Energy-Aware Software Systems

Y.N. Srikant
Computer Science and Automation
Indian Institute of Science
Bangalore

NPTEL Course on Compiler Design

Outline

1. Motivation
2. Energy-aware design: an introduction
3. Clouds and Data Centers: A Case Study
4. Power and Energy models
5. OS and System/Application level optimizations
6. Power-aware Networks
7. Energy-efficiency of System Models
8. Microarchitectural Techniques to save Energy in CPU
9. Compiler Techniques to save dynamic energy in CPU
10. Compiler Techniques to save static energy in CPU
11. Saving communication energy in CPU
12. Energy-aware Memory
13. Summary

Motivation

- This set of lectures is designed to show that optimizing energy consumption in computer systems depends not only on the compiler but also on other subsystems – OS, network interface, memory, cache, etc.
- In some way, this also shows the limitations of what compiler optimizations can do!

Energy-Aware Design: An Introduction

Energy-aware design (1)

- Design trade-off: Performance v/s Power Consumption
- Has received much attention in recent years
  - Battery-operated mobile systems
  - Operating costs of large systems
    - A Data Warehouse with 8000 servers needs 2 MW of power

Energy-aware design (2)

- Energy-aware design does not necessarily minimize power or energy, *e.g.,*
  - It is possible to decrease peak power consumption in a processor by delaying issue of some instructions to smoothen instruction issue distribution
  - But this may increase total power/energy consumption due to extra time needed for the execution of the application
  - This is a power-aware, but not low power design

Energy-aware design (3)

- Power and energy efficiency are separate design goals
  - Clock rate reduction reduces power demand, but may increase time and hence energy
- Power-constrained applications and energy-constrained ones are distinct
  - Energy-constrained : e.g., running on batteries (finite amount of energy)
  - Power-constrained : e.g., running on solar power (finite amount of power)

Opportunities for saving Energy - Examples

- Inside CPU – exploiting idleness
- OS and device drivers manage power of peripheral devices
- Software controlled clock management for on-board peripherals and controllers
- Memory controllers manage power of memory subsystems

What is system-level energy-aware design?

- Includes power and energy modelling and management issues at microarchitecture, compiler, OS and networking layers of the system
- Examples
  - Clustered design with local clock reduces the capacitance, 'C' (micro-architecture)
  - Instruction scheduling reduces the activity factor, 'a' (compiler)
  - OS level heuristic reduces 'V_{dd}' and 'f', when peak performance is not needed
  - Network layer puts network interface in standby mode when it is not likely to receive any message

Energy-aware design: Effect of shrinking device size

- With each generational scaling of feature size, more complex, aggressive designs are used
- These designs employ higher clock frequencies, larger chip area, and a much larger number of transistors

Energy-aware design: Effect of shrinking device size

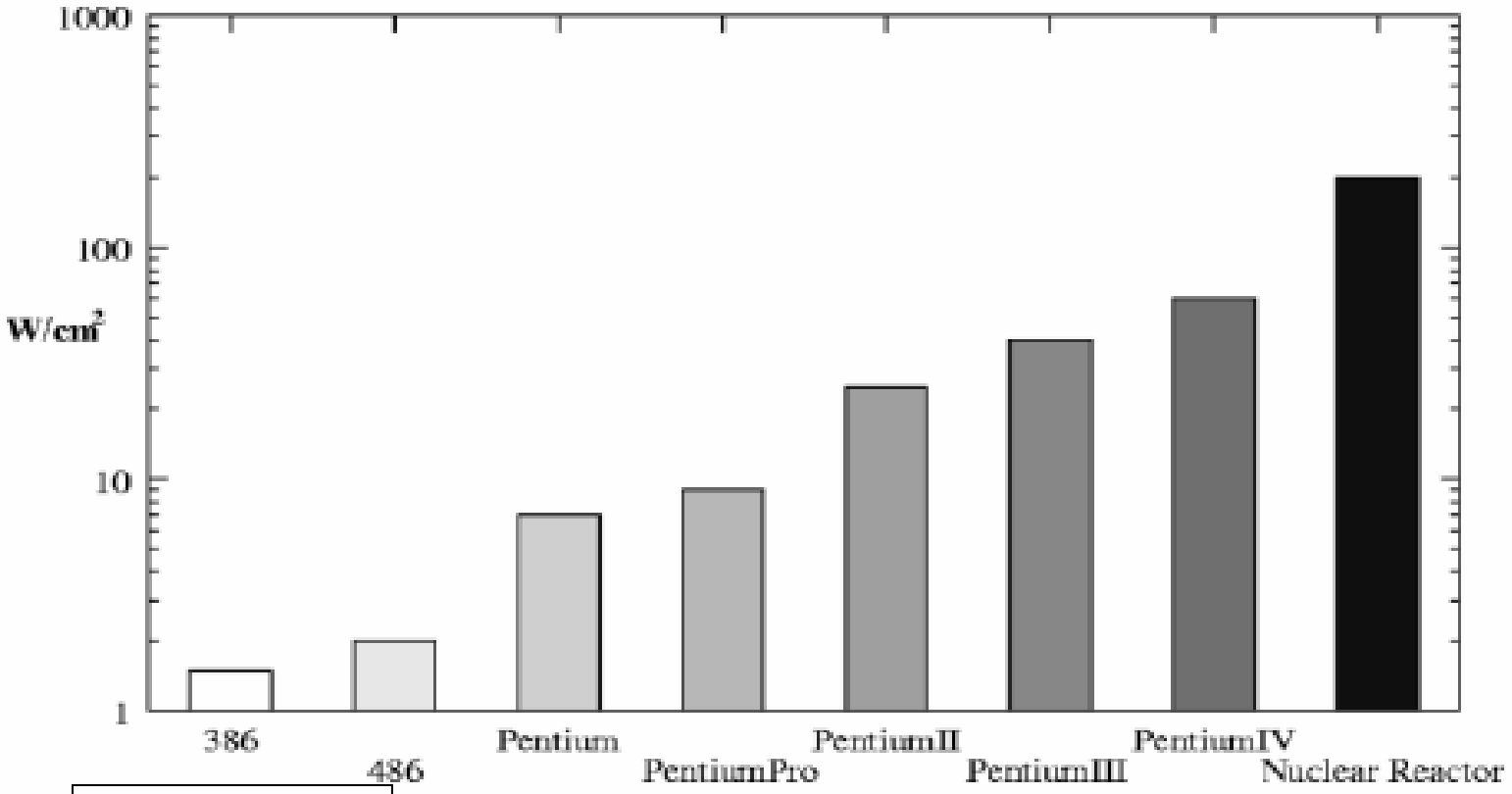
Processor	Power (Watts)	Freq. (MHz)	Die Size (mm^2)	V_{dd}
21064	30	200	234	3.3
21164	50	300	299	3.3
21264	90	575	313	2.2
21364	100	1000	340	1.5
21464	150	2000	396	1.2

From Mudge [18]

Energy-aware design: Effect of shrinking device size

- The result is a significant increase in power dissipation
- Conclusion: Shrinking device size does not imply less power dissipation

Energy-aware design: Effect of power density on chips



From Unsal et al. [4]

Power density of Intel chips

Energy-aware design: Effect of power density in chips

- Decreases reliability and life times of chips
- Decreases battery life
- Increases production cost due to complex cooling and packaging
- Affects environment and also human body
- Power density ultimately precludes further scaling of CMOS chips

Techniques to achieve energy savings at various levels

Classification		Techniques		
Software	Application	Algorithm Design	Scheduling	Leakage Reduction
	Virtual Machine	Resource Hybernation	Memory Management	
	Compiler	Fidelity Reduction	Energy Accounting	
	Operating System	Remote Execution		
Hardware	Architecture	Clock, Memory, and Interconnect Optimizations		
	Gates	Technology Mapping	Gate Restructuring	
	Transistors	Transistor Sizing	Transistor Ordering	

Where does the power go?

		Storage	Communication I/O	Computation	Other
General Purpose Computing	Server	2 DRAM/DISK	3	1	3 Power Supply
	Work Station	2	1 WLAN/LCD	1 GPU/CPU	3
SoC for Digital Conver-gence	Receive	2	1	1	-
	Duplex (Hand-held dev)	3	1 RF/LCD	2 CPU	-
	no Commu-nication	1	1	2 specialized HW	-
Very Low Power (Sensors)		2 may be 1 for non-RF hardware	1	3	-

1 - most important, 3 - least important

Clouds and Data Centers: A Case Study

Clouds and Datacenters (1)

- Warehouse-sized computing systems
- Cost of building datacenter facilities
  - Power capacity provisioning can be as expensive as recurring energy consumption costs
  - Strong economic incentives to operate facilities close to maximum
    - Non-recurring costs can be amortized
- We do not consider power conversion losses and power for cooling

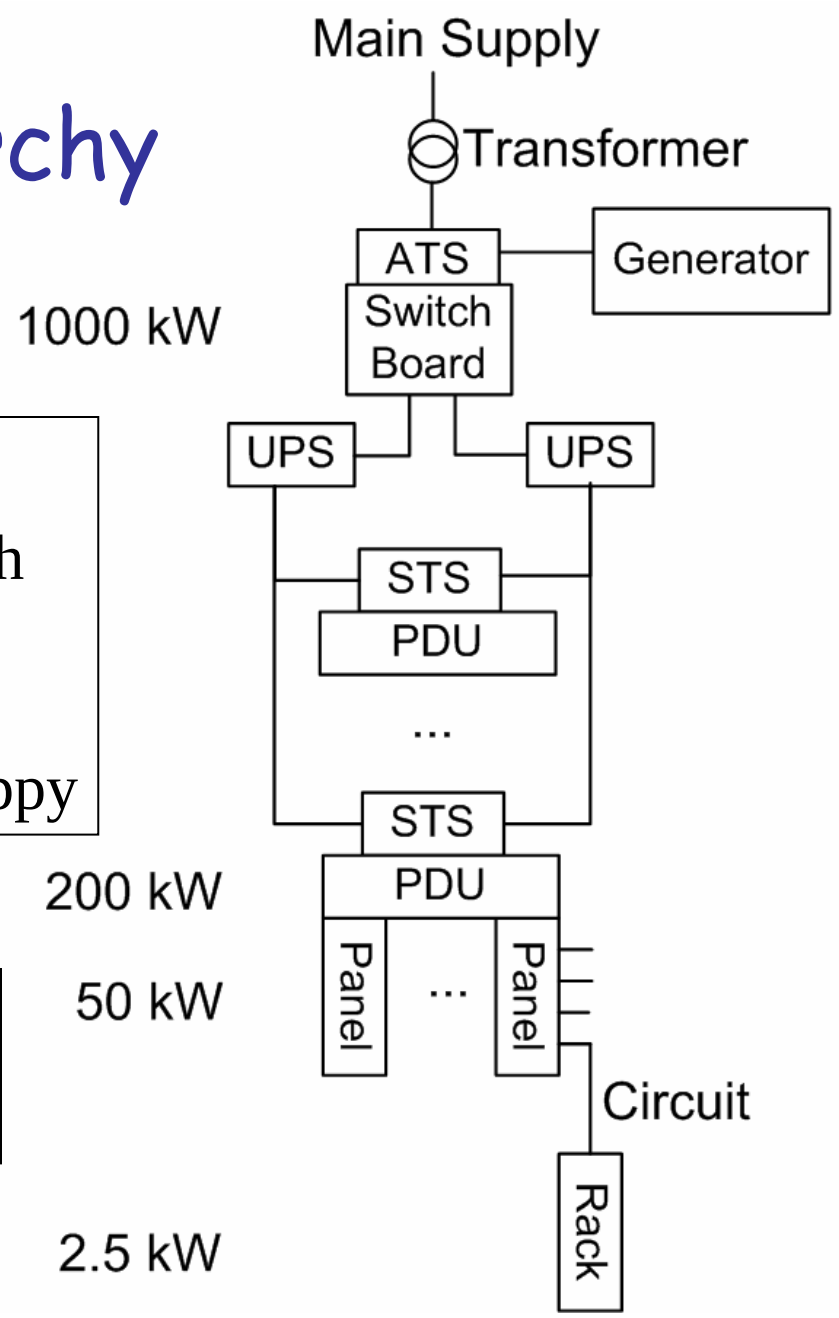
Clouds and Datacenters (2)

- Maintaining max capacity operation is hard in practice
  - Uncertainties in equipment power ratings
  - Power consumption varies with actual computing activity
- Effective power provisioning strategies are needed to determine
  - How much computing equipment can be safely and efficiently hosted within a given power budget
- Business risk of exceeding max capacity
  - Should not result in outages or in
  - Costly violations of service agreements

Determining right deployment and power management strategies

- Requires understanding simultaneous power usage characteristics of 1000's of machines over time
- Important factors
  - Rated max power of computing equipment is overly conservative and not useful
  - Actual consumed power of servers varies significantly with activity
  - Different applications exercise large-scale systems differently
- Hence, only monitoring of large-scale workloads can yield insights into the aggregate load at the datacenter level

Datacenter power distribution hierarchy



Legend:

ATS: Automatic Transfer Switch
PDU: Power Distribution Unit
STS: Static Transfer Switch
UPS: Uninterruptible Power Suppy

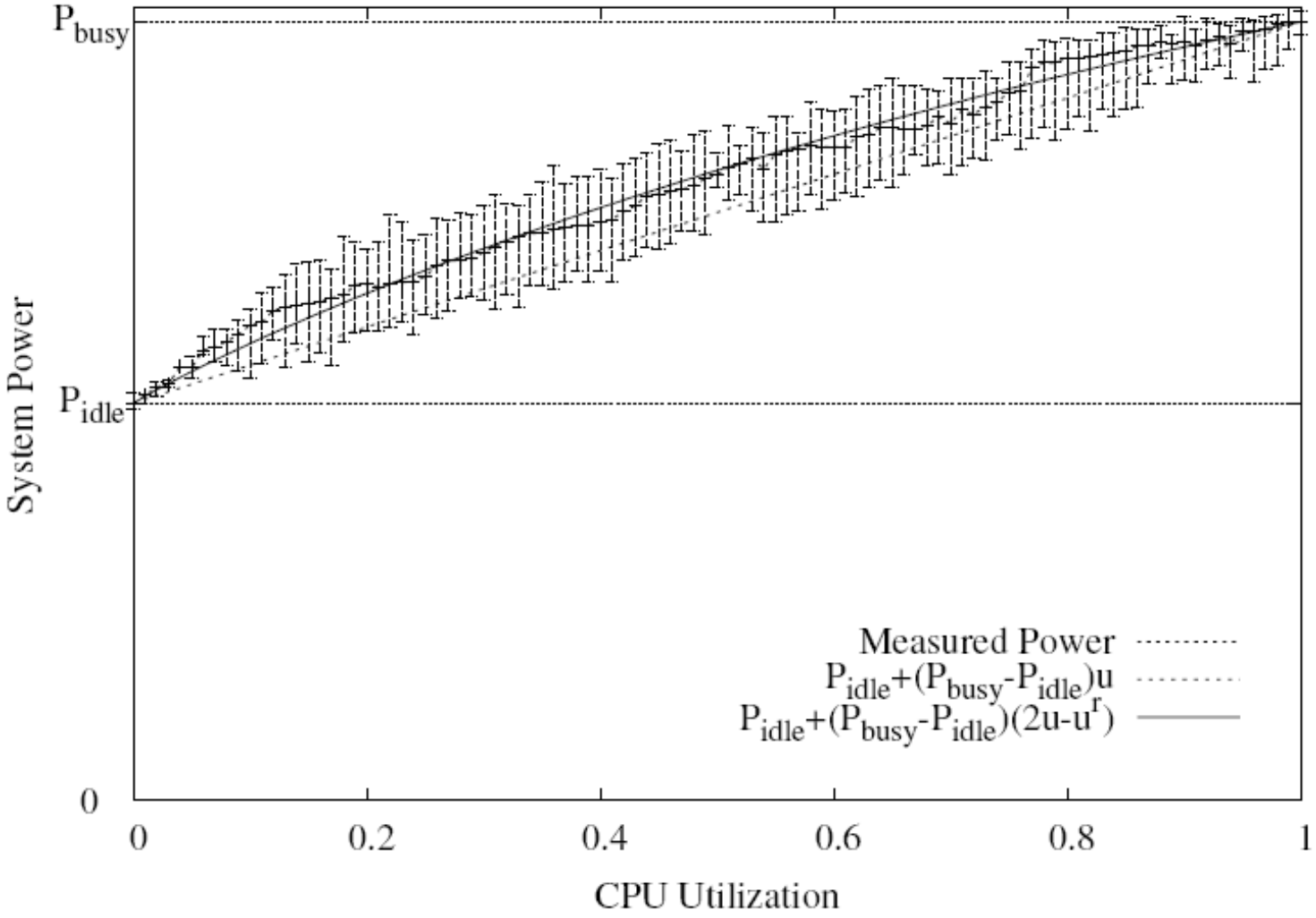
From: Fan et al, Power Provisioning for a Warehouse-sized Computer, ISCA 2007.

Inefficient use of power budget

- Staged deployment
  - Facility is sized to accommodate business demand growth
- Fragmentation
  - Addition of one more unit might exceed that level's limit and such unused capacities add up at the datacenter level
- Conservative equipment ratings (60% more)
- Variable load
- Statistical effects
  - It is unlikely that large groups of systems will be at their peak activity (therefore power) levels as the size of the group increases

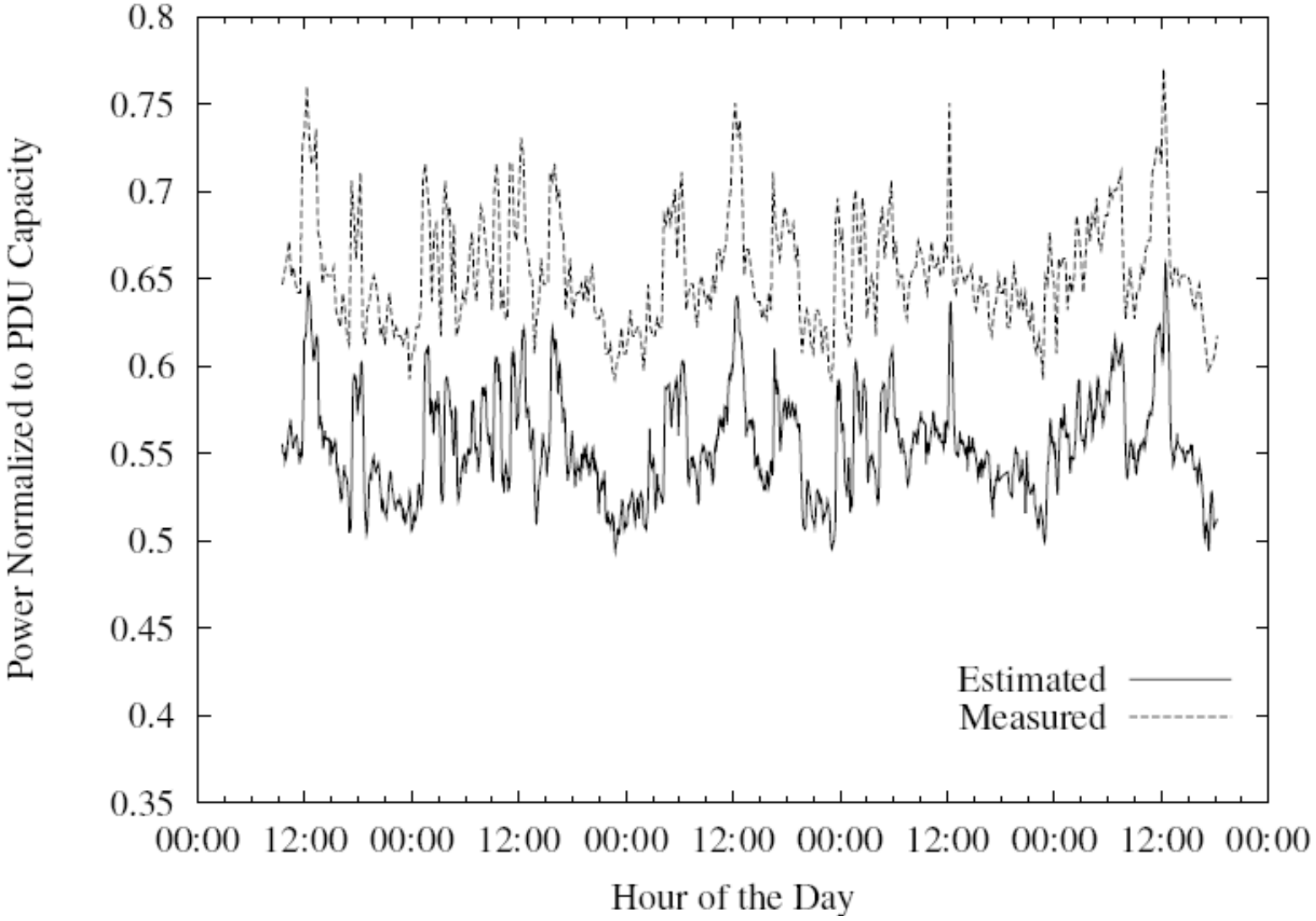
Power Estimation

- Measure CPU usage (using performance counters) and total system power, and find a curve that approximates system power against CPU usage (see figure on next slide)
- Needed for *power capping*



From: Fan et al, Power Provisioning for a Warehouse-sized Computer, ISCA 2007.

Power Estimation-Model Fitting



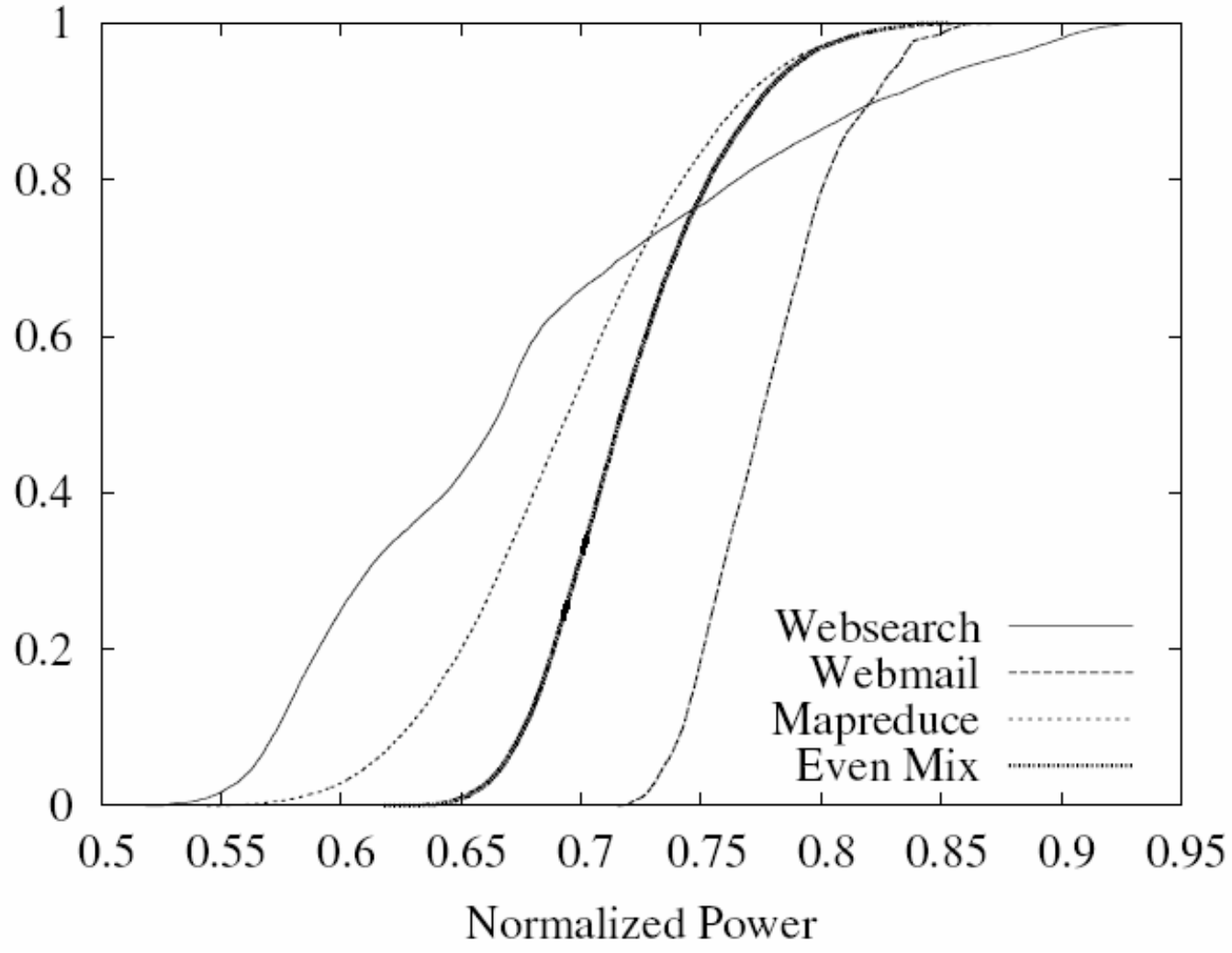
measured

estimated

From: Fan et al, Power Provisioning for a Warehouse-sized Computer, ISCA 2007.

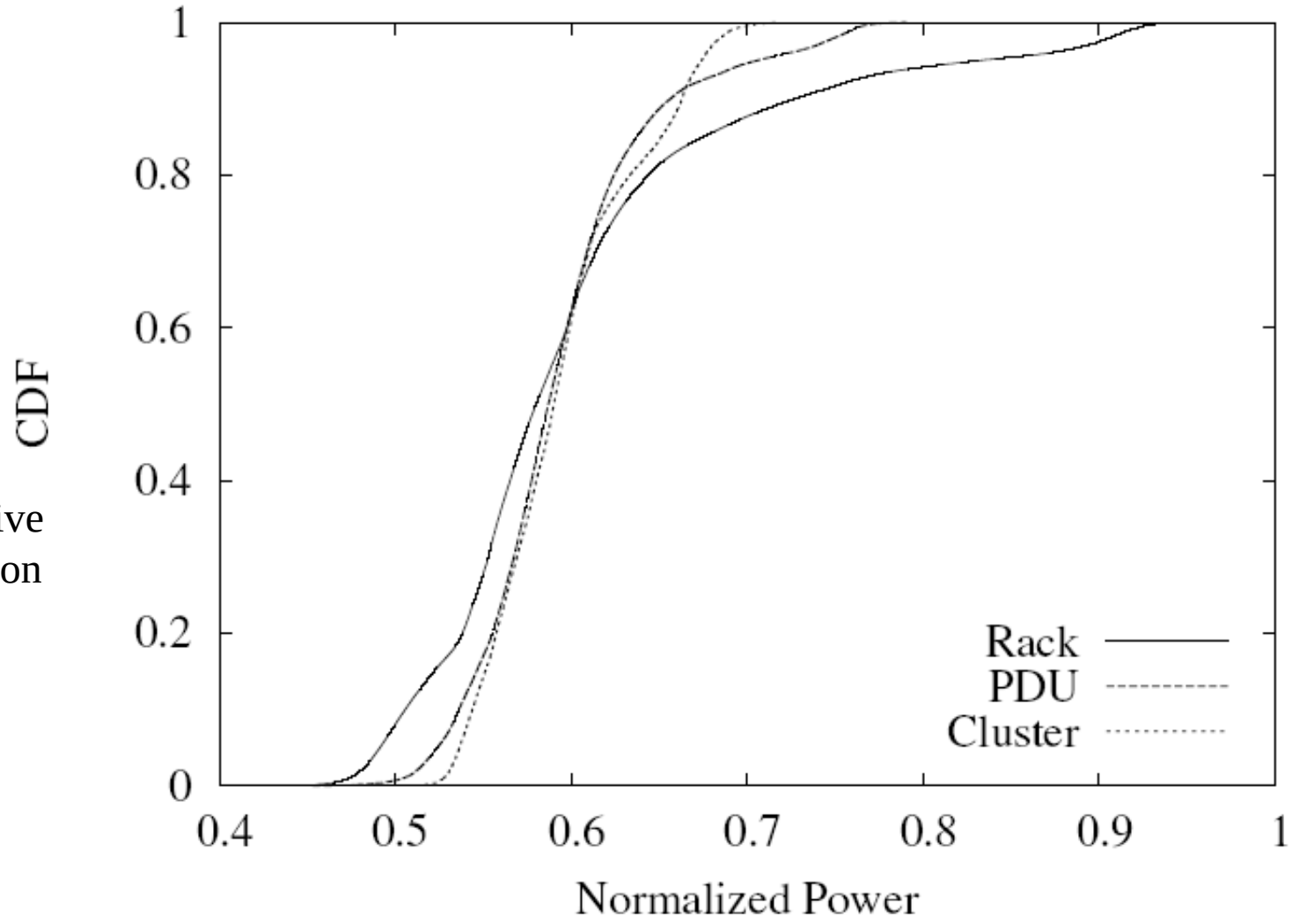
Power Estimation- Modelled v/s Measured

Well Tuned Loads



Cumulative distribution of time; for "Even Mix", power diss. never exceeds 85% of max; for 80% of the time, power diss. is < 75%

From: Fan et al, Power Provisioning for a Warehouse-sized Computer, ISCA 2007.



Cumulative distribution of time

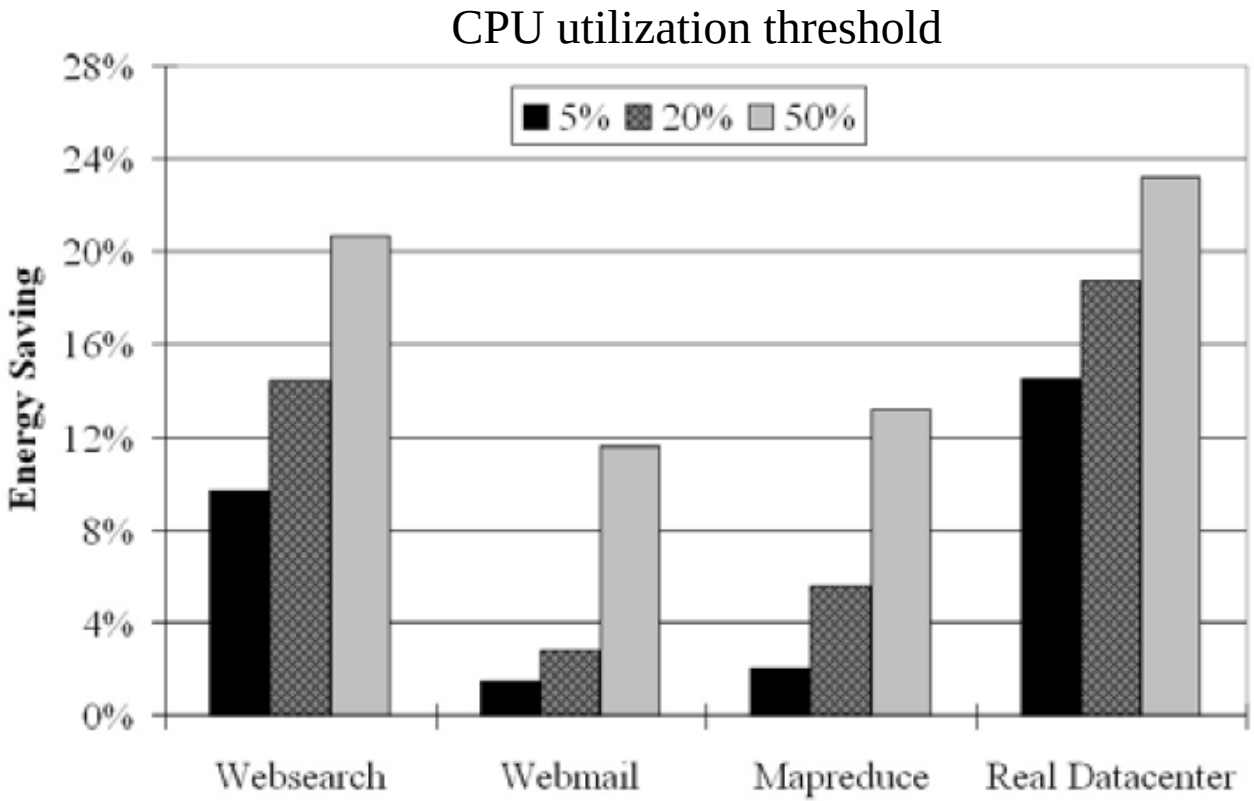
Loads in a Real Datacenter

From: Fan et al, Power Provisioning for a Warehouse-sized Computer, ISCA 2007.

Power Capping

- The system is under-provisioned most of the times
- More processors can be added whenever max power is not used
- Loads can be monitored and whenever power utilization tends to go beyond a *cap,* tasks can be descheduled or dynamic voltage scaling can be deployed
  - Works well with loose service level guarantees
  - About 45% increase in machine deployment with just 1% of time spent in power capping mode
  - #trips beyond *cap* is 9 per month and average length of trips is 48 minutes

Impact of dynamic voltage scaling on energy savings at the Data Centre Level



From: Fan et al, Power Provisioning for a Warehouse-sized Computer, ISCA 2007.

Power and Energy Models

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

Instruction- and Function-Level Power Models (1)

- Assign a power cost to each assembly instruction class
- Experimentally measure current drawn by a processor while executing a sequence of instuctions
- Many inter-instruction effects
  - e.g., cache hit/miss, pipeline interlock
- Expensive
  - large number of inter-instruction effects
  - involves collecting and analyzing large instruction traces

Instruction- and Function-Level Power Models (2)

- Macro models characterizing the average energy consumption of a function or a group of functions Example:
  - choose a quadratic power model, an^2+bn+c, say for insertion sort (n elem)
  - measure actual power dissipation for different values of n
  - Use regression analysis to find a,b,c

Instruction- and Function-Level Power Models (3)

- Such high level models allow designers to assess a number of candidate architectures and alternative software implementations
- For detailed analysis and design, power and energy models of main sub-systems and components are needed

Micro-architectural Models

- Built on top of cycle-accurate simulators such as simplescalar (e.g., *wattch* simulator)
- Measure static and dynamic power consumption
- High simulation time
- Considers power dissipation due to clock distribution also

Cache and Memory Models (1)

- CACTI (Cache Access and Cycle TIme) simulator
  - Given cache hierarchy configuration (size, associativity, #lines) and minimum feature size of target technology
    - generates coarse structural design for such a cache configuration
    - uses built-in models for various constituent elements
      - SRAM cells, row and column decoders, word and bit lines, etc.
    - Makes hit/miss, power, and timing estimates for each access

Cache and Memory Models (2)

- CACTI
  - Using memory traces generated by *simplescalar*, it can generate access-based power dissipation estimates
- DINERO memory simulator
  - simulates memory accesses faithfully
  - provides timing information
- Cache and memory simulation should be combined with processor simulation for a complete simulation

Bus and Interconnection Models

- Provide estimates of transfer time and energy consumption
- Model #segments and the details of each segment based on technology
- INTACTE
  - an interconnect modeling tool
  - enables co-design of interconnects with other architectural components

Battery Models

- Capacity (and hence lifetime)
  - non-linear function of current drawn
  - capacity = k/I^{α}
  - ampere-hours is a constant (apprx.)
- Tradeoffs between quality, performance, and duration of service can be implemented at system level
  - by taking such non-linearity into account
  - however, this needs better battery models

CMOS Device-level Power dissipation basics (1)

- Dynamic power dissipation
  - when a circuit performs the functions it was designed for
  - currently dominant
  - depends on circuit size/complexity, speed/rate, and switching activity

CMOS Device-level Power dissipation basics (2)

- Static power dissipation
  - needed to preserve the logic state of circuits between switching activity
  - caused by sub-threshold leakage mechanisms
  - increases dramatically with shrinking device sizes
  - Significant for technologies below 70nm

CMOS Device-level Power dissipation basics (3)

- Short-circuit power dissipation
  - can be controlled only by
    - superior technology
    - different semiconductor materials
  - due to through current during the switching of a logic gate
  - usually less than 10% of dynamic power in well-designed circuits and can be ignored

CMOS Device-level Power dissipation basics (4)

- Dynamic, static and short-circuit power dissipation in a device (respectively)

$$PW_{device} = (\tfrac{1}{2})\ C\ V_{DD}\ V_{swing}\ a\ f + I_{leakage}\ V_{DD} + I_{sc}\ V_{DD}$$

C : output capacitance, a : activity factor
V_{DD} : supply voltage, f : chip clock frequency
V_{swing} : voltage swing across output capacitor
$I_{leakage}$: leakage current
I_{sc} : average short circuit current

CMOS Device-level Power dissipation basics (5)

- Ignore leakage power and short-circut power
- Usually, $V_{swing} = V_{DD}$

$$PW_{chip} = (\tfrac{1}{2}) \sum C_i \; V_i^2 \; a_i \; f_i$$

  - C_i, V_i, a_i, and f_i are unit or block-specific averages
  - Summation is over all units or blocks at the microarchitecture level (I and D caches, I and FP units, load-store units, register files, and buses)

The Cube-root rule (1)

- Assuming C as a constant (for a given design), worst case activity (a=1), 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
  - lower V_{dd} implies lower threshold voltage to maintain same performance
  - lower threshold leads to larger leakage.
- Hence, voltage scaling combined with other techniques needs to be employed to reduce power consumption

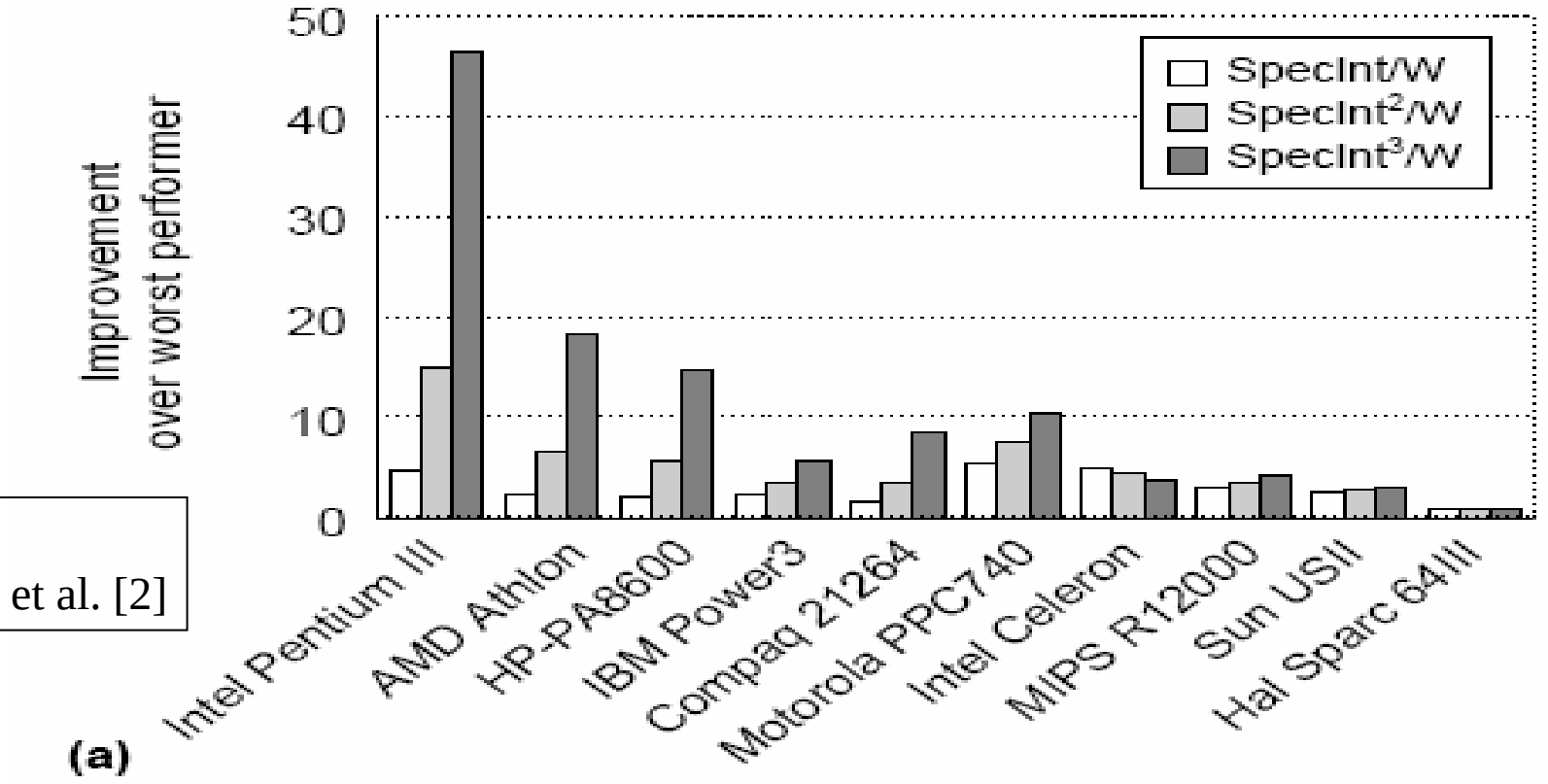
Power-Performance metrics – MIPS/W metric (1)

- Higher the number, the "better" the machine
  - Okay for lower end machines
  - For lowest end m/c, extending battery life even at the cost of performance may be important
  - For servers, where power is not a severe constraint, $(MIPS)^2/W$ or $(MIPS)^3/W$ may be a better choice.

Power-Performance metrics - MIPS/W metric (2)

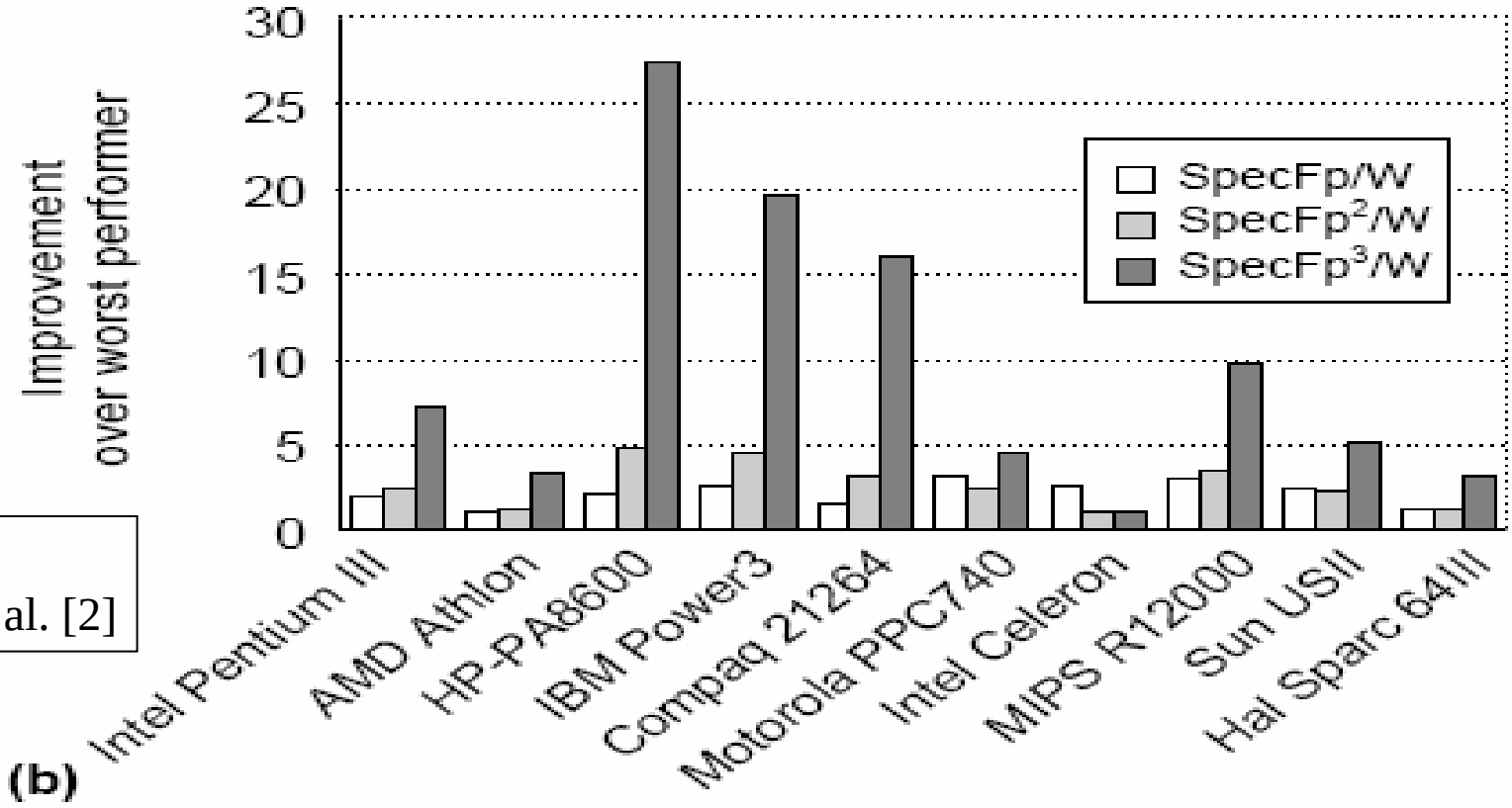
- A higher MIPS/W machine, even though more efficient, may offer a lower level of performance
  - MIPS/W is 1/Energy-per-instruction
  - least energy per instruction is usually obtained for very low voltages where performance is also poor

Power-Performance efficiencies (1)



From Brookes et al. [2]

Power-Performance efficiencies (2)



From Brookes et al. [2]

Power-Performance metrics - PDP and EDP

- Power-delay product (PDP)
  - Suitable for low-power, portable systems, where battery life is the primary index of energy efficiency (PDP is energy)
  - Analogous to MIPS/W
- Energy-delay product (EDP)
  - Suitable for higher end systems
  - Analogous to $(MIPS)^2/W$

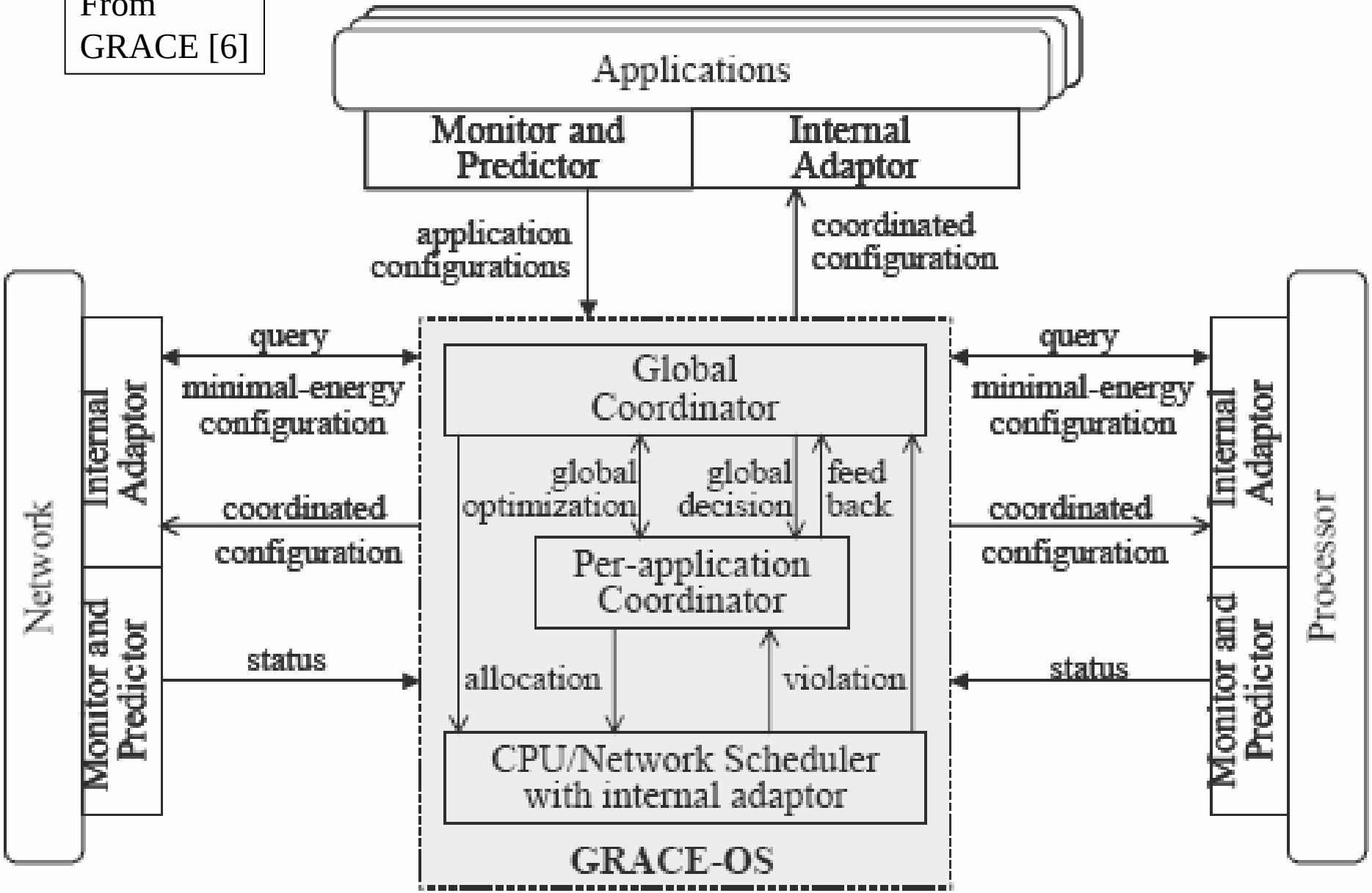
Operating Systems and System/Application level Optimizations

Power-aware Operating Systems

- Dynamic voltage/frequency scaling while scheduling tasks
- Energy-aware scheduling
- I/O Device control (on/off)
- Middleware for coordinated adaptation
- …

The GRACE System

From GRACE [6]



System/Application level Optimizations (1)

- It may be useful to explore different task implementations during design
  - different power/energy versus quality of service for the same functionality, cost, battery etc.
    - tradeoff accuracy for energy savings in a hand-held GPS system
    - image quality for energy savings in an image decoder

System/Application level Optimizations (2)

- Such optimizations may be performed under control of a system-level manager (PM)
  - if battery level drops below a certain threshold
    - PM may drop certain services and/or
    - swap some tasks to less power hungry (lower quality) software versions
    - PM may also shutdown/slowdown subsystems or modules that are idling/under-utilized
  - normally implemented inside OS
  - examples: TinyOS, SOS

Advanced Configuration and Power Interface (ACPI)

- Interface between power managed modules and PM
  - display drivers, modems, hard-disk drivers, processors, network cards, etc.
  - 2 power states – ACTIVE and STANDBY
  - power management policies
    - fixed timeout
    - predictive shutdown
      - use previous history of the subsystem to predict the next expected idle time and based on this, decide to shutdown or not

Power-aware Networks

Power-aware Networks

- Energy impact on network topology and broadcasting
- Power-aware protocols
- Routing optimizations in wireless LANs
- Dynamic clustering of sensor networks
- Energy-efficient packet forwarding in wireless sensor networks
- ...

Low Power MAC Protocol for Wireless Sensor Networks (1)

- Sensors and actuators are integrated into the environment and are powered by cheap batteries
- Charging/changing batteries frequently is not possible
- RF transeiver is one of the biggest power consumers in a sensor node
- Power consumption for idle listening is almost same as that of transmitting
- If RF transeiver is in receive or transmit mode only for 1% of the time (duty cycle), overall system power can be reduced by about 50 times

Low Power MAC Protocol for Wireless Sensor Networks (2)

- Duty cycle scheduling
  - Synchronize the time when receivers are in receive mode with the sending period of transmitter
  - Small duty cycles lead to decreased synch accuracy and increase in network latency
  - Large duty cycles imply more power consumption
  - Tradeoff

IEEE 802.15.4 MAC Protocol

- For master-slave star topology
- Master broadcasts synchronization information using a periodical beacon
  - beacon also mentions the slave to which master has packets to send
- Slaves sleep for most of the time
  - wake up simultaneously to listen to the beacon
  - Slave remains active if self is the target; otherwise sleeps until the next beacon

IEEE 802.15.4 MAC Protocol

- Not very efficient
  - cannot achieve very low duty cycles
  - serves only simple star topology with one master
- Many variations such as, Wake-Up-Frame (WUF), WiseMAC, and SyncWUF (combination of the first two) are available

Power-Aware Routing Protocol for Mobile Ad Hoc Networks

- Routing protocols compute paths from one node to another
- Paths change due to mobility of nodes
- Incorporating energy awareness in routing protocols
  - route discovery and maintenance procedures must compute and maintain energy-efficient routes

CONSET: A Power-Aware Routing Protocol for MANETs

- Each node dynamically computes a connectivity set (CS)
  - a reduced set of that node's neighbourhood
  - guarantees the node's connectivity to the rest of the network
- Transmission power of route request (RREQ) messages is adjusted so that they are sent only to the CS
- The next-hop for a data transmission is selected from the CS of that node
- This may result in longer end-to-end paths (#hops), but they will be more energy-efficient

Energy efficiency estimation from System Models

Energy efficiency estimation from System Models - Algorithms

- Different functionally equivalent algorithms for the same platforms may be made available (with differing energy efficiencies)
- Energy estimates of elementary operations should be obtained by experimentation
- Control Dataflow graphs may be used as an algorithm representation
- Hard to estimate due to dependence on hardware

Energy-Efficient Programming: Some Advice for ARM Processor

- Prefer shifting to multiplication and division by 2
- Predicated (guarded) instructions are more energy-efficient than branching
- Table lookup is better than if-then-else for large switch statements
- Integer types are more energy-efficient
- Pass function parameters in registers

Energy efficiency estimation from System Models– Task Graphs

- Aim is to obtain minimal energy mapping from a task graph to an architectural template
- Requires pre-characterization of power consumption of each task on various platforms and for various voltages
- Overheads due to hardware resource sharing by tasks are not easy to estimate

Microarchitectural Techniques to save Energy

Microarchitectural Level Power-aware Design

- CPU
  - Voltage and frequency scaling
  - Supply voltage gating of function units
  - Clock gating of function Units
  - Bus encoding etc.
- Memory
  - Drowsy cache
  - Compression in I-cache
  - Cache region reservation and partitioning
  - Scratchpad memory

CPU: Voltage Scaling

- Reduction of voltage saves mostly dynamic energy
- In CMOS circuits, delay increases with reduction in voltage
- This in turn requires clock frequency reduction
- Intel's XScale 80200 processor
  - voltage 1.0v to 1.5v in small increments
  - frequency 200 – 733 MHz in steps of 33/66 MHz
  - time for change in voltage: upto 1 ms

Static Voltage/Frequency Scaling

- Intel's speedstep technology
  - detects if system is plugged into a power outlet or is on batteries
  - accordingly runs processor at highest v/f or less power-hungry mode

Dynamic Voltage Scaling

- Possible at task level and done by OS task scheduler
- Modern day embedded processors provide for DVS through program instructions
  - Intel XScale, StrongARM, AMD Mobile K6 Plus, Transmeta Crusoe, PowerPC 405LP

CPU Clock gating

- Clock gating of Function Units
  - Make FU clock zero during idle period
  - Reduces dynamic energy usage, but static leakage remains
  - Both circuit-level techniques and compiler techniques are possible

CPU: Voltage gating

- Supply voltage gating of function units
  - FUs are switched off during idle periods
  - Saves static energy
  - Circuit level automatic techniques have been proposed
    - Sense idle periods (based on history) and act
  - Compiler optimization and program control are possible (recommended)

Compiler Techniques to save Dynamic Energy in CPU

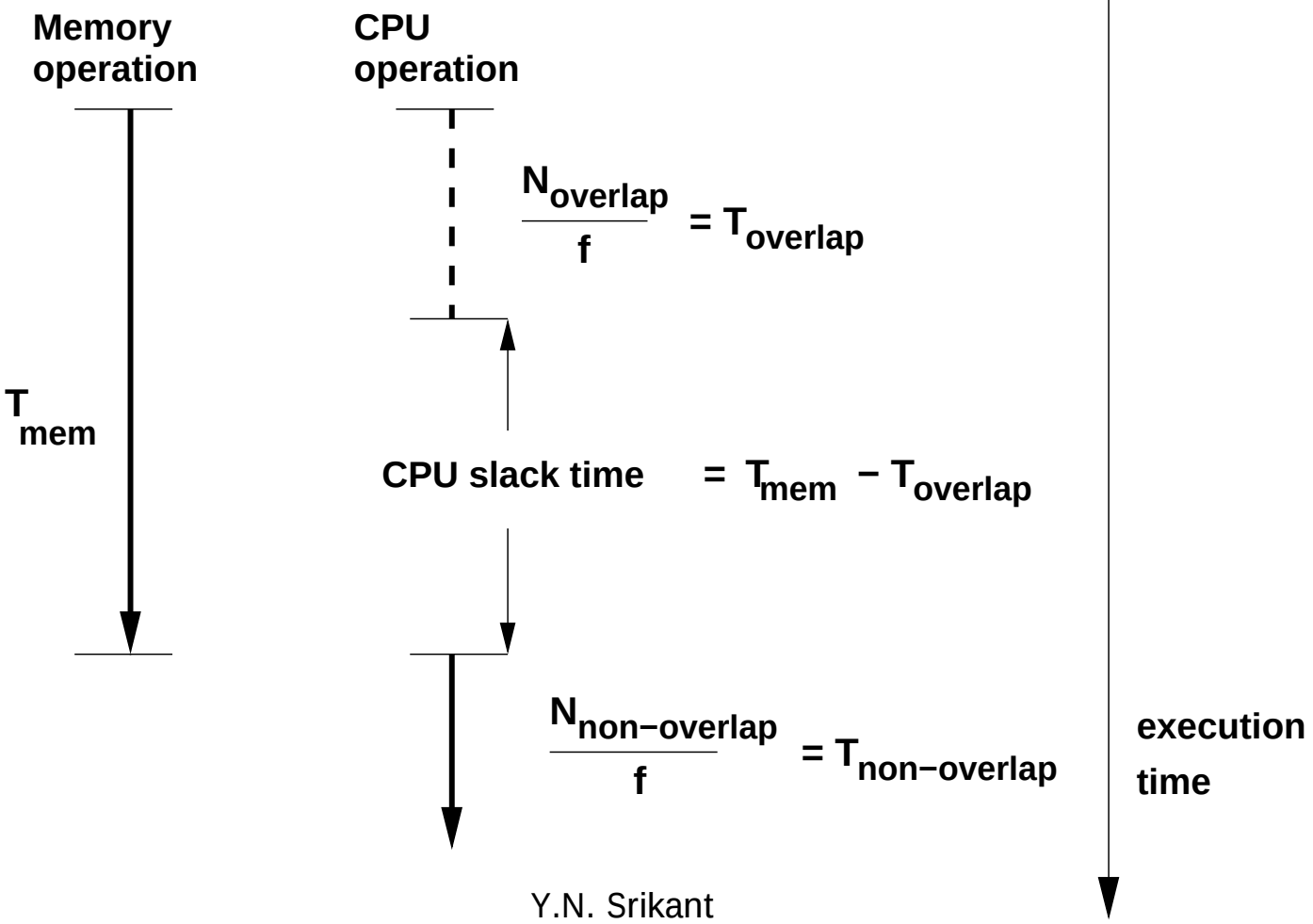
Dynamic Workload Prediction based V/F Scaling

- Requires direct intervention of OS and/or application itself
- Interval based scheme
  - consider idle time on a previous interval as a measure of processor's utilization for the next interval
  - use this to decide on the V/F settings to be used throughout the next interval
  - a moving average of previous intervals may be used to advantage

Compiler DVS – Basic Idea

- Parts of program operate at different (V,f)
- During CPU stall (awaiting completion of memory operations)
  - scale down CPU voltage and freq
  - Save energy without performance degradation
- Memory operations are assumed to be asynchronous
- 0% – 25% energy savings with 0% - 3% performance loss

Compiler DVS – CPU Slack



DVS – An Algorithm

- Partition program into "regions" based on energy consumption at different (V,f)
  - 2 regions: one at a lower frequency and the other at f_{max}
  - Introduce frequency-changing instructions at the entry and exit of the (lower freq) region
- Finding best partitions is an optimization problem
- Time for frequency change: 100 memory accesses (10-20 μs)

DVS - Regions

- Rationale: All top level statements inside a region R are executed the same number of times
- Examples
  - Loop nest
  - Call site
  - A procedure
  - Sequence of statements
  - Entire program

DVS - Constraint on size of R

$T(R, f_{max}) / T(P, f_{max}) \geq \rho$ (about 20%)

- This ensures that
  - R is never too small
  - The time to execute R is longer than a single DVS call
- Very small ρ increases time penalty
- Very large ρ decreases energy benefits

DVS – The Minimization Problem

$$\min_{R,f}\{ P_f \cdot T(R,f) + P_{fmax} \cdot T(P-R, f_{max}) + P_{trans} \cdot 2 \cdot N(R) \}$$

subject to

$$\{ T(R,f) + T(P-R, f_{max}) + T_{trans} \cdot 2 \cdot N(R)\} \leq \{(1+r) \cdot T(P, f_{max})\}$$

r : performance degradation tolerated (5%)

DVS - Implementation

- Code instrumentation for basic regions (call sites, loops, if-else)
  - to measure $T(R, f)$ accurately (using a high precision timer) and $N(R)$ at different frequencies
- P_{fmax} is either measured directly or using a simulator (*Wattch*)
- P_f is obtained from P_{fmax} using interpolation
- Combining basic regions using simple composition rules
- All the region combinations are enumerated and the optimal one is chosen

DVS during Dynamic Compilation

- Identify hot methods using simple profiling
- Compute reduction in frequency as proportional to relative CPU slack time $((T_{mem} - T_{overlap})/T_{total})$
- Use performance counters to compute the above

DVS during Dynamic Compilation

- $T_{mem} / T_{total} \approx$
  k_1(#mem_bus_transactions)/
  #µops_retired)
- $T_{overlap} / T_{total} \approx$
  k_2(#FP_INT_instructions)/
  #µops_retired)

Dynamic Voltage Loop Scheduling

- Repeatedly regroup a loop based on rotation scheduling
- Decrease the energy by DVS as much as possible within a timing constraint
- Not necessarily for memory bound programs

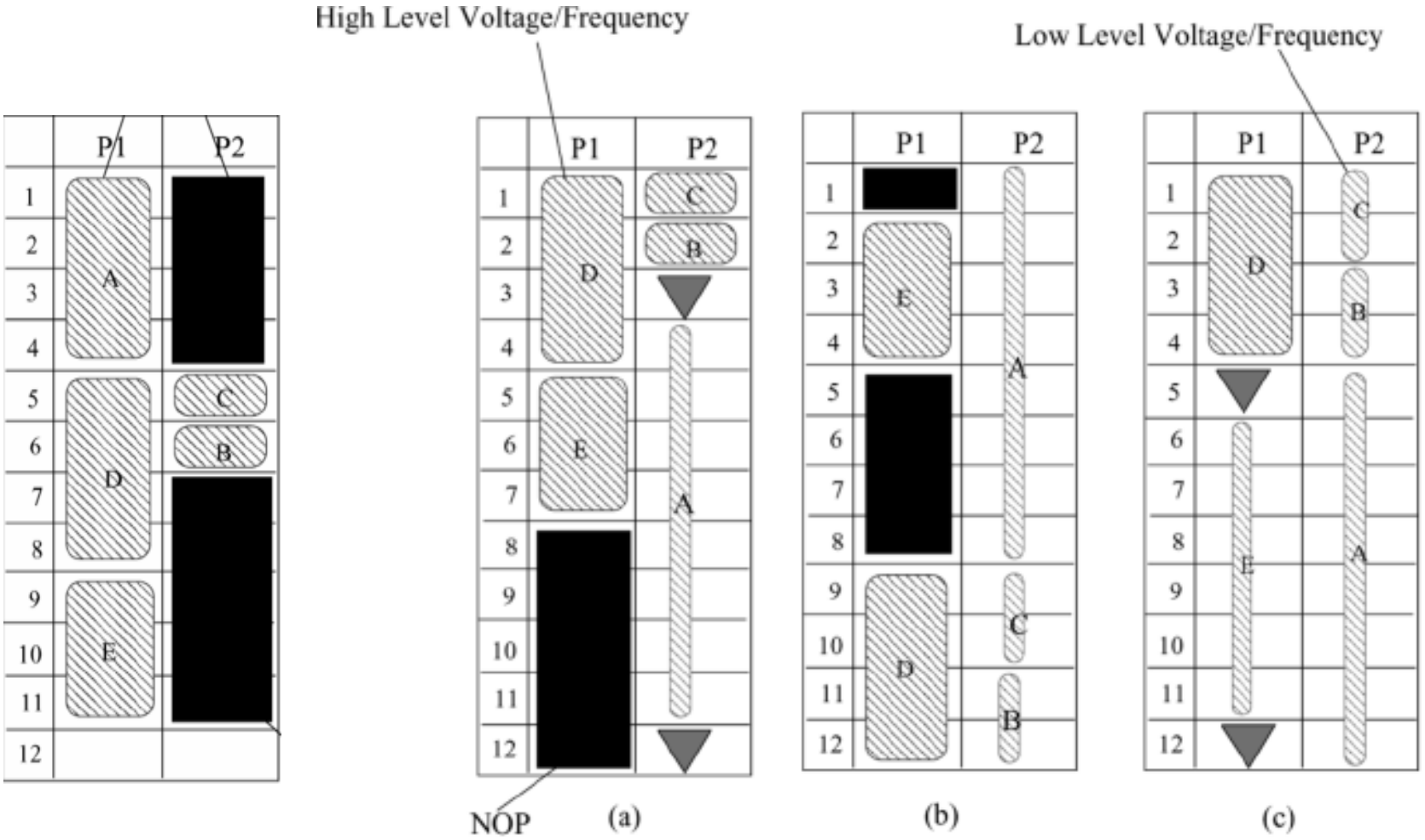
Original Loop

```
E[0]=E[−1]=E[−2]=1;
for ( i=1; i<=N; i++){
    A[i]=E[i−3]*E[i−3];
    B[i]=A[i]+1;
    C[i]=A[i]+3;
    D[i]=A[i]*A[i];
    E[i]=B[i]+C[i]+D[i];
}
```

Rotated Loop

```
E[0]=E[−1]=E[−2]=1;
A[1]=E[−2]*E[−2];                    Prologue
for ( i=1; i<=N−1; i++)
{
    B[i]=A[i]+1;
    C[i]=A[i]+3;                     Loop
    D[i]=A[i]*A[i];                  Body
    E[i]=B[i]+C[i]+D[i];
    A[i+1]=E[i−2]*E[i−2];
}
B[N]=A[N]+1;
C[N]=A[N]+3;
D[N]=A[N]*A[N];                      Epilogue
E[N]=B[N]+C[N]+D[N];
```

Figures from:Shao, et al.,
IEEE TC&S, May 2007, p445-9



Original Schedule

Schedule after first rotation

Schedule after second rotation

Schedule after third rotation

Figures from:Shao, et al., IEEE TC&S, May 2007, p445-9

Multiple Clock Domain (MCD) Processors

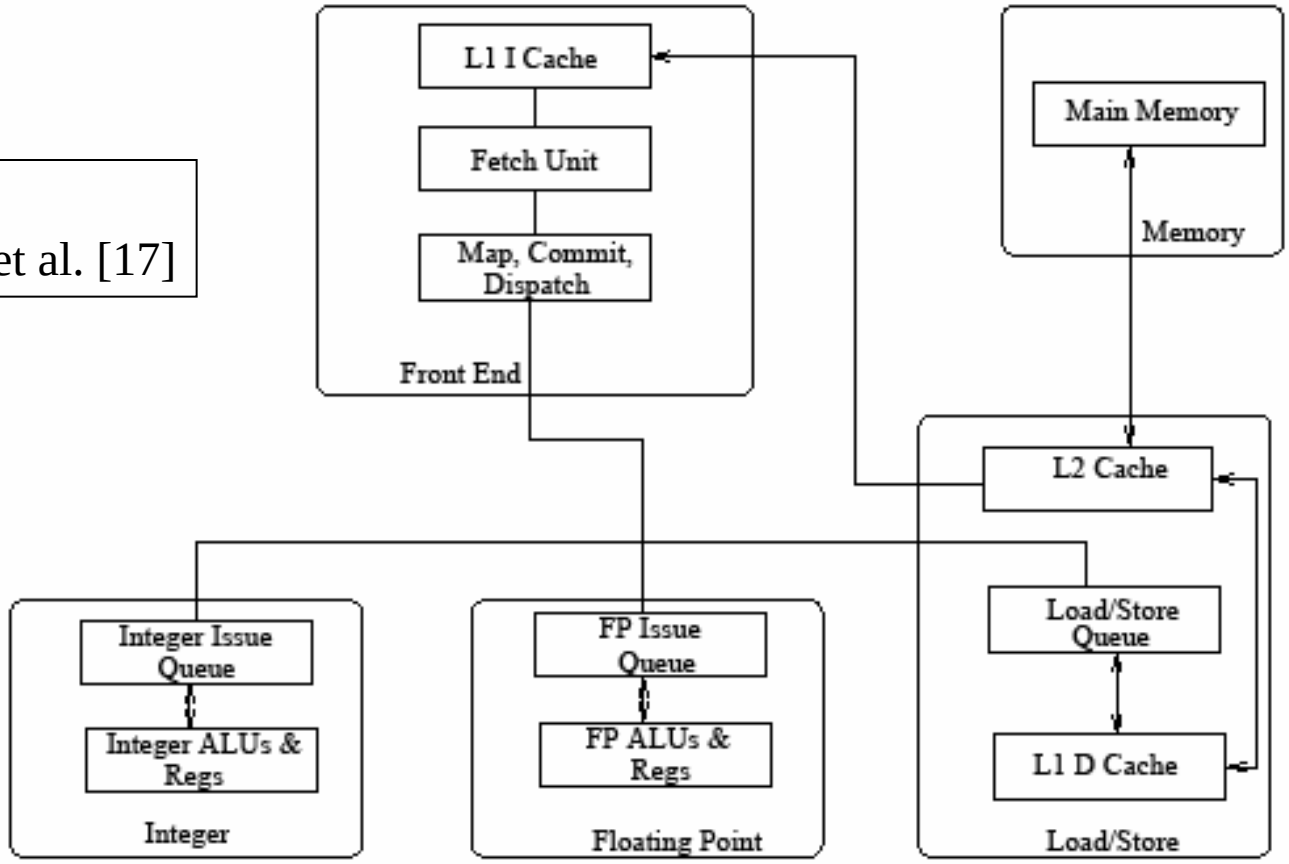
- Address difficult clock distribution and power dissipation problems
- Chip Partitioning: Each partition (domain) functions at an independent voltage and frequency
- Needs synchronization circuits for inter-domain communication

Multiple Clock Domain (MCD) Processors (contd.)

- Domains whose performance is non-critical, can operate at lower voltage and frequency: potentially less power dissipation
- Performance costs for synchronization: not very significant for out-of-order processors

An MCD Microarchitecture

From
Rangaswamy et al. [17]



Compiler based DVS for MCD Processors

- Builds a Timed Petrinet based program performance model, parameterized by microarchitectural settings and resource configurations
- Uses the model to evaluate performance impact of different frequency settings for a program region
- Chooses a low frequency setting with acceptable impact on performance, specified by the user
- Uniquely and directly estimates performance impact of a frequency setting, instead of relying on weak indicators of performance

Results

- SPEC FP: Many L2 misses; ED improvement - CDVS saves 60.39%, while meeting performance constraints; Profile-based DVS saves 33.91%
- Media Benchmarks: almost no L2 miss; ED^2 improvement of CDVS (PDVS) : 22.11 (18.34%)
- Hardware-based DVS saves less energy; relatively better in media benchmarks where queue occupancies of FP and LS domain are low

Compiler Techniques to save Static Energy in CPU

Motivation for Function Unit Voltage Gating (1)

- Leakage energy is the static dissipation energy in CPU, cache, etc.
  - FUs are in active state, but are not doing any useful work
- With 70 nm technology, leakage energy consumption will be on par with dynamic energy consumption

Motivation for Function Unit Voltage Gating (2)

- Dual-threshold domino logic with sleep mode can facilitate fast transitions between active and sleep modes without performance penalty and with moderate energy penalty
  - Can put ALU into low leakage (sleep) mode after even one cycle of idleness
- IALUs are idle for 60% of the time (on the average)

Motivation for Function Unit Voltage Gating (3)

- Pure hardware scheme (Dropsho et al)
  - has 26% energy overhead over ideal scheme (no overhead)
  - frequent transitions between active and sleep states
- A software-based scheme aids the hardware and together they save more energy with little performance loss

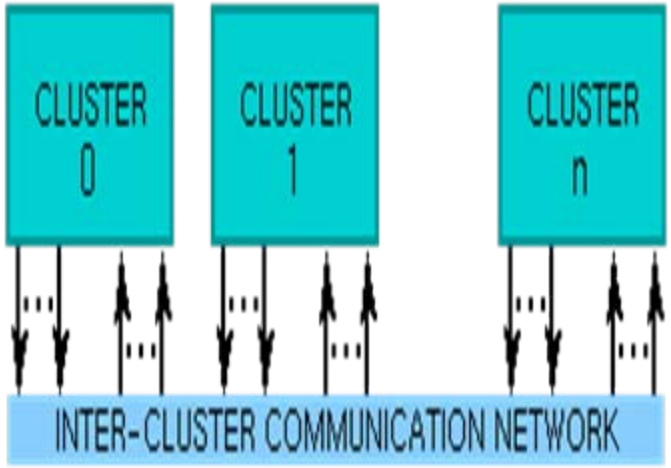
Compiler - CPU function unit voltage/clock gating

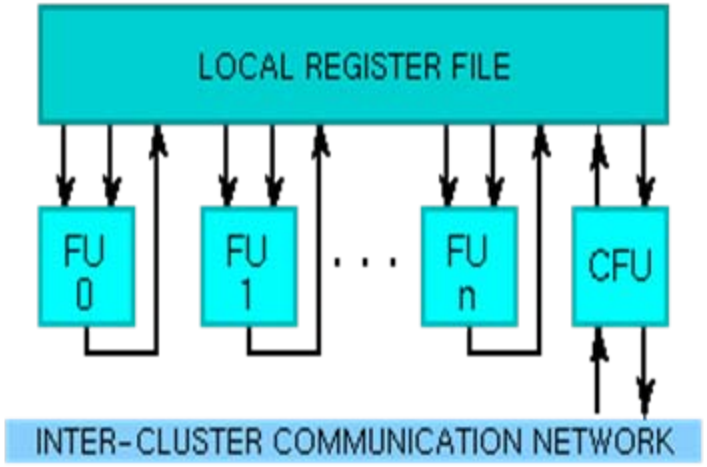
- Try to bunch instructions which use the same FUs so that "active"and "idle"periods of FUs are increased
- CPU uses supply voltage/clock gating during idle periods
- Leads to better benefits and saves transition energy

Instruction Scheduling

- Reordering instructions
  - To reduce pipeline stalls
  - To exploit instruction level parallelism
- NP-complete (with resource constraints also handled)
- Uses a DAG and is limited to basic blocks
- List scheduling with a ready queue is the most common approach

Clustered VLIW Architectures





An Individual Cluster

FU Function Unit
CFU Communication Function Unit

Energy-aware instruction scheduling

An integrated energy-aware instruction scheduling algorithm for clustered VLIW architectures:

- Reduces #transitions between active and sleep states and increases the active/idle periods
- Reduces the total energy consumption of FUs
- Generates a more balanced schedule which helps to reduce the peak power and step power

The scheduling algorithm for clustered VLIW

- Makes cluster assignment decisions during temporal scheduling
- Basic block scheduler using list scheduling
- Three main steps
  - Prioritizing the ready instructions
  - Assignment of a cluster to the selected instruction
  - Assignment of an FU to the selected instruction in the assigned target cluster

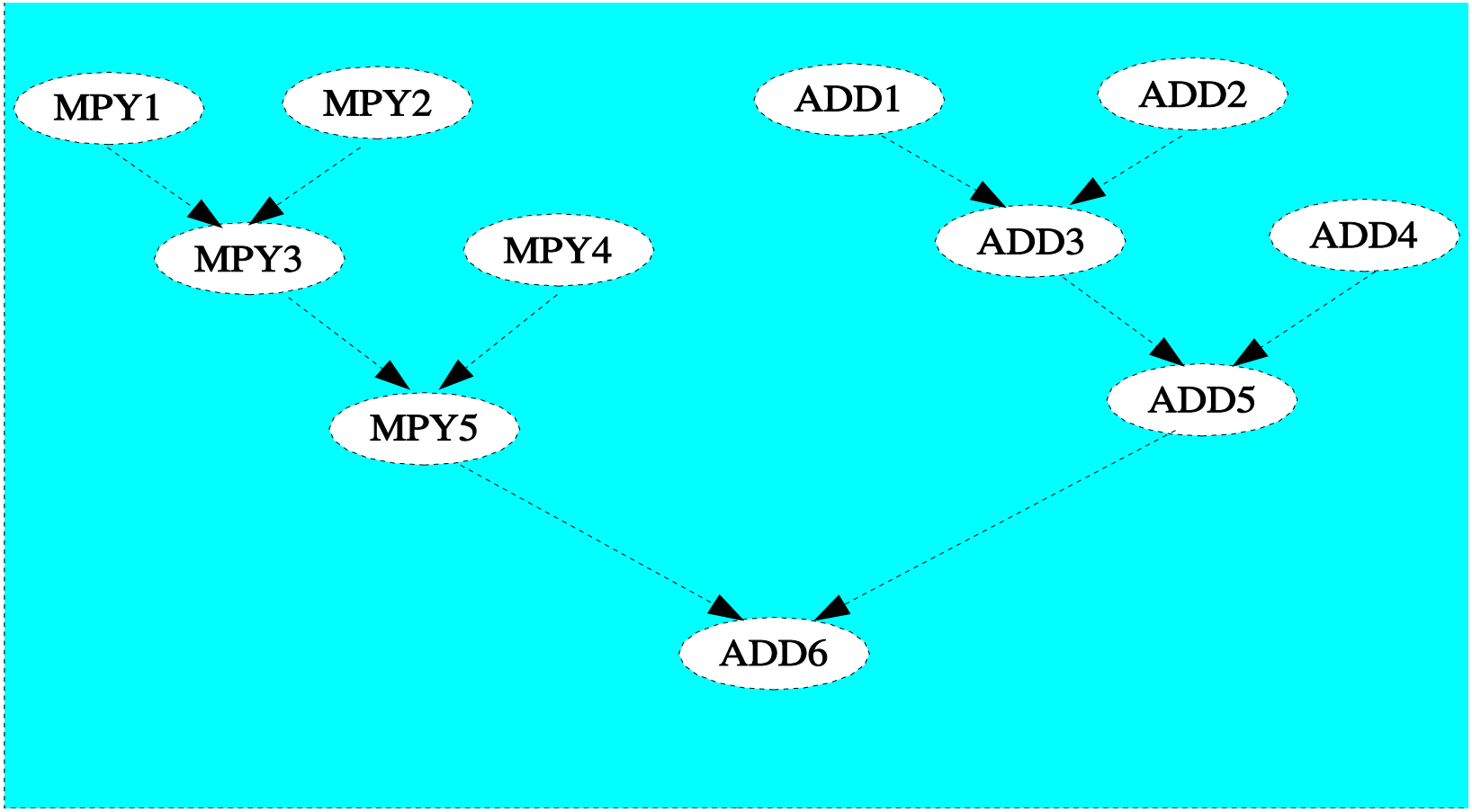
Prioritizing ready instructions

- Priority = *f* (slack, #consumers)
- Slack = Latest finish time - Earliest start time
- Slack is dynamically updated
- The higher the slack, the lesser the priority
- Choose highest priority instruction first

Cluster assignment and function unit binding

- Prefer a cluster that has an active function unit of the type needed
- Bind an active FU, if available
- Otherwise, the FU in sleep mode for a longer duration is woken up
  - only if instruction slack < threshold
  - otherwise, instruction is put back in the ready queue

Example – Dependence Graph



Example – Schedules 1 & 2

	Schedule 1
1	MPY1/M1, MPY2/M2, ADD1/A1, ADD2/A2
2	MPY4/M1, ADD4/A1, ADD3/A2
3	MPY3/M1, ADD5/A1
4	
5	MPY5/M1
6	
7	ADD6/A1
8	

	Schedule 2
1	MPY1/M1, MPY2/M2
2	MPY4/M1, ADD1/A1
3	MPY3/M1,ADD2/A1
4	ADD3/A1
5	MPY5/M1,ADD4/A1
6	ADD5/A1
7	ADD6/A1
8	

Traditional scheduler (performance-oriented)

Energy-efficient scheduler

	M1	M2	A1	A2	
Schedule 1	2	2	4	2	# transitions from low
Schedule 2	2	2	2	0	to high and vice-versa

Resource usage vectors
Schedule 1 (4,3,2,0,1,0,1,0)
Schedule 2 (2,2,2,1,2,1,1,0)

- #Transitions has reduced (energy savings)
- Cycle to cycle variation in resource usage has reduced
  - this reduces step and peak power dissipation

VLIW, one cluster, 2 MULs (2 cy latency), 2 ADDs (1 cy latency)

Example – Schedules 3 & 4

	Schedule 3	
	Cluster 1	Cluster 2
1	MPY1,ADD1	MPY2,ADD2
2	MPY4,ADD4	
3	ADD3	
4	MPY3,ADD5	
5		
6	MPY5	
7		
8	ADD6	
9		

	Schedule 4	
	Cluster 1	Cluster 2
1	MPY1	
2	MPY2	
3	MPY4,ADD1	
4	MPY3,ADD2	
5	ADD3	
6	MPY5,ADD4	
7	ADD5	
8	ADD6	
9		

Traditional scheduler (performance-oriented)

Energy-efficient scheduler

	M1	M2	A1	A2	
Schedule 3	2	2	4	2	# transitions from low
Schedule 4	2	0	2	0	to high and vice-versa

Resource usage vectors
Schedule 3 (4,2,1,2,0,1,0,1)
Schedule 4 (1,1,2,2,1,2,1,0)

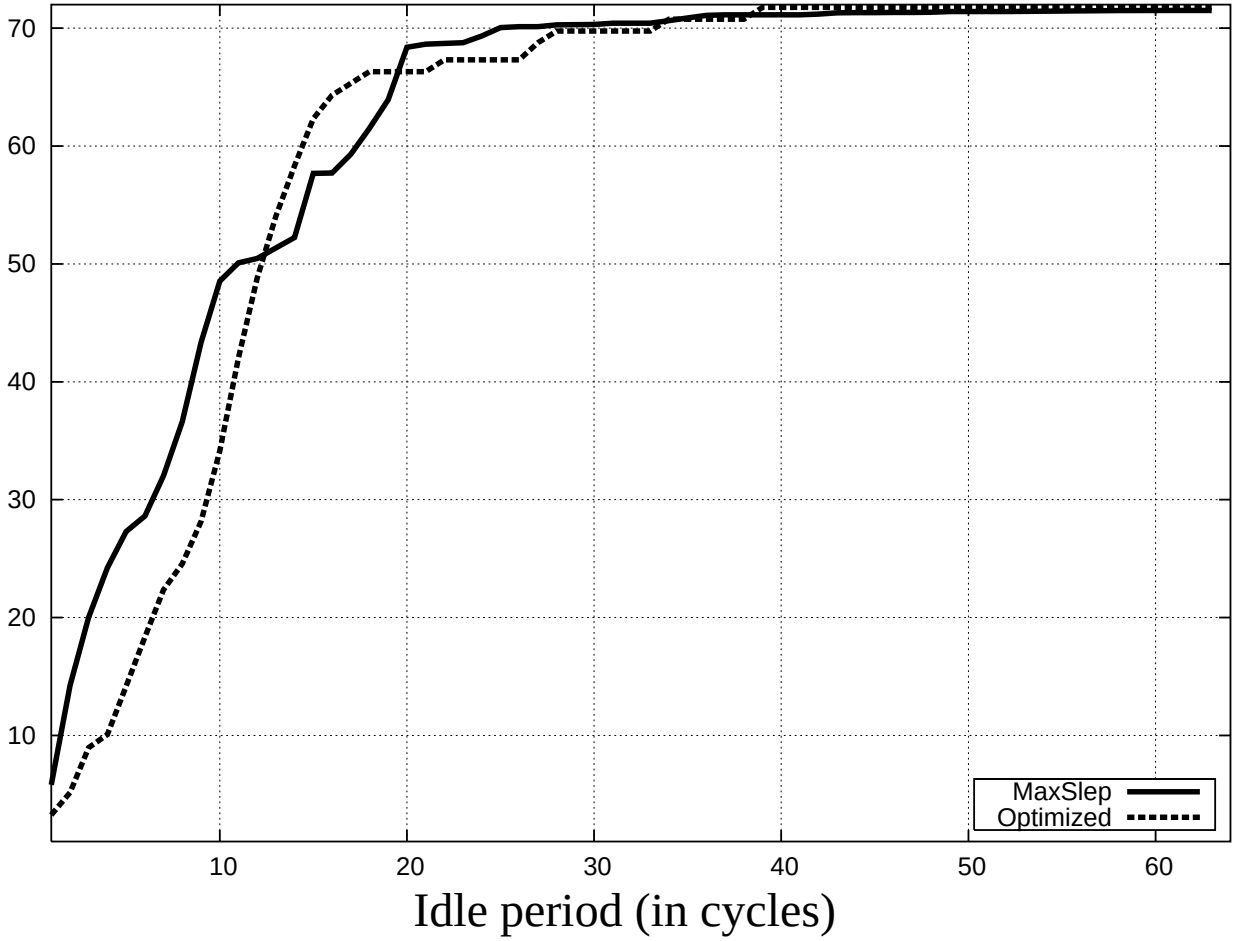
In schedule 3, ADD3 (MPY3) is scheduled in cycle 3 (4) because a cycle is needed to transfer the result of ADD2 (MPY2) from cluster 2 to cluster 1

VLIW, two clusters, 1 MUL (2 cy latency), 1 ADD (1 cy latency) in each cluster

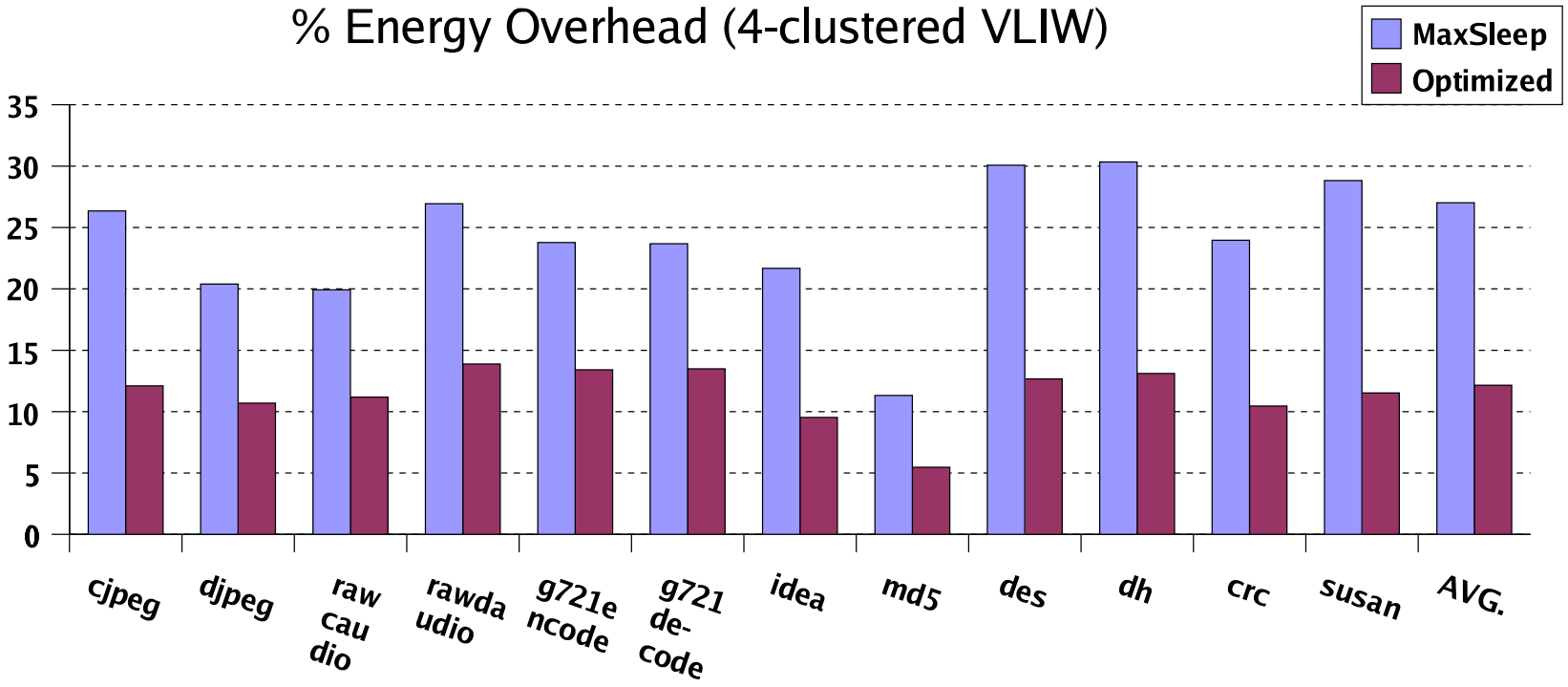
Results

- Comparison with hardware-only schemes
- #Transitions reduce on the average by 58.29% (4-clusters)
  - Reduction in the #transitions is directly proportional to the available slack
- Average reduction in energy overhead is 16.92% (4 clusters)
- Only 34% of the overall #idle periods are now smaller than 10 cycles (48% in Maxsleep)

Cumulative Distribution of Idle Periods (in cycles)



Energy Overhead (4-clusters) w.r.t No-overhead Scheme



Saving Communication Energy in CPU

CPU – Optimization of Communication Resources

- Low swing signalling over buses
  - Saves power in capacitive charging
  - Reduces reliability (more errors)
  - Needs coding to increase redundancy
- Data Encoding
  - Minimize average switching activity over communication channel
- Bus design – hierarchical seems better
- Heterogeneous buses are possible

Compiler – Reduction of switching on the instruction bus

- Hamming distance between two consecutive instructions is minimized
- When instructions are fetched, switching on the I-bus reduces
- Instruction scheduling techniques can be used here

Heterogeneous Interconnects

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

Exploiting Heterogeneous Interconnects

- Selectively mapping communication to the appropriate interconnect
  - urgent communications to
    - low latency (high energy) path
  - non-urgent communications to
    - high latency (low energy) path
  - identify urgent comm. using comm. slack (60.88% of comm. have 3-cycle slack)
  - Increase in execution time is 1.11% and reduction in comm. energy is 39% (both for a 4-cluster processor)

INTACTE: An Interconnect Area, Delay, and Energy Estimation Tool

- Interconnects can consume power equiv. to one core, area equiv. to three cores, and delay can account for 0.5 of L2 cache access time
  - Can be a major source of performance bottleneck
- We present an interconnect modeling tool
  - Enables co-design of interconnects with other architectural components

INTACTE – What?

- Interconnect microarchitecture exploration tool to estimate
  - Delay
  - Power
- Technology, area, clock frequency and latency are inputs
  - for point to point interconnect
- Analogous to CACTI

INTACTE – How?

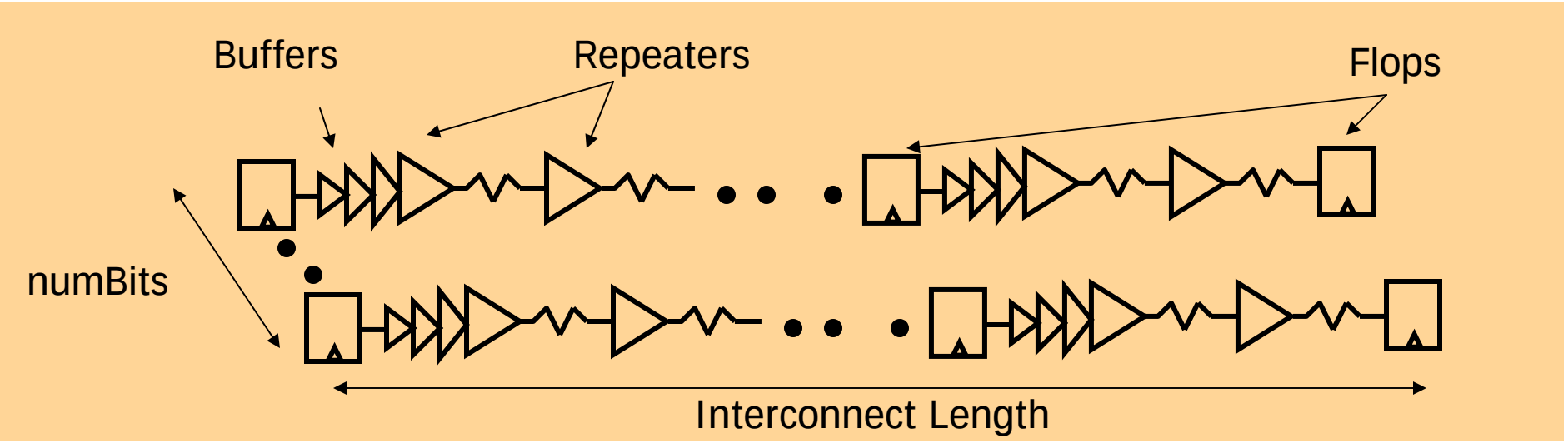
- Solves an optimization problem of minimizing power by finding the optimal values for
  - Wire width
  - Wire spacing
  - Repeater size
  - Repeater spacing

INTACTE - More

- Additional design variables - can be either constraints or determined by the tool
  - Area
  - Pipelining
- Voltage Scaling Support
  - Tool optimizes power and delay for nominal (Maximum) supply
  - Power and Delay numbers reported
    - for 32 different voltage levels separated by 15 mV from the nominal values

INTACTE Tool description (1)

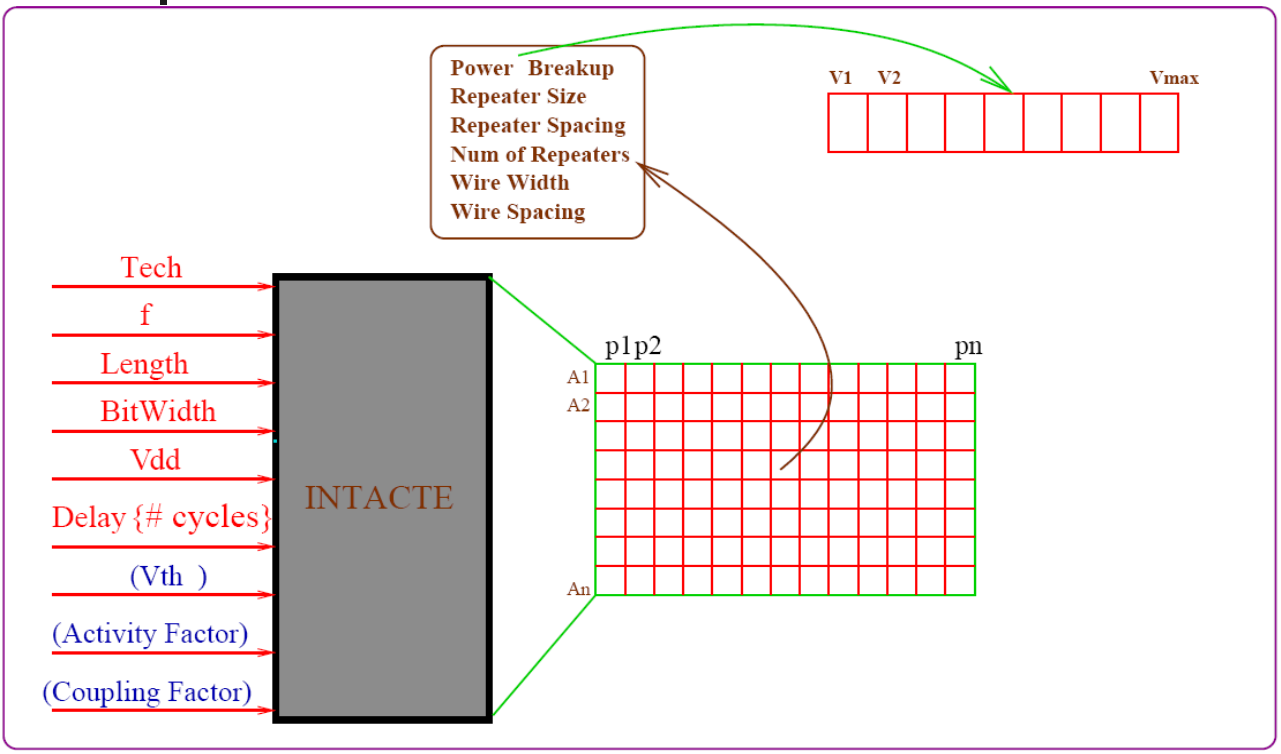
- The tool models the interconnect as consisting of a set of identical, equal length pipeline stages
- Each stage starts with a Flop driving a repeater through a set of buffers followed by equally spaced wire-repeater sections.
- All parameters for the model are taken from detailed HSPICE simulations and ITRS



INTACTE Tool description(2)

- The parameters related to the flops, repeaters, wires and buffers are pre-computed for 4 different technology nodes (90, 65, 45 and 32nm) and 32 different supply voltages.
- For each iteration of optimization, the tool computes the power and delay for each wire-repeater section.
- These values are multiplied by #repeaters and degree of pipelining and added to the pipelining overhead to get the overall power and delay numbers.
- This reduces the size of the search space

Block Diagram of INTACTE



- Tool Inputs
  - Technology
    - 90,65, 45 & 32nm
  - Clock frequency
  - Length of interconnect
  - Bit width
  - Supply
  - Delay (in cycles)
  - Activity Factor
  - Coupling Factor
- Tool Outputs
  - Power, Delay versus
    - Area, Pipelining, Supply

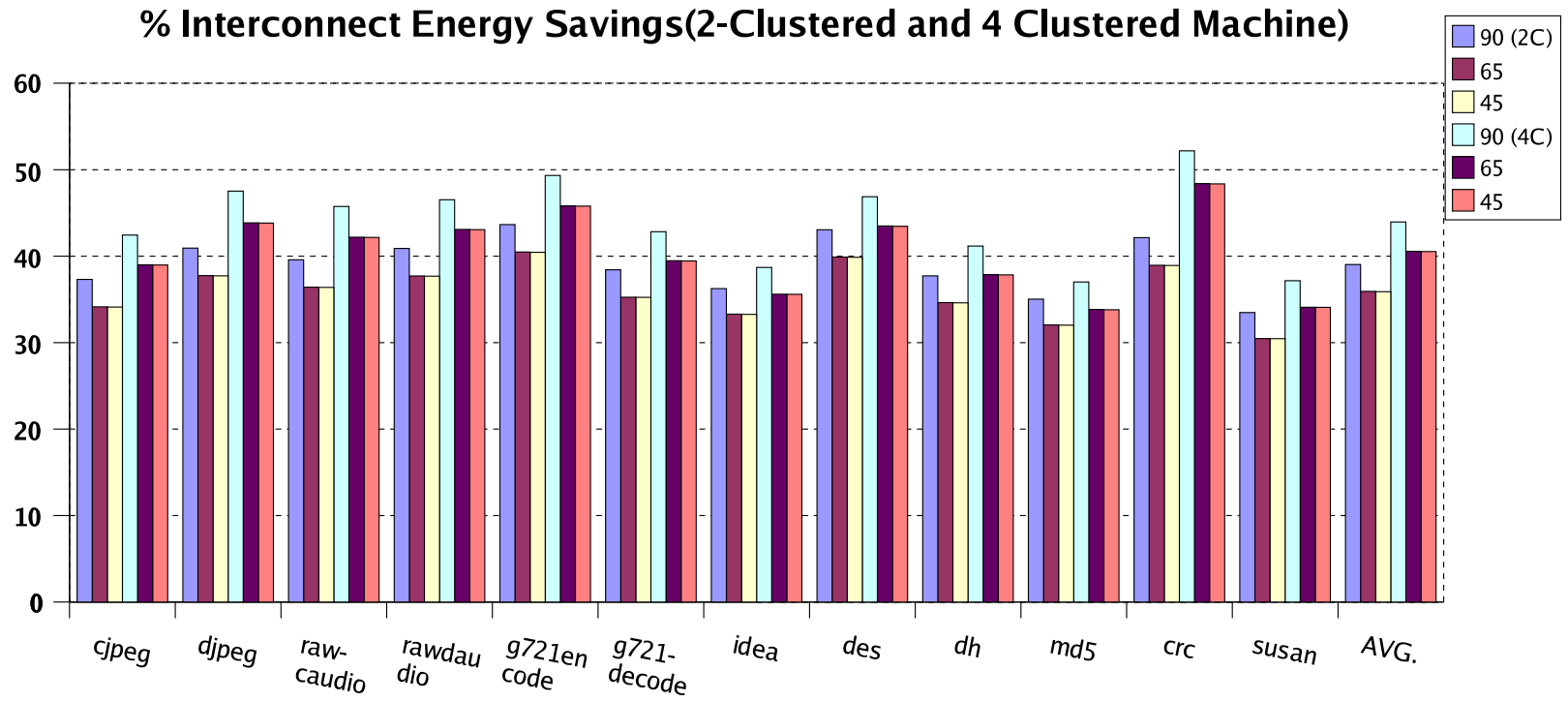
Experimental Results

- Demonstrate the accuracy of the tool
  - various trends in interconnect power and performance have been exhibited
  - detailed HSPICE simulations have been carried out to validate the results.

Architectural Tradeoff Evaluation

- Architectural tradeoffs in having two heterogeneous wires can be evaluated using our tool
- Architect provides length, no. of bits, target technology, operating voltage, and delay estimates
- Tool provides a set of possible interconnect design options to choose from

Interconnect Energy Savings



Energy-Aware Memory

Memory

- Hierarchical memory design better
  - *L0 – L1 – L2 – DRAM Bank*
  - Sizes of lower level memories are smaller
  - Energy consumption per access
    - *L0: 150 mW, L1: 300 mW, L2: 700 mW, DRAM Bank: 12.71 W for a burst transaction*
    - Smaller memories need less power per access
- Memory could consume 50% more power than processors

Memory models and control

- Each RDRAM (Rambus DRAM) chip can be activated separately
- Supports *standby*, *nap*, and *powerdown* modes
- The Controller controls switching between modes based on performance and permitted *slowdown*
- Hard disks can also be modelled and controlled similarly
  - Variable speed drives are available too

Drowsy cache memory (1)

- Cache memory lines may have
  - Clock gating
  - Supply voltage gating and scaling
- Cache line gating may be at circuit level or at program level
- Switch off cache lines when not in use for a certain number of cycles
  - this could be a fixed scheme or an adaptive scheme
  - useful for both I and D caches

Drowsy cache memory (2)

- Compiler analysis is possible
  - identify critical data in a program and place these in a hot cache (non-drowsy)
  - place non-critical data in a drowsy cache
  - Needs simple modifications to the architecture to accommodate extra information

Vertical Cache Partitioning: A Filter Cache

- A very small cache placed in front of L1 data cache
- Most data will be accessed from filter cache
- L1 D-cache will be placed in standby mode
- Good for applications with small working sets (e.g., streaming media applications)

Vertical Cache Partitioning: Pre-decoded Buffers and Loop Buffers

- Pre-decoded buffers
  - Store recently used instructions in an instruction buffer in decoded form
  - Eliminates dynamic energy spent in fetching and decoding
- Loop buffers
  - Hold time-critical loop bodies in small dedicated buffers

Horizontal Cache Partitioning: Region-based Cache

- Two small additional 2 KB L1 D-caches
  - one for stack and one for global data
  - dedicated decoding circuitry detects data access to the appropriate cache
- Substantial gain in dynamic energy consumption for streaming media application with negligible impact on 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%
- WCET can be performed accurately

A Simple SPM allocation algorithm – (1)

- Single SPM, single memory
- Memory segments
  - each global variable is a segment (locals are not considered for SPM allocation)
  - each function (completely)
- Formulated as a knapsack problem and solved using integer programming

A Simple SPM allocation algorithm – (2)

Maximize $G = \Sigma (x_i * Eg_i)$

subject to $\Sigma (x_i * s_i) \leq K$, where,

Eg_i = energy gain resulting from allocation of segment i to SPM

s_i = size of memory segment i

K = total size of SPM

x_i = 1, if segment i is mapped to SPM

= 0, otherwise

A Simple SPM allocation algorithm – (3)

$Eg_i = (E_m - E_s) * n_i$

E_m (E_s) = energy for one access to main memory (SPM)

n_i = #memory accesses to segment i (obtained by static analysis or profiling)

Extensions to the simple SPM allocation algorithm

- Extensions to include a hierarchy of SPMs, basic blocks, stack frames, etc., are not hard
- Dynamic overlaying of memory segments in SPM, based on life times of segments is a non-trivial extension

Summary

- Energy optimizations are essential in tomorrow's electronic systems
- Energy optimization of a computer system should be carried out at all possible levels
  - Algorithm level, Micro-architecture level, Compiler level, Operating system level, and Network level.
- Energy optimizations should be considered at the design stage itself, and not as an after-thought

Research Issues

- DVS for memory banks and caches
- DVS for speculative execution architectures
- DVS for multi-core architectures
- Cache reconfigurations
  - change associativity and size based on program analysis
- Cache bank remapping for tiled architectures
- Energy models for hardware transaction memory
- Memory bank control
- Energy-efficient OS kernel design

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. Compiler-Directed Frequency and Voltage Scaling for a Multiple Clock Domain Microarchitecture, R. Arun, Rahul Nagpal, and Y.N. Srikant, ACM Conference on Computing Frontiers, 2008, pp 209-218.
18. Power: A First-Class Architectural Design Constraint, T.Mudge, IEEE Computer April 2001.

Thank You
Questions?