

# CPU Power Saving (in practice)

for CS 695

Brandon Thomson

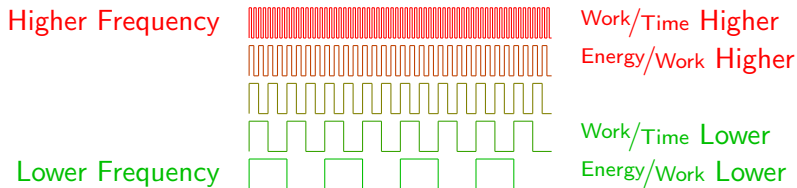
`brandon.j.thomson@gmail.com`

December 7 2010

# 10 Second Review

## CPU Power-saving Features

- Modern CPUs support fixed set of clock frequencies and sleep states



- ▶ Small but non-zero frequency transition latency
- CPU Sleeping?
  - ▶ **Zero** work done, **reduces energy use at any frequency**
  - ▶ Deeper sleep = higher wake latency
- Transitions must be controlled by operating system

# FreeBSD Intro

- FreeBSD™: Popular unix clone, especially for servers
- Used by: Cisco, Juniper, NetApp, Yahoo!, Netcraft, ...

# FreeBSD Intro

- FreeBSD™: Popular unix clone, especially for servers
- Used by: Cisco, Juniper, NetApp, Yahoo!, Netcraft, ...
- Easy to modify CPU speeds:

```
# sysctl dev.cpu.0.freq
dev.cpu.0.freq: 1700
# sysctl hw.acpi.thermal.tz0.temperature
hw.acpi.thermal.tz0.temperature: 67.0C
```

# FreeBSD Intro

- FreeBSD™: Popular unix clone, especially for servers
- Used by: Cisco, Juniper, NetApp, Yahoo!, Netcraft, ...
- Easy to modify CPU speeds:

```
# sysctl dev.cpu.0.freq
dev.cpu.0.freq: 1700
# sysctl hw.acpi.thermal.tz0.temperature
hw.acpi.thermal.tz0.temperature: 67.0C
# sysctl dev.cpu.0.freq=600
dev.cpu.0.freq: 1700 -> 600
```

# FreeBSD Intro

- FreeBSD™: Popular unix clone, especially for servers
- Used by: Cisco, Juniper, NetApp, Yahoo!, Netcraft, ...
- Easy to modify CPU speeds:

```
# sysctl dev.cpu.0.freq
dev.cpu.0.freq: 1700
# sysctl hw.acpi.thermal.tz0.temperature
hw.acpi.thermal.tz0.temperature: 67.0C
# sysctl dev.cpu.0.freq=600
dev.cpu.0.freq: 1700 -> 600
```

(3 minutes later ...)

```
# sysctl hw.acpi.thermal.tz0.temperature
hw.acpi.thermal.tz0.temperature: 33.0C
```

# Problem Statement

## FreeBSD's Speed-Scaling Algorithm is Not Ideal

- Let's look at the algorithm

# Problem Statement

## FreeBSD's Speed-Scaling Algorithm is Not Ideal

- Let's look at the algorithm
- “load”: How often CPU was busy during last 250ms interval

**while** *true* **do**

**if** load > 95% **then**

        | *double frequency*

**else if** load > 75% **then**

        | *increase frequency slightly*

**else if** load < 50% **then**

        | *reduce frequency slightly*

        | *sleep for 250ms*

**end**

<http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/usr.sbin/powerd/powerd.c>

# Problem Statement

## FreeBSD's Speed-Scaling Algorithm is Not Ideal

- Let's look at the algorithm
- “load”: How often CPU was busy during last 250ms interval

“More Responsive”

**while** *true* **do**

**if** load > 95% **then**

        | *quadruple* frequency

**else if** load > 75% **then**

        | *increase frequency slightly* ×2

**else if** load < 50% **then**

        | *reduce frequency very slightly*

        | *sleep for 250ms*

**end**

<http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/usr.sbin/powerd/powerd.c>

## Problem: Slow Responsiveness

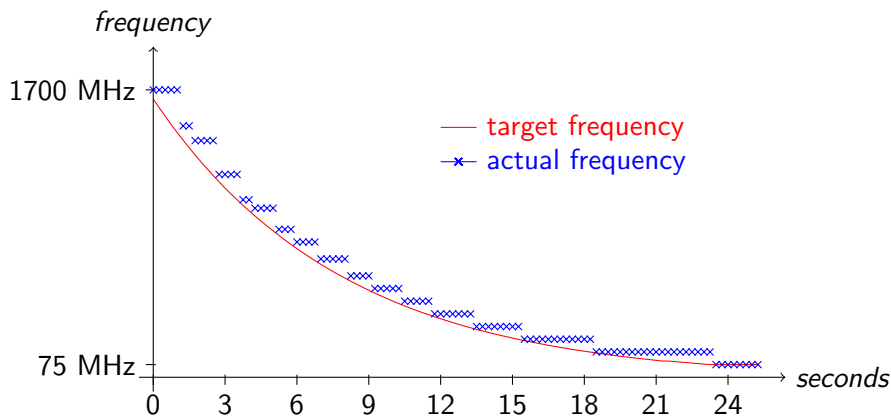
- Worst case: 100% load for 437.5 ms before frequency is doubled or quadrupled
  - ▶  $.75 \times 250 + 250 = 437.5$

# Problem: Slow Responsiveness

- Worst case: 100% load for 437.5 ms before frequency is doubled or quadrupled
  - ▶  $.75 \times 250 + 250 = 437.5$
- Solution: Reduce polling interval from 250 ms?
  - ▶ Lower polling interval increases background system load
  - ▶ With tickless kernel, CPU will wake from low-power sleep just to check its own speed [1]

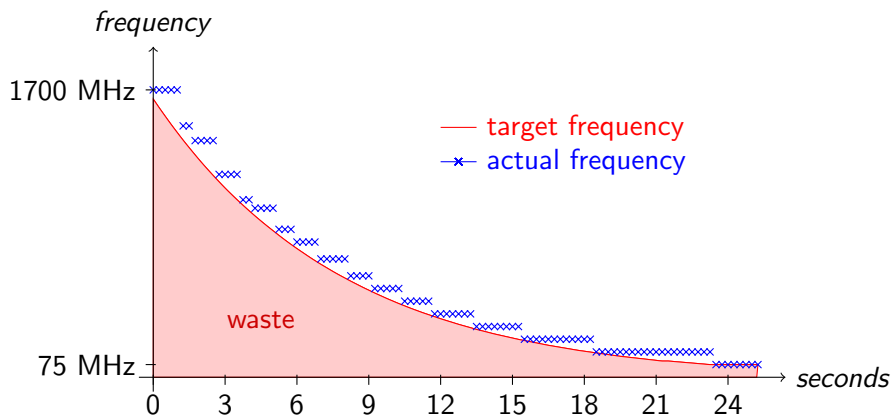
## Problem: Busy Bursts

- After full frequency, return to minimum is slow even if load is 0
  - ▶ Why? It minimizes impact of previous problem
- Measurements from laptop:



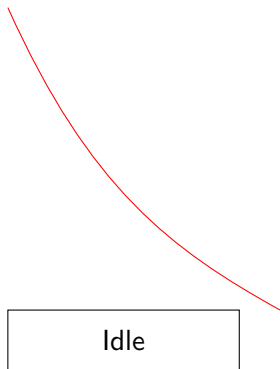
## Problem: Busy Bursts

- After full frequency, return to minimum is slow even if load is 0
  - ▶ Why? It minimizes impact of previous problem
- Measurements from laptop:



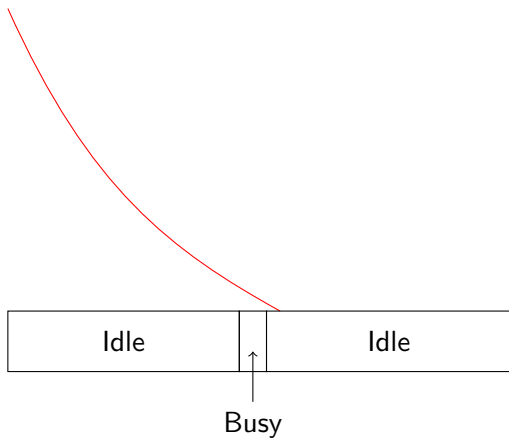
# Worst Case

Small busy periods just long enough to trigger frequency increase



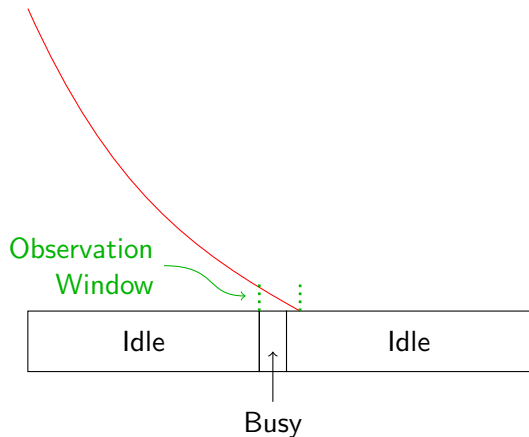
# Worst Case

Small busy periods just long enough to trigger frequency increase



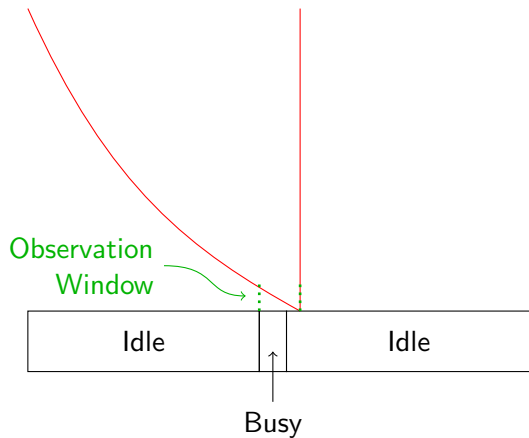
# Worst Case

Small busy periods just long enough to trigger frequency increase



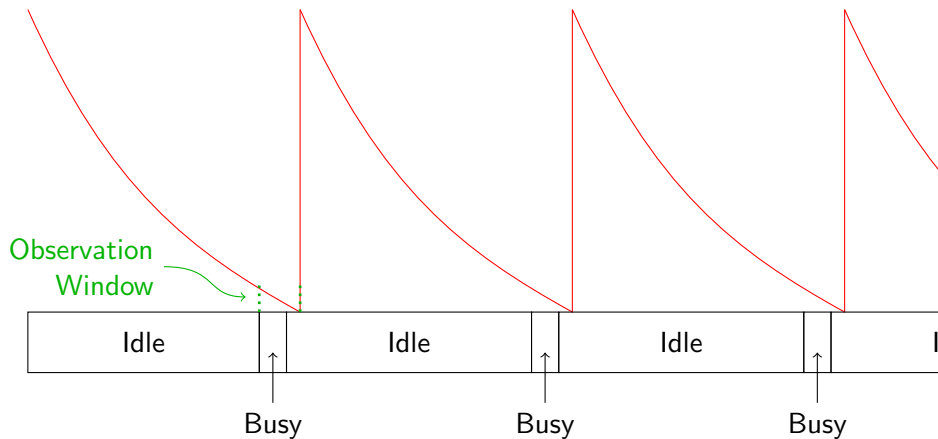
# Worst Case

Small busy periods just long enough to trigger frequency increase



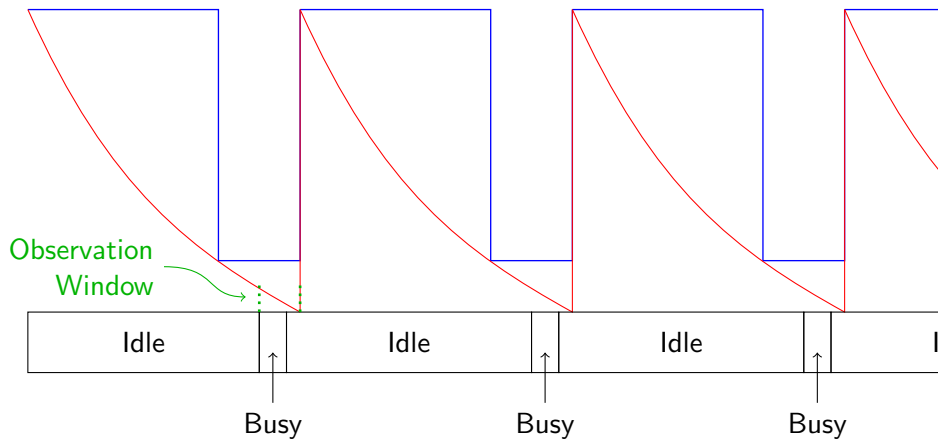
# Worst Case

Small busy periods just long enough to trigger frequency increase



# Worst Case

Small busy periods just long enough to trigger frequency increase



# What is Practical?

- Closer Approximation → Less Energy Consumption
  - ▶ ... or does it?

# What is Practical?

- Closer Approximation  $\rightarrow$  Less Energy Consumption
  - ▶ ... or does it?

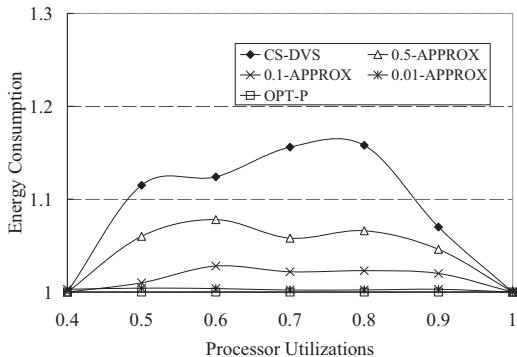


Chart: [2]

# What is Practical?

- Closer Approximation  $\rightarrow$  Less Energy Consumption
  - ▶ ... or does it?

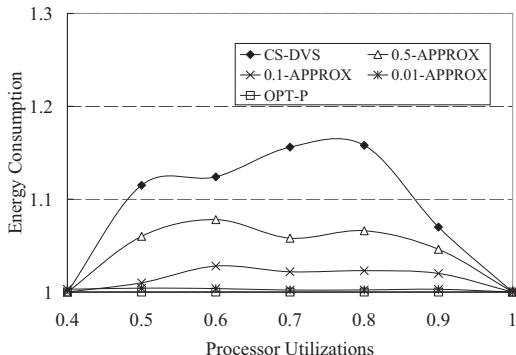


Chart: [2]

- Problem: Algorithms run on same CPU we are managing
- Problem: Excessive data collection is wasteful
- Question: What is practical?

# Improvement Ideas

- Move algorithm into kernel
- Use run queue length as frequency increase hint
- Reuse kernel's sleep state selection algorithms to reduce frequency if necessary
  - ▶ In other words, if we are going to spend 80  $\mu$ s to go into C3 because the system has been idle, might as well spend a few ns to see whether CPU is at a minimum speed first and adjust it if not
- After increasing CPU frequency due to load, check again after a smaller interval than normal to see if it should be increased again

# Measuring Power Usage

- Watt-meter between plug and wall-socket
  - ▶ Cons: \$\$\$, can't save data to PC quickly
- Software calculations based on manufacturer's power figures for speed/sleep states
  - ▶ Pro: Easy
  - ▶ Cons: Not available for all CPUs, may be inaccurate
- CPU Temperature
  - ▶ Pro: Measures CPU only, not other subsystems
  - ▶ Con: Inaccurate, not direct measure of current consumption
- ACPI battery sensors
  - ▶ Pro: Fairly Accurate
  - ▶ Con: Measures whole system, not just CPUs

# Linux's Advantage

- Tickless Scheduling [3]
  - ▶ Don't wake the CPU at regular fixed intervals
  - ▶ Works only with newer CPUs
  - ▶ Fewer timing interrupts = Better use of sleep states
- Scheduler Power Saving Mode
  - ▶ Consolidate processes onto similar CPU cores in SMP systems.
  - ▶ Fewer cores in active use = Better use of sleep states

# FreeBSD Software Layout

- `cpufreq`: CPU/architecture-dependent cpu speed-changing code
- `acpi_cpu`: Sleep state controller
- `powerd`: Speed-scaling daemon (makes the decisions)



# Bibliography I

- [1] Eric Anholt. *PowerTOP for FreeBSD* ? freebsd-acpi mailing list. 2007. URL: <http://lists.freebsd.org/pipermail/freebsd-acpi/2007-May/003712.html>.
- [2] Xiliang Zhong and Cheng zhong Xu. "System-wide energy minimization for real-time tasks: Lower bound and approximation". In: *In Proceedings of ICCAD*. 2006.
- [3] *Saving power on Intel systems with Linux*. URL: <http://www.lesswatts.org/results/server/index.php>.