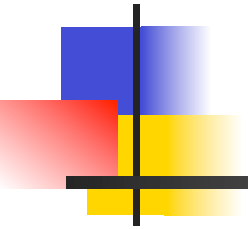
Green Software Development:
Energy Aware Compilers and Green Programming

Y.N. Srikant
Department of
Computer Science and Automation
Indian Institute of Science
Bangalore
srikant@csa.iisc.ernet.in

Outline of the talk

- Introduction and energy models
- Energy-aware compiler optimizations
  - Dynamic voltage scaling
  - Leakage energy optimization using instruction scheduling
  - Scratchpad and drowsy memory allocation
- Green programming
- Summary



Introduction

System-level energy-aware design

Includes

- power and energy modelling.
- energy management issues at
  - microarchitecture, compiler, OS and networking layers of the system.
- and now green programming.

Energy-aware design: Effect of shrinking device size

Processor	Power (Watts)	Freq. (MHz)	Die Size (mm^2)	V_{dd}
21064	30	200	234	3.3
21164	50	300	299	3.3
21264	90	575	313	2.2
21364	100	1000	340	1.5
21464	150	2000	396	1.2

- Conclusion: Shrinking device size does not imply less power dissipation.

From Mudge [18]

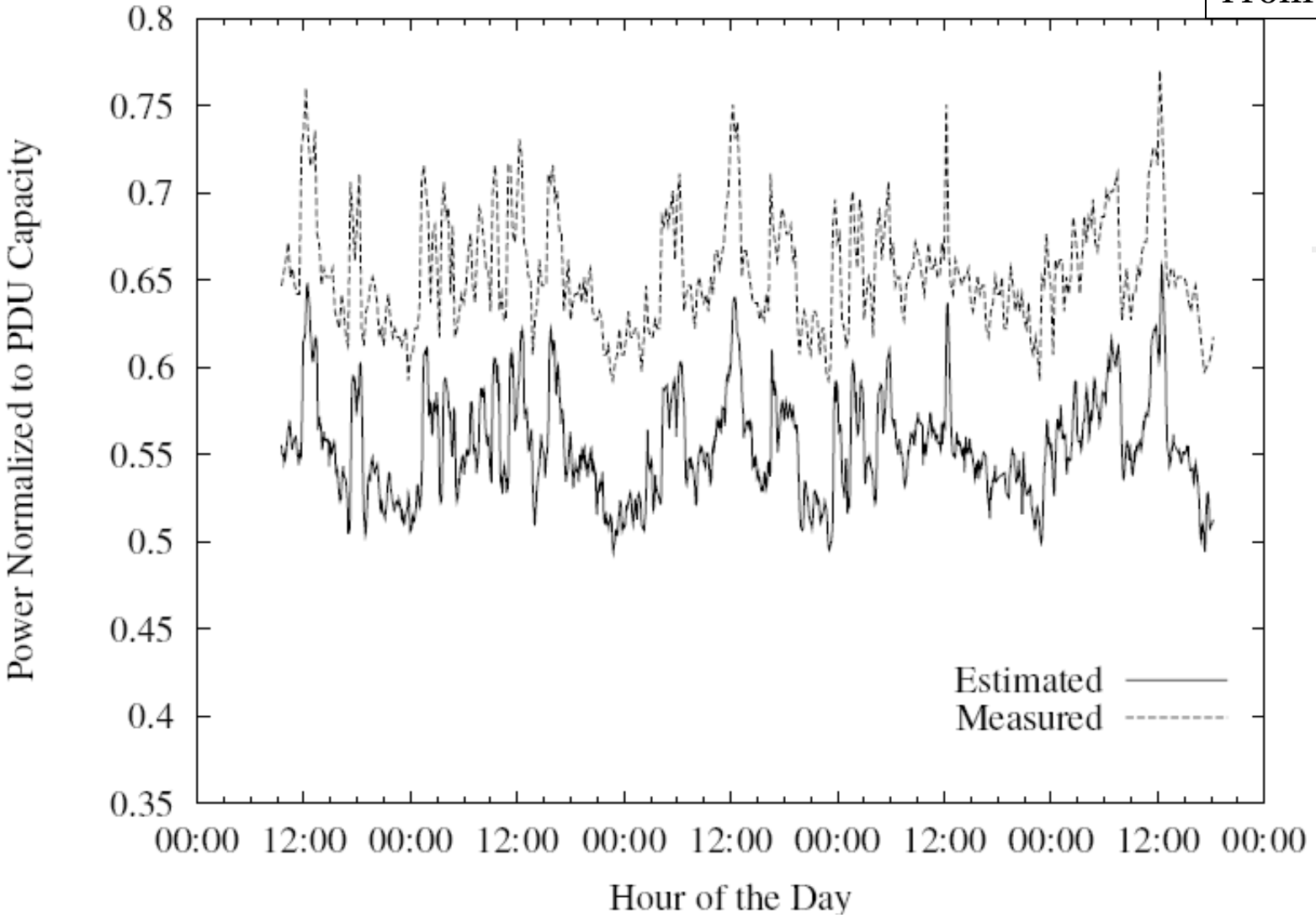
Power and Energy Models

- Needed during early design space exploration for power estimation.
  - May not be very accurate but address relative power efficiency.
- Required to perform optimizations in compilers and OS
  - to estimate power and energy consumption and savings.

Power Estimation in Data Centers

- Measure CPU usage (using performance counters) and total system power, and find a curve that approximates system power against CPU usage (next slide).

From Fan [12]



measured

estimated

Power Estimation - Modelled v/s Measured

Macro- and Micro-Level Power Models

- Instruction level power model.
- Function level power model.
- ALU and register models.
- Cache and memory models.
- Bus and interconnection models.
- Battery models.

CMOS Device-level Power dissipation basics

- Dynamic power dissipation
  - currently dominant.
- Static power dissipation
  - increases dramatically with shrinking device sizes.
- Short-circuit power dissipation
  - Can be controlled only by superior technology.

The Cube-root rule (1)

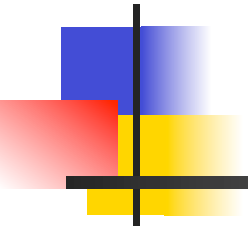
- Assuming a single voltage and frequency for the whole chip, and that $f = kV$

  $PW_{chip} = K_v V^3 = K_f f^3$

  where K_v and K_f are design-specific constants.

The Cube-root rule (2)

- This implies that voltage (hence frequency) reduction is the single most efficient method for reduction of power dissipation.
- V_{DD} cannot be reduced beyond a limit.
- Hence, voltage scaling combined with other techniques needs to be employed to reduce power consumption.

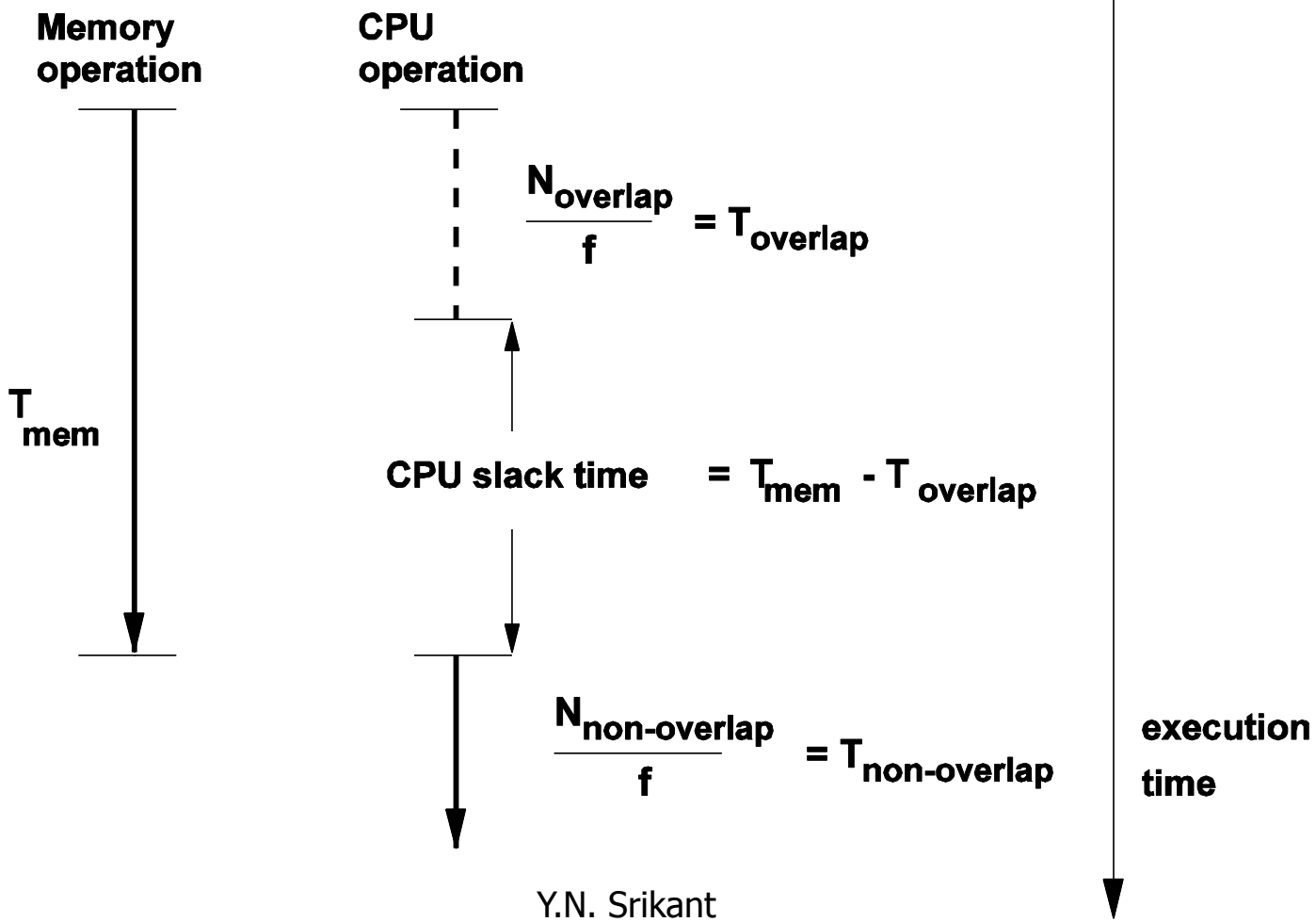


Compiler Techniques

Compiler-Based Dynamic Voltage Scaling

- For memory and I/O bound programs
  - based on profiling.
  - during CPU stall, scale down CPU voltage and frequency.
  - 0% – 25% energy savings with 0% - 3% performance loss.
- Partition program into "regions" based on energy consumption at different (V,f)
  - An optimization problem.

Compiler DVS – CPU Slack



Leakage Energy Optimization

- Leakage energy is the static dissipation energy in CPU, cache, etc.
  - The FUs are in active state, but are not doing any useful work.
  - IALUs are idle for 60% of the time (on the average).
- Even with 70 nm technology, leakage energy consumption will be on par with dynamic energy consumption.

Energy-aware Instruction Scheduling

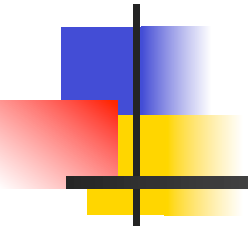
- Uses hardware facility to put ALU into sleep state very efficiently
  - after just one cycle of idleness.
  - frequent transitions between active and sleep states – high energy overhead.
- Reduces #transitions between active and sleep states and increases the active/idle periods.
- Reduces the total energy consumption of FUs.

Heterogeneous Interconnects

- An interconnect composed of two sets of wires.
  - One set optimized for latency and another optimized for energy.
  - Lesser area than two sets of low latency wires.
  - Instr. scheduling can help to reduce energy but maintain performance.

Scratchpad Memory

- As fast as cache memory.
- No tag array, no comparisons.
  - Consumes far lesser amount of energy than cache.
- Software-controlled and needs efficient allocation algorithms.
- Caters to both program and data objects
- Energy benefits: 12% - 43%.
- Performance benefits: 7% - 23%.



Green Programming

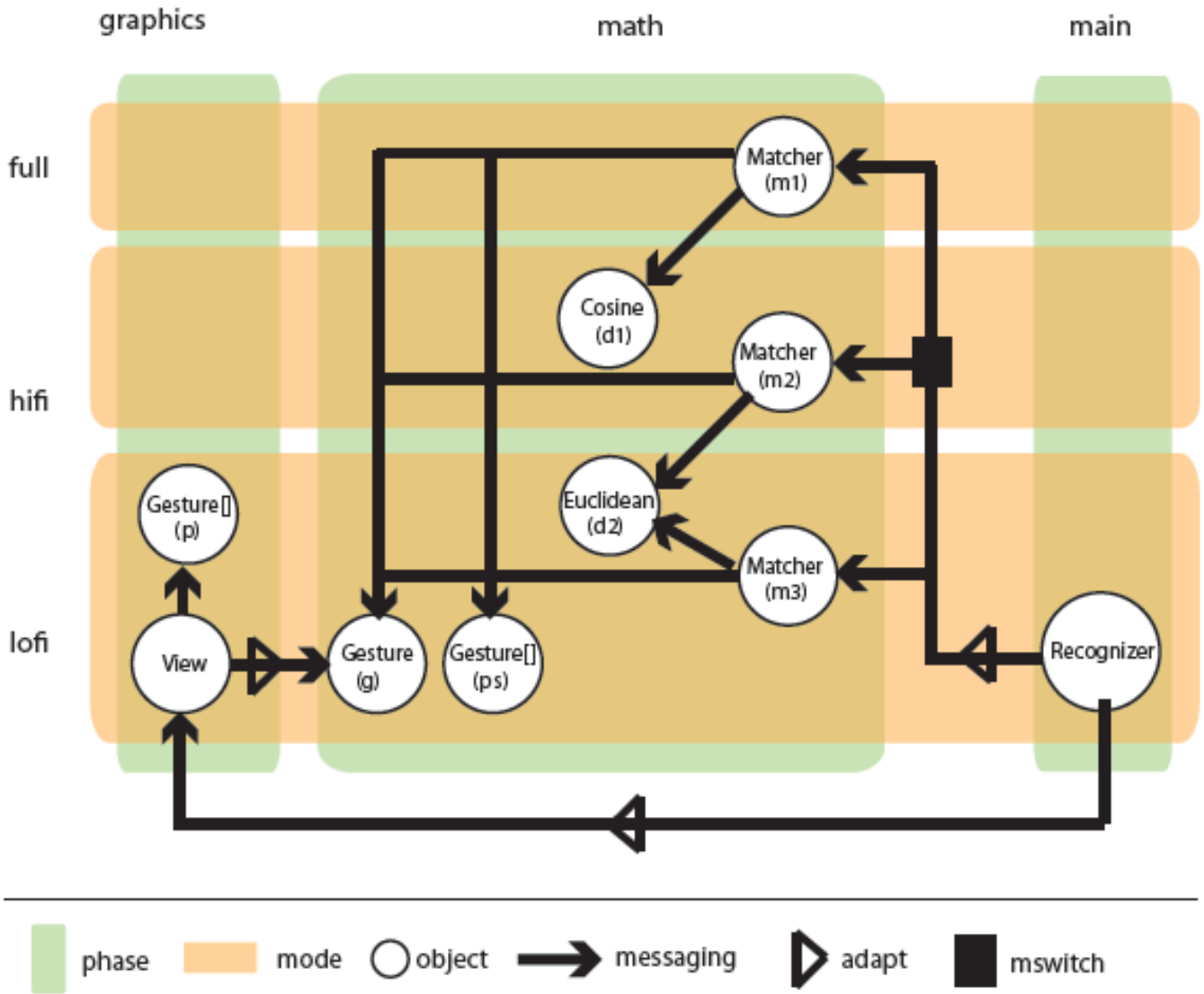
Energy Types (1)

- Programs exhibit phased behavior at runtime.
- Several characteristics change very little during each phase.
  - Cache misses, CPI, energy consumption.
- Techniques for determining phases are well known.
  - Use profiling, dynamic monitoring, and clustering.
- However, programmers also intuitively know phase behavior of their programs.
  - Specially w.r.t. energy consumption.
  - CPU-intensive parts v/s I/O-intensive parts.

Energy Types (2)

- Programmers tag classes/objects with phase type qualifiers.
  - Makes inter-phase object interaction explicit.
  - Objects are also partially ordered w.r.t. energy consumption (e.g., graphics <cpu main).
  - DVS can be enforced using this partial order.
- Within phases, each object is tagged with a mode: full, hifi, or lofi (battery levels).
  - Objects with lower level modes are used when battery levels are lower.

Phases and Modes – An Example [21]



Energy Types (4)

- All the above features can also be implemented by an expert programmer without such language features.
  - However, this entails several problems of usability, platform dependence, etc.
  - For example, DVFS calls may be inserted incorrectly or forgotten. They also have to be changed according to the platform. Frequencies may be lowered but not increased again.

Energy Types (5)

- With ET, no need to think about low-level CPU frequency scaling.
  - Rather, think how CPU-intensive each program fragment (phase) is, and declare phases.
  - Compiler automatically inserts DVFS calls.
- With ET, no need to memorize that object A consumes more energy than object B
  - Rather, declare the mode of the objects and let the compiler do the checking.

The Green Framework (1)

- Approximations are common in real life.
  - Images, videos, and music are compressed and encoded by compromising on quality.
  - Game programmers employ such techniques for ray tracing on PCs and smart phones.
- The Green Framework provides
  - programming support for approximating expensive loops and functions in programs.
  - statistical guarantees that the specified QoS will be met.

The Green Framework (2)

- Programmers specify
  - approximate versions of functions.
  - tolerable maximal QoS loss
    - due to approximations of functions and
    - premature termination of loops.
- How to compute QoS loss?
  - Result returned by a function.
  - Loops – programmer provided code.

The Green Framework (3)

- Green builds a calibration program.
- Runs it using programmer-provided inputs and builds a QoS model.
- Uses the QoS model to determine (at runtime) whether to use
  - the approximate version or
  - the normal version.
- Also tweaks the QoS model at runtime.

Approximate Data Types (1)

- How to isolate computations that must be precise from those that can be approximate?
  - Use type qualifiers to declare approximate data variables.
- The compiler maps approximate variables to low-power storage and also uses low-power operations.

Approximate Data Types (2)

- The compiler can also apply more energy-efficient algorithms provided by the programmer.
- In addition, the compiler can statically guarantee isolation of the precise program component from the approximate component.

Approximate Data Types (3)

- Conversion from approximate type to precise type must be explicitly done by the programmer.
- Conversion the other way can be automatic.

Conclusions

- Compiler techniques such as DVS, instruction scheduling, scratchpad and drowsy memory allocation aid the hardware in reducing energy consumption in programs.
- Green programming requires compiler support to go beyond what compilers can do on their own, but does not interfere with compilation technques.
- Green software engineering still has a long way to go and is a promising area of research.

Thank You
Questions?

References (1)

1. Power Reduction Techniques for Microprocessor Systems, V. Venkatachalam, and M. Franz, ACM Computing Surveys, Vol 37, No.3, September 2005.
2. Power-Aware Microarchitecture, D.M. Brooks, et al, IEEE Micro Nov-Dec 2000.
3. System-Level Power Optimization Techniques and Tools, L. Benini and G. De Micheli, ACM TODAES, Vol.5, No.2, April 2000.
4. System-Level Power-Aware Design Techniques in Real-Time Systems, O.S. Unsal, and I. Koren, Proc.IEEE, July 2003.

References (2)

5. Power-Aware Embedded Computing, M.F. Jacome, and A. Ramachandran, in Embedded Systems Handbook, CRC Press, 2009 (2nd ed.).
6. GRACE: A Hierarchical Adaptation Framework for Saving Energy, D.G. Sachs et al, Tech.Rep. UIUCDCS-R-2004-2409, CS Dept., UIUC, 2004.
7. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction, C-H. Hsu and U. Kremer, PLDI 2003.
8. Real-Time Dynamic Voltage Loop Scheduling for Multi-core Embedded Systems, Z. Shao, *et al.*, IEEE Tr.Circuits & Systems, May 2007.

References (3)

9. CONSET: A Cross-Layer Power Aware Protocol for Mobile Ad Hoc Networks, V. Bhuvaneshwar, et al., IEEE Communications, 2004, pp 4067-4071.
10. SyncWUF: An Ultra Low-Power MAC Protocol for Wireless Sensor Networks, X. Shi, and G. Stromberg, IEEE Tr. Mobile Computing, Vol. 6, No. 1., Jan 2007.
11. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance, Q. Wu, et al., MICRO-38, 2005.
12. Power Provisioning for a Warehouse-sized Computer, Fan et al, ISCA 2007.

References (4)

13. INTACTE: An Interconnect Area, Delay, and Energy Estimation Tool for Microarchitectural Explorations, Rahul Nagpal, Arvind Madan, Bharadwaj Amrutur, and Y.N. Srikant, CASES, October 2007.
14. Compiler Assisted Leakage Energy Optimization for Clustered VLIW Architectures, Rahul Nagpal, and Y.N. Srikant, EMSOFT, October 2006.
15. Energy-aware Compiler Optimizations, Y.N. Srikant, and K. Ananda Vardhan, in: The Compiler Design Handbook: Optimization and Machine Code Generation, 2nd ed., CRC Press, 2008.
16. Exploring Energy-Performance Tradeoffs for Heterogeneous Interconnect Clustered VLIW Processors, Rahul Nagpal, and Y.N. Srikant, HiPC, December 2006.

References (5)

17. Evaluation of dynamic voltage and frequency scaling for stream programs, Arun Rangasamy, Y. N. Srikant, ACM Conf. Computing Frontiers, 2011, Article no. 40.
18. Power: A First-Class Architectural Design Constraint, T.Mudge, IEEE Computer April 2001.
19. Exploiting Critical Data Regions to Reduce Data Cache Energy Consumption, Anand Vardhan K., and Y.N. Srikant, SCOPES, 2014.
20. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, D. Brooks, Vivek Tiwari, and Margaret Martonosi, ISCA 2000.

References (6)

21. Energy Types, Michael Cohen, Haitao Steve Zhu, Senem Ezgi Emgin, and Yu David Liu, OOPSLA 2012.
22. Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation, Woongki Baek and Trishul M. Chilimbi, PLDI 2010.
23. EnerJ: Approximate Data Types for Safe and General Low-Power Computation, Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman, PLDI 2011.
24. SEEDS: A Software Engineer's Energy-Optimization Decision Support Framework, Irene Manotas, Lori Pollock, and James Clause, ICSE 2014.