Contents

1 Motivation

1.1 Jevons Paradox

Efficiency gain causes an increase of demand that might outweight the efficiency gain. Explains why battery life improved only by a factor of 10 while transmission speed et al. increased by factors of millions in recent decades [L6-23].

2 Principles of Energy-Aware Computing Systems

2.1 NMOS / CMOS Power Demand

Ptotal = Pdynamic + Pshort-circuit + Pstatic Ptotal = (Cload · fp · A · Vdd2) + (Ishort · Vdd) + (Ileak · Vdd)

2.2 Dark Silicon

smaller transistors + unchanged chip area -> too high power density -> all cores can't be used simultaneously

counter-measures:

- switch off cores
- reduce clock speed

- reschedule activities

require energy-aware software

2.3 Basic Metrics

2.3.1 Power

suitable metric to

- stay within power supply / cooling facility constraints (limit peak power)
- predict head dissipation (everage/peak power)

2.3.2 Energy

suitable metric to

- predict battery life
- predict energy bill

2.3.3 Composite

$P_{avg} * t^{1..4}$

- power-delay product (PDP): energy demand per switching event
- energy-delay product (EDP): energy demand and performance, misleading with dynamic voltage scaling (DVS)
  - [Horowitz 1994 Digital, Figure 1] Almost the same EDP but lower performance when Voltage is smaller.
- ED^{2P} for DVS
- ED^{3P} for high-performance

2.4 Horowitz 1994: Low-Power Digital Design

2.4.1 Trade Speed for Power

- Supply Voltage Scaling: Time it takes to charge the load capacitor by the threshhold voltage increases.

- Transistor Sizing: Reducing the gate size increases the capacitance (-> time it takes to charge it by the threshhold voltage decreases)
- Adiabatic Circuits, mostly unused because
  - simple circuit design gets complicated
  - double the area of CMOS & slower

2.4.2 Conserve Energy

- Technology Scaling: Size reduction by y decreases the EDP by y^4.
  - Problem: Vth has to be scaled too -> limited by leakage current -> only y^2
- Transition Reduction
  - keep inputs stable (no longer effective because of leakage currents)
  - idle modes

2.4.3 Parallelism

Idee: Delay reduction by N, energy per op constant -> EDP reduction by N -> trade speed for power!

Real World: Fabrication technology more important because issuing multiple instruction is costly.

2.4.4 Redefine the Task

Do less operations! Improves EDP by x^2.

3 Energy Demand Analysis

3.1 Physical Methods

3.1.1 Shunts

- requires constant DC voltage
- measure voltage drop V over a precisely known small resistor R
- calculate electric current I by applying Ohm's law I=V/R

1. Problems

- sampling -> miss volatage peaks -> unrecognized power demand
- inaccessible wires (SoC peripherals, ball grid arrays)
- custom setup -> not portable
- host and measurement system

3.2 Logic Methods

3.2.1 Performance Counters

Events demand a certain amount of energy. E.g. retired instructions or L2 cache references.

1. Problems

   - accuracy depends on model quality (hardware platform and usage scenario specific)
   - overhead-prone because systems monitors itself

3.3 Tiwari 1994: Power Analysis of Embedded Software: A First Step Towards Software Power Minimization

3.3.1 Contributions

- present methology to create an instruction level power model
- use it to [vision: supplied for every processor by vendor]
  - verify that software meets the specified power constraints
  - perform software power optimization

3.3.2 Method

Measure average current consumed while short instruction sequences are executed in a loop.

3.3.3 CPU Model

```
Program Energy Cost = sum(instruction_base_energy_costs) + inter_instruction_circuit_st
```

1. Base Energy Cost

   n pipeline stages can be ignored, because

$E_{cycle} = E1_{I1} + E1_{I2} + ... + En_{In} = \sum_j^n Ej_{I1} = E_{ins}$ weil $I1 = I2 = ... = In$.

When an instruction causes pipeline stalls, this adds to their energy cost.

Caveats:

- loops must be large enough, otherwise branch statement has a large effect
- loops should not be too large, otherwise cache misses occur
- addressing modes have a significant effect
- immediate values and memory alignment have a limited effect (3-5%)

2. Inter-Instruction Effects

   (a) Circuit State

   - varying instructions consume more power than expected from the base-cost average
     - thesis: base-costs were obtained in *a context of least change*
     - e.g. no switching on data/address lines
     - **effect limited** (max. 5-7%) contrary to popular belief
       * most instruction have much in common on this CISC (e.g. instruction prefetch, pipeline control, clocks)
       * may be different on RISC?
   - consider costs of pairs of instructions -> <1% error
     - cicuit state overhead between 5-30mA, most frequently 15mA
     - allows simple model: `program_cost = sum(base_cost + 15mA)`

   (b) Resource Constraints triggering Stall Cycles

   Method:

   i. determine average stall cycle cost by isolating it (e.g. 250mA for prefetch buffer stall)
   ii. estimate number of stalls through static analysis of program code
   iii. add to program cost

(c) Cache Misses

Consumption fo 216mA per cycle measured using special scenarios. Add to base costs by estimating cach miss rate of the program run using a cache simulator.

3.3.4 Memory System Model

- cost is approx. 2 times larger for page misses
- hard to estimate using static analysis

3.3.5 Evaluation: Software Power Optimization

- **Instructin Reordering** -> 2%
- **Code Generation** -> 33-40%
  - prefer **Registers over Memory**
  - reduce **Running Time**

4 Energy Managment

4.1 Energy Resource Scenarios

- Finite, e.g. phones with batteries. Maximize operating time.
- Revolving, e.g. servers. Adhere to operating (e.g. cooling) constraints.

4.2 Control Theory and Practice

1. System processes load (e.g. schedules application).
2. Measurement device supplies values (e.g. performance counters) to controller.
3. Controller performs action (e.g. reduce frequency) to influence System (e.g. reduce power demand).

4.2.1 Control Methods

1. Non-Blocking

   (a) Dynamic Voltage and Frequency Scaling

      i. Reduce frequency to lengthen execution time (linear). Reduces energy/operation indirectly by reducing the temperature.

      ii. Reduce voltage (as much as the frequency allows) to reduce power consumption (quadratic).

   Energy consumed by CMOS while executing: $E = P{*}t$ $P_{dynamic} = \alpha * C * f * V^2$ $P_{static} = V * I_{leak}$ $P = P_{dynamic} + P_{static}$
   $P_{idle} <= P$
   To minimze global energy consumption, minimize $E_{active} = (P - P_{idle}) * t$. The smaller P_{idle}, the larger the incentive to reduce t and not P.
   DVFS is a way to trade t increase for P decrease.

      i. without C-States
      $P_{idle} := P_{slow} = P_{static} + P_{dynamic,slow}$
      If $P = P_{slow}$ factor is zero and t does not matter. Idle time represents wasting of energy. Lengthen execution time to minimize idle time.

         A. PAST
         Idea: Run as slow as possible while still not causing contention.

      ii. with C-States
      $P_{idle} := P_{sleep} < P_{static} + P_{dynamic,slow}$

         A. Race-to-Sleep: Maximum Frequency
         $f := f_{max}$
         $t = t_{min}$ $E_{dynamic} = E_{dynamic,max}$ $E_{static} = E_{static,min}$

         B. Weissel 2002: Process Cruise Control
         Constraints and assertions:
         - $t <= t_{limit}$
         - $P_{idle} := P_{idle,pcc}$

         Goal:
         - Find f so that $E = \min E_{active} = \min (P - P_{idle}) * t$
         - Find the f that fullfills that using performance counters.

Abstract: Continuously monitor running applications using performance counters to determine their optimal clock frequency for a given maximum runtime and a give idle power consumption. Trade in slight performance losses (<10%) for subtantial energy savings.

C. Measurement Setup
Shunt resistor with 3MHz sampling. CPU runs at 333-733MHz. Dynamic power consumption (i.e. current draw minus idle draw) is measured. This times the execution time is the "dynamic energy consumption" of an application.
Here $P_{min} := P_{idle}$. But why not P_{sleep}?

D. Idea
Dynamic power consumption of CPU and DRAM depends on rate of
- executed instructions
- branches
- data cache references
- memory requests

E. Frequency Scaling
Power consumption scales linearly with frequency. Only the performance of CPU- and cache intensive applications scales linearly with frequency.
Energy efficiency of CPU- and cache intensive applications is always good and best at high frequencies because of the constant overhead of kernel activities (e.g. timer interrupt).
Energy efficiency of memory intensive applications is best at low speeds because it avoids CPU stalls.

F. Implementation
Energy specific characteristics of
- a single thread change slowly
- concurrently running threads change quickly (**Criticism:** this is not considered in the eval)

PCC loop

G. run application

H. read event rates

- memory requests per clock cycle: the higher, the more savings from frequency reduction (i.e. choose smaller CPU speed)
- instructions per clock cycle: the higher, the more performance loss from frequency reduction (i.e. choose higher CPU speed)

I. apply model: policy matrices defined by userspace
- processor specific
- determines expected performance loss tolerated

J. set CPU speed, goto 1

K. Evaluation

L. Homogeneous Apps
`find|grep`, `gzip`, `djpeg` and `factor`.

M. Run at every possible speed and measure energy consumption to determine optimal frequency for a given allowd penalty.

N. Apply PCC and check whether it chooses the optimal frequency for the given penalty.
- Optimal frequency is usually reached after 1-2 iterations.
- The target performance loss may be exceeded.

O. Synthetic Inhomogeneous App
- Includes memory bound and CPU bound workloads.
- Performance loss / energy savings depend on phase.

P. ghostscript
- Different optimal frequencies depending on input file.

Q. Dynamic Voltage Scaling
Large potential. Not only avoid stall cycles but also decrease the amount of energy used in every cycle. Math suggest at 2-8x the savings.

R. Crawl-to-Sleep: Minimum Frequency
\$f := f_{min}
$t = t_{max}$ $E_{dynamic} = E_{dynamic,min}$ $E_{static} = E_{static,max}$

iii. with Power-Off
$P_{idle} := 0$
Minimize $E = P * t$. Creates picture from execises.

(b) **TODO** Adaptive Voltage and Frequency Scaling

(c) **TODO** Running Average Power Limit

2. Blocking

   (a) **TODO** C-States

   Reduce power consumption of CPU cores when idle.

5 Components and Subsystems

5.1 Memory-aware Scheduling

Combining of compute-bound an memory-bound processes on core pairs sharing the L2 cache. Choose highest frequency if a compute-bound process is running.

Problems

- potential priority inversion
- requires knowledge of memory/cache hierarchy

5.2 Load/Store and Execute Streams

Decouple load/store from execute operation stream. Allows CPU to group operations that execute best at high/low speeds. Requires compiler support and synchronization points between the two streams.

5.3 Thread Assignment to Heterogeneous Cores

Combine multiple processors that have different power/performance characteritics.

Big cores are inefficient when operated a small frequencies. Smaller cores with fewer transistors (i.e. caches) have a smaller leakage current, they can perform the same work using less power.

5.4 DVFS for Memory

5.5 Gupta 2012: Uncore

Abstract: Evaluation of the design of heterogeneous multicore processors regarding energy-efficiency.

- small cores: IO bound or background tasks
- large cores: CPU bound or foreground tasks

5.5.1 Hardware

1. Uncore

   Consists of

   - Last Level Cache (usually L3)
   - Integrated Memory Controllers
   - On-Chip Interconnect

   Package C-State is only entered when C-State of every contained core is larger or equal.

2. HMPs

   Big cores allow entering a C-state faster. Smaller cores may use less energy to execute the task, but may keep the uncore active for longer using more energy in total.

   Suggestion: A scalable uncore with frewer memory channels, controllers and less cache.

5.5.2 Client Workloads

The following appications were evaluated:

- Bursts: browers
- Substained low/high: hardware accelerated media playback, video encoding
- Multi-threaded: parallel compression

5.5.3 Evaluation

Run every workload on only big/small cores (note: why didn't they simply use different processors?).

1. Hardware Model

   Core power depends on idle time and power consumption while active. Active power consumption depends on capacitance (different for big/small core), voltage and frequency.

   Uncore power is constant when active and scales to some degree with the L3 cache access rate.

   A scalable uncore is asserted to potentially consume half the power of a fixed uncore.

2. Results

   (a) Idle Residency

   Application keeping a core or the integrated GPU busy rarely allow for package sleep states, therefore preventing the uncore from sleeping.

   (b) Performance

   IO bound and GPU workload performance is rarely influenced by smaller cores.

   (c) Energy

   - Core-only energy usage was better on the small core for every application.
   - Total energy usage was not better for every app with the fixed uncore. With the scalable uncore it was better for every app.
   - GPU workloads would have greatest benefit from a scalable uncore.

6 Cyber-Physical Systems

6.1 TailEnder

Reduce the energy consumption of networking hardware by optimizing the time spent in tail states.

6.2 LARN

Cyber physical systems such as

- industrial facilities
- medical applications

require networks with predictable low latency and energy demand (two things which are usually in coflict and require trade offs). To do this tools have to be developed that analyze the delay from the hardware and the operating system (e.g. X-Lap).

6.3 Agarwal 2010: Building

Abstract: Use presence-sensors to control heating, ventilation, and air conditioning (HVAC) systems.

Thermal HVAC systems consume a significant portion of total building energy and to-date are not as well managed as lighning or electrical HVAC systems. The designed system is low-cost and combines a magnetic reed switch door sensor and a passive infrared sensor.

Using the occupancy information the potential savings are simulated. These are about 10-15% (which is significant for HVAC).

7 Cluster Systems

High-performance nodes vs low-power nodes are better at different areas of the power/performance space.

Power proportionality can save power because datacenters are usually over provisioned.

Small nodes can be energy efficient for I/O bound tasks.

Jeonghwan Choi et al.: thermal-aware task scheduling

- core hopping
- task deferral: reschedule hot-running tasks

7.1 Krioukov 2010: NapSAC

Abstract: Make a heterogeneous web cluster power proportional, allowing it to exploit low server utilization to save energy.

- server system

  - high peak performance/watt
  - no suspend-to-RAM, power-off/up in 60s
- mobile platform
  - lower peak performance/watt
  - suspend-to-RAM in 2.4s
  - not powerful enough for certain high-performance database servers
  - uses more physical space
  - requires more networking infrastructure
- heterogeneous cluster
  - outperforms both regarding energy savings vs. violations

For 30-160 req/s it makes more sense to power on 1-5 Atom Mobiles than to power on 1 Nehalem Server.

8 System Software

1. accounting
   - map resource demand by process to energy demand proportionally
   - track energy demand (use appropriate metric) using models (performance counters, time used) or measurements
2. allocate energy
3. admistering

Problem: conflict of interest with scheduler (priority inversion, data dependencies).

8.1 Requester-Aware Power Reduction

Idea: Track requests to device accesses and by which process they are generated. Then reorder requests to reduce the overhead (e.g. wake/sleep).

8.2 Currentcy and ECOSystem

1 currentcy unit := 0.01 mJ

ECOSystem schedules processes based on available currentcy (total / user-defined battery lifetime), not CPU time. Processes may accumulate up to 10x the usual currentcy per epoch.

8.2.1 Heng Zeng et al.; Currentcy: A Unifying Abstraction for Expressing Energy Management Policies

8.2.2 Heng Zeng et al.; ECOSystem: managing energy as a first class operating system resource

8.3 Cinder Operating System

Arjun Roy et al.; Energy Management in Mobile Devices with the Cinder Operating System

8.4 Linux Energy-Aware Scheduling

The scheduler, `cpuidle` and `cpufreq` have traditionally been isolated from each other. Possible consequences

- The scheduler tries to balance load, while cpufreq tries to keep the frequency as low as possible, thereby increasing load on individual CPUs. They may work agains each other.
- The scheduler used to place a task on any CPU that is currently in a C-State, but it would be better to place it on a CPU that is in a lower C-State thereby minimizing wakeup time.

The basic assumption that is violated here, is that every CPU is equal to the scheduler. This is wrong for cpus at lower frequencies, different idle states and especially, big.LITTLE.

In big.LITTLE a task might be placed on either a small or a large CPU regarding performance.

- Placing it on the small CPU would require a frequency increase for the little cluster, thereby wasting energy.
- Placing it on the big CPU might require more power at it is not that efficient at a low frequency.

Which is chose is based on the energy model.

8.5 Neugebauer 2001: Nemesis OS

Abstract: When accounting capabilites for other resources exist, energy can be accounted to individual processes. Applications can offer different modes of operations to save power (e.g. not upload/download media not but do it later when on WiFi). To communicate something like this to the applications, they are "charged" for the operations they perform.

8.5.1 Resource Accounting in Nemesis

Account energy usage of applications in proportion to device usage. Determined by having a single multiplexing-point for each device:

1. CPU
    - processes reserve time slices
    - Earliest Deadline First (EDF) scheduler
2. Memory
    - request ranges of virtual memory
    - guaranteed to be backed by a specified number of physical pages
    - managment happens in the process
3. Device Drivers
    - interrrupt ACK happens in kernel
    - multiplexing and out-of-band managment function in device driver process
    - processing (e.g. network stack processing) happens in individual processes

    E.g. Display: draw screen area in userspace using shared library, push out pixels in kernel.

8.5.2 Energy Measurement

Get possible power states of each device using ACPI, then measure how system battery power draw is influenced when the state is changed using ACPI control methods. -> Estimate of device power draw for each state.

State transitions may consume considerable power (e.g. disks), therefore measure them as well.

Problems:

- Device consumption may also depend on processing load and not only the state (e.g. brightness level). Processing load has to be generated by someone (ususally the CPU) therefor disturbing the measurement.
- Is the smart battery interface accurate enough to measure power consumption during disk state transitions?
- Unrelated events such as cache misses may consume considerable power.

8.5.3 Energy Managment

Principles

- basic resource consumption is free
- if a resource is congested, apps responsible for it are charged proportionally

Apps maximize utility minus costs for a resource x. System tries to maximize total utility minus total "costs".

Costs are for example dropped packets in networks. For energy it is the draw above the target draw (the draw that allows the system to reach the requested battery charge lifetime).

Apps are only charged for a interval, if the draw in that interval exceeded the average target discharge rate.

Not Discussed: How is the "charge" communicated to the application? Using interrupts or should they poll?

Criticism: Apps using constant power are not charged much althought they contribute a lot to energy usage. Maybe something like the CFS for energy would be better (every app has it's share of energy, if it is used up it can no longer run). Maybe also the battery should always be considered congested and only uncongested when the phone is charging.

Apps that are not programmed to adapt, can be slowed down by throttling their inputs. Otherwise the user must decide whether the app is worth it.

9 System Activities and Energy Demand

9.1 Cross-Layer Considerations

9.1.1 Application Code

- Optimizations at Compile Time
- Tracing and Profiling at Runtime

9.1.2 System Software

- Control of DVFS and Sleep Modes

9.1.3 Hardware

- Device Specific Power Saving Features
- Implementation of DVFS and Sleep Modes

10 Energy-Aware Programming

10.1 HEAL

See "SEEP: Exploiting Symbolic Execution for Energy-Aware Programming".

10.2 ROAM

See "Proactive Energy-Aware Programming with PEEK".

10.3 Pereira 2017: Programming Languages

Abstract: Analyse runtime, memory usage and energy consumption of programming languages using the same algorithm implemented in each language. Which language should be used when (regarding only these criteria).

10.3.1 Benchmarks

10 problem solutions implemented (to be as fast as possible) in 27 different programming languages.

RQ1: Can we analyze the energy efficiency of programming languages -> Yes, by comparing how the programs written in them perform.

10.3.2 Measurement

Measure

- CPU and DRAM energy consumption using jRAPL
- execution time
- peak memory usage using `time`

10.3.3 Analysis

RQs:

- Is the faster language always the most energy efficient? -> No, because the usage scenario matters more. No language is consistently the fastest and for slower languages energy and time are rarely correlated.
- How does memory usage relate to energy consumption? -> Peak memory usage rarely correlates with DRAM energy consumption. Future: Is it correlated to continuous memory usage?
- Is one language always the fastest, the most energy efficient and also the most memory efficient? -> For energy+time it is C, for both and memory it is not possible.

11 Infrastructure

11.1 HVAC et al.

- power usage effectiveness (PUE) = total energy demand / energy demand of computing systems
  - refinements to account for mixed us of renewable and non-renewable energy or reuse of secondary energy

11.1.1 Temperature-Aware Workload Placement

Reduce hot-spots and therfore cooling requirements by up to 50% by distributing workloads.

See Cluster Systems Lecture.

11.1.2 Building Operating System Services

Most energy demand is caused by buildings. Reduce it by providing OSs for commercial buildings, e.g. with occupancy-driven energy managment.

11.1.3 Runtime System for Heterogeneous HPC Clusters

Exploit variable power pricing to consume energy when it's cheap and save energy when it is expensive.

11.2 Fan 2007: Power Infrastructure Building Costs

Abstract: Non-recurring cost of facilities delivering power rival recurring energy costs of datacenters. Datacenters should be opearated at maximum capacity to avoid costs to provide power capacity that is not needed. This is hard because power ratings are overly conservative and consumption varies with load.

11.2.1 Problem

Large Datacenters have

- one-time building costs for the power infrastructure
- ongoing electricity costs for the servers

If operated at 85% of peak capacity for ten years, the building cost is still higher than the electricity cost.

11.2.2 Reasons for Under-Utilization

- New facilities have space for more.
- Internal fragementation at server, rack or PDU level.
- Nameplate ratings of servers are too high. +40% in comparison to maximum power draw achieved.
- Variable load.
- It is unlikely that many servers are active at the same time.

11.2.3 Model for Power Usage Estimation

Use global CPU utilization as it is sufficiently accurate (error $< 1\%$) at PDU level when accounting for a fixed offset (caused e.g. by network hardware which consumes mostly costand power).

-> Aggregate CPU utilization data from many servers and mapreduce to produce power consumption estimates.

11.2.4 Results for Different Workloads

- Websearch: many requests, much data processing, usage depends on time of day
  - the larger the group of machines, the more narrow the observed aggregated operating range
- Webmail: disk IO, usage depends on time of day
  - smaller variance compared to websearch
- Mapreduce: process terabytes of data, batch jobs not correlated much to time of day
  - largest difference between rack/pdu/cluster
- Combinded Cluster: dynamic range of the mix is narrower than that of any individual workload
- Real Datacenter: even narrower dynamic range

11.2.5 Power Capping

Observation: There are few timer intervals in which close to the highest power is drawn at cluster, PDU and rack level.

Power capping at 1% of the time allows for up to 20% more machines with the same power budget. However, the more diverse the workload the smaller the potential for power capping.

Provides a safety valve in cases where the power usage estimation model is wrong or where workloads change unexpectedly.

11.2.6 DVFS

Not much potential to host more machines but some potential to save energy.

11.2.7 Idle Power Consumption

Observation: Idle power is usually >50% of peak power. If idle power was 10%, peak power usage and energy consumption would be greatly reduced, especially in a real datacenter.