scheduling (Combining)

- 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

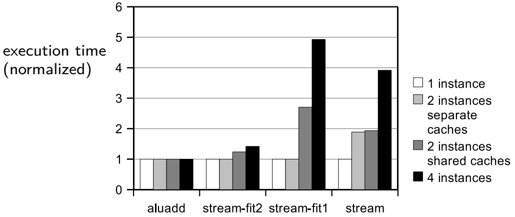


Figure 1. Normalized runtime of microbenchmarks running on the Core2 Quad

[4, 5]

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

Memory-aware Scheduling (Combining)

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

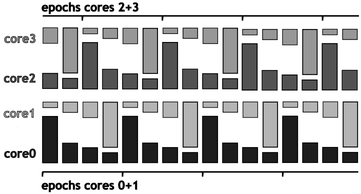


Figure 4. Sorted scheduling. Bars correspond to memory intensity.

[4, 5]

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

Memory-aware Scheduling (Combining)

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

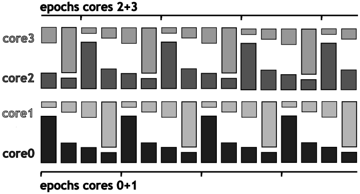


Figure 4. Sorted scheduling. Bars correspond to memory intensity.

[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]

Memory-aware Scheduling (Combining)

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

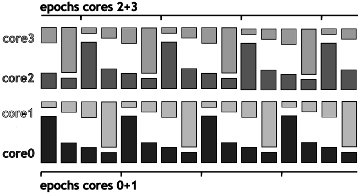


Figure 4. Sorted scheduling. Bars correspond to memory intensity.

[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

Load/Store and Execute (Sequencing)

- 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])

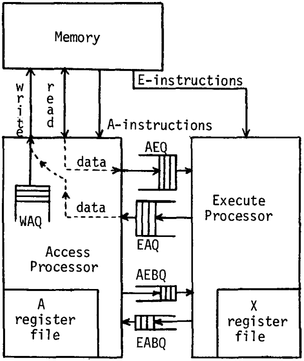
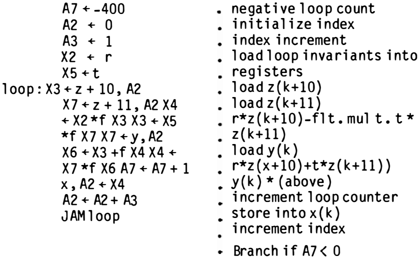


Fig. 1. Conceptual DAE Architecture

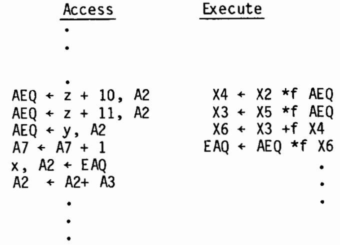
Load/Store and Execute (Sequencing)

- 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              . negative loop count
         A2  ←  0               . initialize index
         A3  ←  1               . index increment
         X2  ←  r               . load loop invariants into
         X5 ← t                 . registers
loop: X3 ← z + 10, A2           . load z(k+10)
         X7 ← z + 11, A2 X4     . load z(k+11)
         ← X2 *f X3 X3 ← X5     . r*z(k+10)-flt.mult.t*
         *f X7 X7 ← y, A2       . z(k+11)
         X6 ← X3 +f X4 X4 ←     . load y(k)
         X7 *f X6 A7 ← A7 + 1   . r*z(x+10)+t*z(k+11))
         x, A2 ← X4             . y(k) * (above)
         A2 ← A2 + A3           . increment loop counter
         JAM loop               . store into x(k)
                                . increment index
                                .
                                . Branch if A7 < 0
```

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



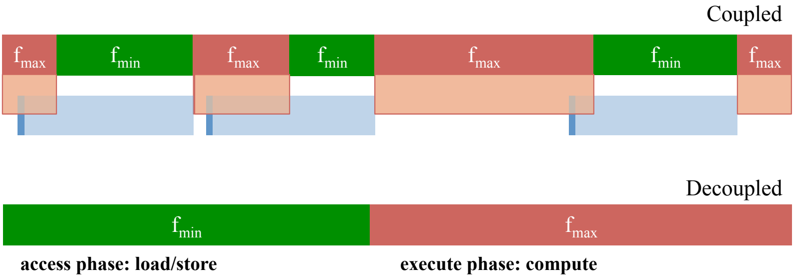
```
        Access                Execute
          .
          .

          .
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

Load/Store and Execute (Sequencing)

- create two streams for operations of the same kind



Access Phase

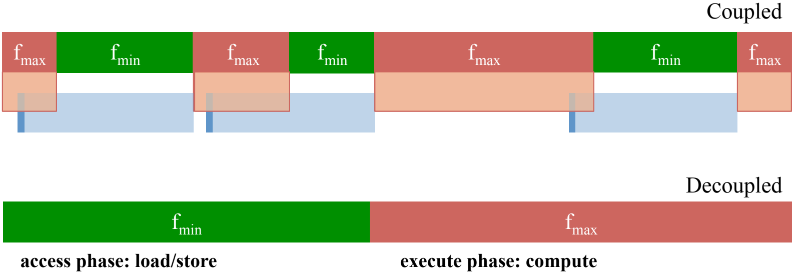
- prefetch data into caches, write intermediate results to memory
- run with low clock speed

Execute Phase

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

Load/Store and Execute (Sequencing)

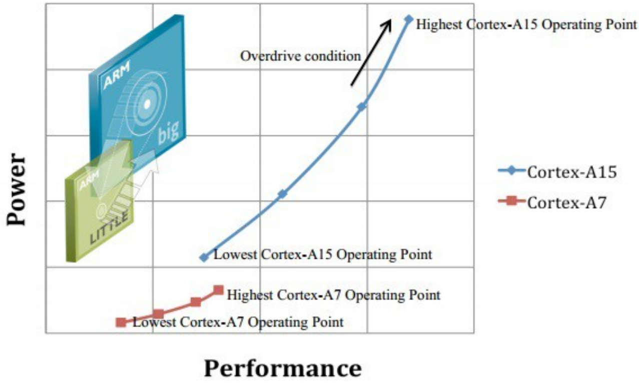
- create two streams for operations of the same kind



- 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

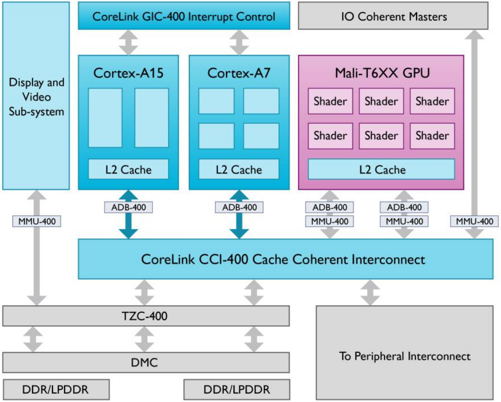
- proposed strategy: **assigning** homogenous operations to **heterogeneous cores**
- exploit characteristics at the hardware level (i.e., heterogeneous cores)



Thread Assignment to Heterogeneous Cores

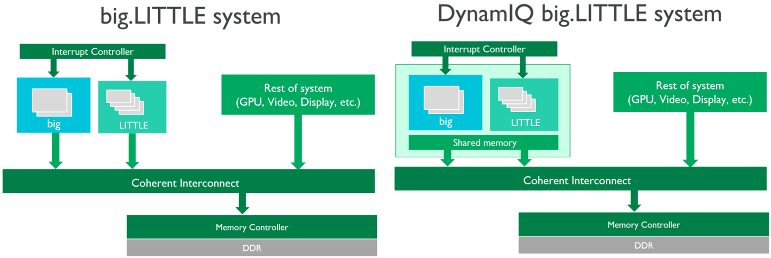
- proposed strategy: **assigning** homogenous operations to **heterogeneous cores**
- exploit characteristics at the hardware level (i.e., heterogeneous cores)
- application of previously proposed strategies (i.e., combining, sequencing) depends on
  - last level cache
  - memory interconnect

...



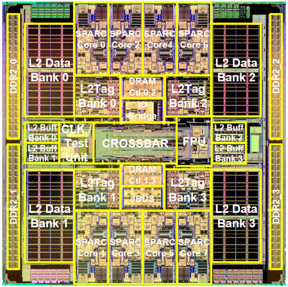
Thread Assignment to Heterogeneous Cores

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



Volatile Data

- 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



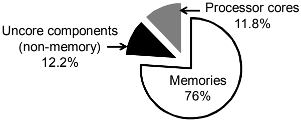
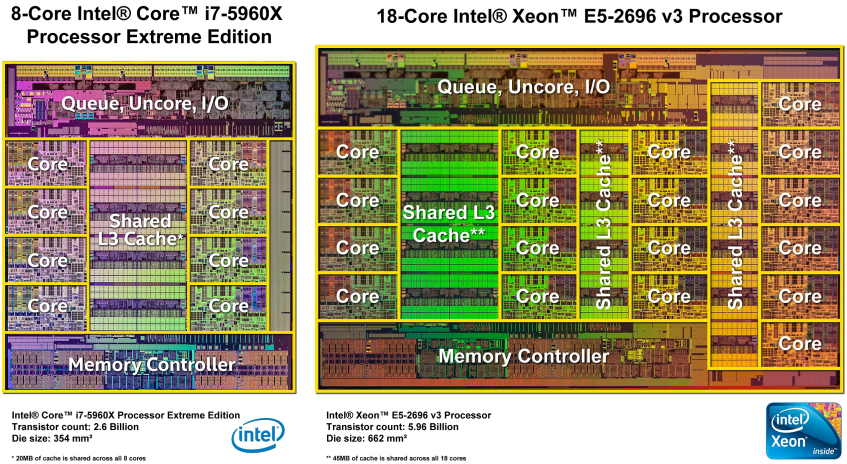


Figure 1. Area breakdown of the OpenSPARC T2 SoC.

[3]

Volatile Data: Caches, Memory and I/O Controllers



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

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

Intel® Core™ i7-5960X Processor Extreme Edition
Transistor count: 2.6 Billion
Die size: 354 mm²

* 20MB of cache is shared across all 8 cores

Intel® Xeon™ E5-2696 v3 Processor
Transistor count: 5.96 Billion
Die size: 662 mm²

** 45MB of cache is shared across all 18 cores

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

Volatile Data: Memory

- 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

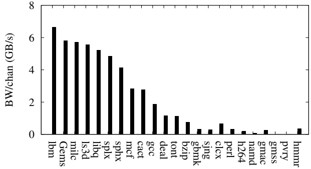


Figure 4: Memory bandwidth utilization per channel for SPEC CPU2006 with 1333MHz memory.

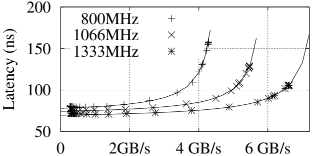


Figure 5: Memory latency in as a function of channel bandwidth demand.

[1]

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

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. ; Hanebutte, 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-Schaffer, 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