Lecture 7: Duty cycling

MO801/MC972 – Energy-Aware Computing
Lucas Wanner – IC/Unicamp
lucas@ic.unicamp.br
www.lucaswanner.com/eac
Agenda

- Revision: variability and dark silicon
- Duty cycling
  - Concept and basic formulation
  - Variable power consumption
  - Duty cycling OS
Revision: variability

• Definition: “systematic and random variations in process, supply voltage and temperature” [Borkar, 2003]
  • Manufacturing beyond 90nm becomes probabilistic instead of deterministic
  • Transistors with different channel length and threshold voltage
• Expanded definition: Variations between identically specified components due to manufacturing (process, vendors), environment (voltage, temperature), and aging
• Effects of variability
  • Performance characteristics, e.g. clock speed
  • Reliability, e.g. device lifetime, error characteristics, gradual degradation
  • **Power**: Active (switching) and Sleep (leakage) power varies between parts with identical specifications
Revision: variability

• To ensure effective use by software, we need accurate characterization (of performance, power).

• Variability imposes a limit on how accurate the models can get to
  • Mean error ~20% + 12% due to variability for 34% overall error in Nehalem 45nm CPUs
  • 15-20% variation across 22 DIMMs
  • 20-24% read, 40-67% write variation in Flash
  • Rooted in inherent non-observability of power states.

• New regime of hardware/software operation
  • Machines built from parts with variations in performance, power and reliability
  • Machines that incorporate sensing circuits
  • Machines w/ interfaces to change ongoing computation & structures
  • New machine models: QOS or Relaxed Reliability parts

Source: McCullough, UCSD
Adapted from Gupta, Variability Expedition
Revision: Dennard scaling

\[ S = 1.4 \]

\[ S^3 \approx 2.8 \]

S = 1.4x

Faster Transistors

S = 1.4x

Lower Capacitance

Scale down by \( S = 1.4x \)

\[ S^2 = 2x \]

\[ S = 1.4x \]

Scale down by \( S = 1.4x \)

\[ S^3 \approx 2.8 \]

Leakage issues prevent voltage scaling!

Adapted from Taylor, UCSD
The utilization wall that causes dark silicon is the solution to dark silicon; it is merely industrial misconception blank, useless, or unused silicon; it’s just energy-

timation wall. In exponentially darker silicon over time, these gains must be offset by a correction.

The end result is a shortfall of energy-efficiency Check”. Follow-up work (see the sidebar “Is Dark Silicon Real? A restated, the true new potential remains inaccessible. More positively stated, the true new potential remains inaccessible. More positively stated, the true new potential remains inaccessible. More positively stated, the true new potential remains inaccessible.

Although the utilization wall is based on a first-order model that simplifies many factors, it has proved to be an effective tool for designers to gain intuition about the future of Moore’s law is a 1.4 improvement per generation, which could be realized with leaving core count (for example, 32/22 nm). In both Dennard and post-Dennard regime, the total chip utilization for a fixed power budget drops by 2/3, or 2.8 with each process generation. The result is an improvement that causes dark silicon, we could do even better.

In exponentially with e, these gains must be offset by a correction.

Let’s clear up a few misconceptions before advancing a single process generation would allow enough transistors to increase core count by 2,3,4. A modern processor might consist of a single core. Under these conditions, the impact of a single process generation would amount to a 1.4 improvement per generation, which could be realized with leaving core count (for example, 32/22 nm). In both Dennard and post-Dennard regime, the total chip utilization for a fixed power budget drops by 2/3, or 2.8 with each process generation. The result is an improvement that causes dark silicon, we could do even better.

Although the utilization wall is based on a first-order model that simplifies many factors, it has proved to be an effective tool for designers to gain intuition about the future of Moore’s law is a 1.4 improvement per generation, which could be realized with leaving core count (for example, 32/22 nm). In both Dennard and post-Dennard regime, the total chip utilization for a fixed power budget drops by 2/3, or 2.8 with each process generation. The result is an improvement that causes dark silicon, we could do even better.

Table 1 shows the derivation of the utilization wall from equation (1) to equation (3). Let the transistor property be 

<table>
<thead>
<tr>
<th>Transistor property</th>
<th>Dennard</th>
<th>Post-Dennard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
<td>(S^2)</td>
<td>(S^2)</td>
</tr>
<tr>
<td>Frequency</td>
<td>(S)</td>
<td>(S)</td>
</tr>
<tr>
<td>Capacitance</td>
<td>(1/S)</td>
<td>(1/S)</td>
</tr>
<tr>
<td>(V_{DD}^2)</td>
<td>(1/S^2)</td>
<td>1</td>
</tr>
<tr>
<td>(\Delta) Power = (\Delta) QFCV^2</td>
<td>1</td>
<td>(S^2)</td>
</tr>
<tr>
<td>(\Delta) Utilization = 1/Power</td>
<td>1</td>
<td>(1/S^2)</td>
</tr>
</tbody>
</table>

Revision: post-Dennard scaling
Revision: approaches to handling Dark Silicon

- Dim silicon
  - Heavily underclocked parts of the chips
  - Inherently dark areas, e.g. caches
  - Turbo-boost: increase clock for short bursts of time
  - Near-threshold voltage computing (NVT)
    - Higher susceptibility to PVT, leakage
    - Temporal dimness: e.g. switching between cores in Big.Little designs
- Specialization: Accelerators, specialized cores
- Parallel with human brain
  - Very dark, low duty cycle, low voltage operation
Duty cycling

\[ \Delta = c/p \]

↑ \( \Delta \) ⇒ ↑Quality ↑Energy

c↑ p↓
Duty cycle rate

• How can you determine duty cycle as a function of $P_A$, $P_S$, $E$, $L$?
Determining the lifetime for a given duty cycle

- Average power used by an application
  - PA: Active Power
  - PS: Sleep Power
  - \( \Delta \): Duty Cycle Rate

\[
P_{average} = \Delta P_A + (1 - \Delta) P_S
\]

- Energy storage and lifetime
  - E: Battery capacity in Watt-Hours
  - L: Lifetime in hours

\[
L = \frac{E}{P_{average}}
\]
Determining the duty cycle rate for a target lifetime

- Average power used by an application
  - PA: Active Power
  - PS: Sleep Power
  - $\Delta$: Duty Cycle Rate

\[ P_{average} = \Delta P_A + (1 - \Delta) P_S \]

- Maximum average power available for an application
  - E: Battery capacity in Watt-Hours
  - L: Lifetime in hours

\[ P_{max} = \frac{E}{L} \]

- How to find the allowable duty cycle rate?
Determining the duty cycle rate for a target lifetime

- Duty cycle the device at the maximum allowable power consumption

\[
P_{\text{average}} = P_{\text{max}}
\]

\[
\Delta P_A + (1 - \Delta)P_S = P_{\text{max}}
\]

\[
\Delta(P_A - P_S) + P_S = P_{\text{max}}
\]

\[
\Delta = \frac{P_{\text{max}} - P_S}{P_A - P_S}
\]

\[
\Delta = \frac{E}{L} - P_S
\]

\[
\Delta = \frac{E}{L - P_S}
\]
Feasible Duty Cycle

How to determine duty cycle when $P_A$, $P_S$ vary with instance and temperature?

$$\langle c, p \rangle = f (P_A, P_S, E, L)$$
Implications of Variation for Duty Cycling

• Scenario: deploy a network of sensors. All nodes have identical batteries, and should have identical lifetimes
  • If active and sleep power are constant for all instances, duty cycle can be obtained trivially from
  \[
  \Delta = \frac{E}{L} - \frac{P_S}{P_A - P_S}
  \]
• Recall power variation in ARM Cortex M3
  • More than 8x in Sleep mode at room temperature
  • Around 10% in Active mode
• Uniform duty cycle across the network will be suboptimal
Duty cycle based on datasheet spec

• Use $P_A$, $P_S$ from datasheet

will not meet lifetime

will leave energy untapped
Duty Cycle based on Worst-Case Power

- Use worst case $P_A$, $P_S$ across all instances and target temperature

![Graph showing power consumption vs temperature]  

- all the nodes will leave energy untapped
Implications of Variation for Duty Cycling

- Active Mode: 48 MHz
- Sampling Task: 10 s
- Battery: 2xAA (5.4 A-h)
- Room Temperature

P2 can acquire 70% more data than P3.
Implications of Variation for Duty Cycling

Temperature + Instance performance improvement = 80%!

P1 can acquire 51% more data than P3

Active Mode: 48 MHz
Sampling Task: 10 s
Battery: 2xAA (5.4 A-h)
Lifetime: 20000 hours
Variability-Aware Duty Cycling

- Instance dependent Duty Cycle

\[ \Delta = \frac{E}{L} - P_S(i) \]

- \( P_A(i) \) and \( P_S(i) \) are instance-dependent active and sleep power
- Assumes constant temperature
- Picking an arbitrary point in the DC vs temperature curve is suboptimal
- Can we do better if we know something about temperature in advance?

25% variation in DC in a single instance due to temperature
Coping with Temperature-Dependent Variation

- If we knew the future: deploying a sensor network in Death Valley, CA (2009)

\[ ~30F \text{ Diurnal Variation} \]

\[ ~50F \text{ Seasonal Variation} \]
Coping with Temperature-Dependent Variation

- If we knew the future: deploying a sensor network in Death Valley, CA (2009)
  - Annual temperature variation of ~80F
  - Picking an arbitrary point in the DC vs temperature curve is suboptimal
- Assume there is perfect knowledge about future temperature
  - Temperature as a function of time: $T(t)$
  - Power as a function of instance and temperature: $P_A(i, T)$ and $P_S(i, T)$
  - Power as a function of instance and time: $P_A(i, T(t))$ and $P_S(i, T(t))$
- How could you define duty cycle for each instance?
Instance and Temperature-Dependent Duty Cycle

\[
P_A^\mu (i) = \frac{\sum_{t=0}^{L} P_A(i, T(t))}{L}
\]

\[
P_S^\mu (i) = \frac{\sum_{t=0}^{L} P_S(i, T(t))}{L}
\]

\[
\Delta(i) = \frac{E}{L} - \frac{P_S^\mu (i)}{P_A^\mu (i) - P_S^\mu (i)}
\]
Relaxing the temperature knowledge assumption

- Having temperature as a function of time $T(t)$ is not realistic
  - We can barely predict temperature for the next few days
  - Temperature distribution is easier to predict, can be learned over time

Figure by John L. Daly, data from NASA Goddard Institute for Space Studies
Relaxing the temperature knowledge assumption

- From temperature as a function of time $T(t)$ to frequency of temperature $f(T)$
Relaxing the temperature knowledge assumption

• From temperature as a function of time $T(t)$ to frequency of temperature $f(T)$
  • Power as a function of instance and temperature: $PA(i, T)$ and $PS(i, T)$
  • Temperature as a frequency distribution $f(T)$
  • Discretized temperature bins, e.g. one bin for each degree

\[
P_A^\mu (i) = \frac{\sum_{t=0}^{L} P_A(i, T(t))}{L} \quad \rightarrow \quad P_A^\mu (i) = \sum_{T=T_{\text{min}}}^{T_{\text{max}}} P_A(i, T) \times f(T)
\]
Variable Duty Cycle

• Operating each instance at a constant duty cycle may not be optimal
  • “Energy Cost” of remaining active at any time is determined by the difference between active and sleep mode power
  • If $P_A(i, T) - P_S(i, T)$ changes with temperature $T$, “cost of activity” will be different for different temperatures

• Duty cycles for each temperature that maximize total active time can be found with a linear program
Variable Duty Cycle

Maximize \[
\sum_{T=T_{\text{min}}}^{T_{\text{max}}} DCT \cdot f_T
\]

s.t. \[
\sum_{T=T_{\text{min}}}^{T_{\text{max}}} f_T \cdot (P_A(T) \cdot DCT + P_S(T) \cdot (1 - DCT)) \leq \frac{E}{L}
\]

\[
DC_{\text{min}} \leq DCT \leq DC_{\text{max}}, \quad T_{\text{min}} \leq T \leq T_{\text{max}}
\]
Programming duty-cycled systems

while(1) {
    do_something(duration);
    sleep(time);
}

Active Power ($P_A$)  Sleep Power ($P_S$)  Energy ($E$)

Lifetime ($L$)
Reactive Duty Cycle

- Duty cycle may also be adapted dynamically, based on resource availability
- Typical adaptive strategy: adjust workload to resource availability
  - More resources available: higher quality of service
  - Imprecise computation (EDF)

\[
\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \\
\sum_{i=1}^{n} \frac{C_i + O_i}{T_i} \leq 1 \\
\chi = \frac{\sum_{i=1}^{n} \left( \frac{C_i + O_i}{T_i} \right) - 1}{\sum_{i=1}^{n} \left( \frac{O_i}{T_i} \right)}
\]

- Tasks are divided into mandatory and optional parts. If there is sufficient processor time, run all optional parts, else, discard a fraction
- Same principle can be applied using energy as a resource
Reactive Duty Cycle

- Duty Cycle can also be determined in a reactive fashion
  - At every decision point, estimate remaining available energy (battery capacity)
  - Analyze energy delta from time t-1 to current time t
  - Project expected lifetime from energy delta and remaining capacity
    - If there is an energy surplus, increase duty cycle
    - If there is an energy deficit, decrease duty cycle
  - One potential model:

\[
DC_t = \frac{E_t \cdot DC_{t-1}}{(E_t - E_{t-1}) \cdot (L - t)}
\]

- Assumptions
  - Remaining battery capacity can be easily and accurately estimated
    - May be true for “smart” batteries, but not in general
  - Energy delta will remain constant for a given duty cycle
    - Not true with temperature-dependent variability
Determining $\Delta$ for each instance

Objective: maximize \textbf{active time} for each instance subject to energy capacity and lifetime
Knobs for $\Delta$ control in VaRTOS

for *knob* do  // computation time
    sleep (constant - *knob*)  // period

knob: app variable shared with OS

↑ knob value $\Rightarrow$ ↑ $\Delta$, ↑ quality
Sample task: adjusting computation time

/* Task 1's quality is improved by extending a for loop (like ADC samples, etc.) */
static void vExampleTask1( void *pvParameters )
{
    portTickType xLastExecutionTime = xTaskGetTickCount();

    for( ;; )
    {
        /* Enforce task frequency */
        vTaskDelayUntil( &xLastExecutionTime, TASK1_DELAY );

        volatile unsigned long i, j, dummyVal;
        for( i=0; i<task1_knob; i++){
            dummyVal = 0;
            for( j=0; j<1000; j++){
                dummyVal += (((dummyVal+5)%3)*3)/2;
            }
        }
        dummyVal = 0;
    }
}
Sample task: adjusting Activation Frequency

/* Task 2's quality is improved by increasing task frequency (like sending radio messages, etc.) */

static void vExampleTask2( void *pvParameters )
{
    portTickType xLastExecutionTime = xTaskGetTickCount();

    for( ;; )
    {
        /* Enforce task frequency */
        vTaskDelayUntil( &xLastExecutionTime, 500/(task2_knob*0.1) );

        task_body();
    }
}
Knobs for $\Delta$ control in VaRTOS

xTaskCreate(..., &task_knob, min, max, priority);
Δ control in VaRTOS

1) Requirements

**App:** knobs, lifetime, temperature profile

**Hardware:** power, temperature

Rough histogram

2) Model Training

\[ T \rightarrow PA, PS \]

knob ↔ time

40 points: 2.5% error

3) Optimization

Maximize Δ

Assign knob values

LP + Greedy Opt.
Greedy optimization of knob values

Global $\Delta$

Global Utility

active

sleep

Task 1

Task 2

Task 3
Recap: Choices in determining $\Delta$

- **Quality**: Underestimated (Datasheet), Optimal (Variability-Aware), Sub-Optimal, Guard-banded (worst case)
- **Lifetime**: Infeasible
Results: lifetime reduction with datasheet spec $\Delta$

Average: 55 days
Results: energy untapped by worst-case \( \Delta \)

Average: 63%

Remaining battery (%)

Processor Copy

P1 P2 P3 P4 P5 P6 P7 P8 P9 P10

Lifetime: 1 year, Battery: 5400 mAh
Temperature: Stovepipe Wells, CA, 2009
Results: improvement over worst-case $\Delta$

Average: 22x

Average for multiple temperature profiles: 6x

Lifetime: 1 year, Battery: 5400 mAh
Temperature: Stovepipe Wells, CA, 2009
VaRTOS vs. Oracle

Utility vs. Oracle (%)

Best
Nominal
Worst

Temperature

Best
Nominal
Worst

Utility vs. Oracle (%)

80 84 88 92 96 100

Lucas Wanner – IC/Unicamp

Energy-Aware Computing
Duty cycling IoT devices

• Duty cycle must be able to detect event of interest

- DC fails to capture events
- DC tailored to event duration
Multiple low-power modes, wakeup latencies

- Wakeup latency vs. power tradeoff
  - Devices typically use (close to) full power during transitions
- Typical laptop
  - Sleep mode (wake up on keyboard, LAN)
  - Hibernation mode (state dump/restore, power off)
  - Independent duty cycling of peripherals (disk, wireless, etc.)
- Typical embedded processor: NXP LPC13xx Cortex M series
  - Sleep mode: clock gated, state preserved, peripherals active
  - Deep sleep mode: clock gated, state preserved, analog peripherals off
  - Deep power down mode: power gated, limited sources of wakeup
Moving to and from low power states

- Processor: set up sleep mode, halt, and wait for instruction
- Example: ARM Cortex
  - Setup sleep mode by writing to specific registers
  - Setup an interrupt source (e.g. timer, push button)
    - Available interrupt sources depend on sleep mode
  - Wait for interrupt (WFI)
- General-purpose: Advanced Configuration and Power Interface (ACPI)
  - D-States, C-States (more about this later in the course)
- Historic curiosity: look up the ”HCF” instruction
  - Halt and catch fire
Summary

• Duty cycle
  • Fraction of time in which the system is active
  • Average power consumption is a function of power in active mode, power in sleep (inactive) mode, and duty cycle

• Duty cycle $\leftrightarrow$ Lifetime

• Trivially determined for known power consumption
  • Complicated by variations in power
    • Uniform duty cycle is suboptimal
    • Can be determined or learned for individual instances, power profiles
  • Complicated by transition latencies
  • Complicated by multiple active/sleep states