Duke’s Milly Watt Project
Carla Ellis

Faculty
• Alvin Lebeck
• Amin Vahdat (UCSD)
Alumni
• Xiaobo Fan, Ph.D.
• Heng Zeng, Ph.D.
• Surendar Chandra, Ph.D

Students
• Sita Badrish
• Rebecca Braynard
• Angela Dalton
• Albert Meixner
• Shobana Ravi

Milly Watt Motivation

Energy for computing is an important problem (& not just for mobile computing)
– Reducing heat production and fan noise
– Extending battery life for mobile/wireless devices
– Conserving energy resources (lessen environmental impact, save on electricity costs)

How does software interact with or exploit low-power hardware?
Energy should be a “first class” resource at upper levels of system design
- Focus on Architecture, OS, Networking, Applications
- Energy has an impact on every other resource of a computing system – it is central.

HW / SW cooperation to achieve energy goals
Power Budget

[Download targets]

Outline

• Introduction and motivation
• Milly Watt activities
  – ECOSSystem
    Explicitly managing energy via the OS
    (ASPLOS02, USENIX03)
  – Power-aware memory
    (ASPLOS00, ISLPED01, PACS02, PACS03)
  – FaceOff
    Sensor-based display power management (HOTOS03, Mobisys Context Aware 04)
• Current and future directions
Outline

• Introduction and motivation
• Milly Watt activities
  – ECOSystem
    Explicitly managing energy via the OS
    (ASPLOS02, USENIX03)
  – Power-aware memory
    (ASPLOS00, ISLPED01, PACS02, PACS03)
  – FaceOff
    Sensor-based display power management (HOTOS03,
    Mobisys Context Aware 04)
• Current and future directions

NSCU, September 2004
Energy Centric Operating System (ECOSystem)

1. Energy can serve as a unifying concept for managing a diverse set of resources.
   - We introduce the currentcy abstraction to represent the energy resource.

2. A framework is needed for explicit monitoring and management of energy.
   - We develop mechanisms for currentcy accounting, currentcy allocation, and scheduling of currentcy use.

3. We need policies to achieve energy goals.
   - Need to arbitrate among competing demands and reduce demand when energy is limited.

Unified Currentcy Model

Energy accounting and allocation are expressed in a common currentcy. Abstraction for

1. Characterizing power costs of accessing different resources
2. Quantifying overall energy consumption
3. Sharing among competing tasks
Energy Goals

1. Explicitly manage energy use to reach a target battery lifetime.
   - Coast-to-coast flight with your laptop
   - Sensors that need to operate through the night and recharge when the sun comes up

2. If that requires reducing workload demand, use energy in proportion to task’s importance.
   Scenario:
   - Revising and rehearsing a PowerPoint presentation
   - Spelling and grammar checking threads
   - Listening to MP3s in background

NSCU, September 2004
Energy Goals

3. Deliver good performance given constraints on energy availability
   • Fully utilize the battery capacity within the target battery lifetime with little leftover capacity – no lost opportunities.
   • Encourage efficiency in performing desired work.
   • Address observed performance problems (e.g. energy-based priority inversions).

Mechanisms in the ECOSystem Framework

Currentcy Allocation
   • Epoch-based allocation – periodically distribute currentcy “allowance”

Currentcy Accounting
   • Basic idea:
     Pay as you go for resource use – no more currentcy ➔ no more service.
1. Determine overall amount of currency available per energy epoch.
2. Distribute available currency proportionally among tasks.

3. Deduct currency from task’s account for resource use.
Device Specific Accounting

- CPU: hybrid of sampling and task switch accounting
- Disk: tasks directly pay for file accesses, sharing of spinup & spindown costs.
- Network: local source or destination task pays based on length of data transferred

ECOSystem Prototype

- Modifications to Linux on Thinkpad T20
- Initially managing 3 devices: CPU, disk, WNIC
- Embedded power model:
  - Calibrated by measurement
  - Power states of managed devices tracked
    - Orinoco card: doze 0.045W, receive 0.925W, send 1.425W.
Experimental Evaluation V1.0

- Validate the embedded energy model.
- Can we achieve a target battery lifetime?
- Can we achieve proportional energy usage among multiple tasks?
- Assess the performance impact of limiting energy availability.

Achieving Target Battery Lifetime

- Using CPU intensive benchmark and varying overall allocation of currentcy, we can achieve target battery lifetime.
Proportional Energy Allocation

Battery lifetime is set to 2.16 hours (unconstrained would be 1.3 hr)
Overall allocation equivalent to an average power consumption of 5W.

Proportional CPU Utilization

Performance of compute bound task (ijpeg) scales proportionally with currentcy allocation
But - Netscape Performance Impact

Some applications don’t gracefully degrade with drastically reduced currentcy allocations

Experiences

Identified performance implications of limiting energy availability that motivate further policy development:

- Mismatches between user-supplied specifications and actual needs of the task
- Scheduling not offering opportunities to spend allocation
- I/O devices and other activity causing a form of inversion
**Challenge**

To fully utilize available battery capacity within the desired battery lifetime with little or no leftover (residual) capacity.

⇒ Devise an allocation policy that balances supply and demand among tasks.  
  *Currentcy conserving allocation.*

**Problem: Residual Energy**

Allocations do not reflect actual consumption needs
**Problem: Residual Energy**

A task’s unspent currenty (above a “cap”) is being thrown away to maintain steady battery discharge.

⇒ Leftover energy capacity at end of lifetime.

---

**Currentcy Conserving Allocation**

Two-step policy. Each epoch:
1. Adjust per-task caps to reflect observed need
   - Weighted average of currenty used in previous epochs.
Currentcy Conserving Allocation

2. Redistribute overflow currentcy

Currentcy Conserving Allocation Experiment

Workload:
- Computationally intensive ijpeg – image encoder
- Image viewer, gqview, with think time of 10 seconds and images from disk
  - Performance levels out at 6500mW allocation.
- Total allocation of 12W, shares of 8W for gqview (too much) and 4W for ijpeg (capable of 15.5W).

Comparing against total allocation “correction” method in original prototype.
**Currentcy Conserving Allocation Results**

<img src="image.png" alt="Graph showing power consumption over battery lifetime">

<1% remaining capacity

NSCU, September 2004

**Challenge**

To produce more robust proportional sharing by ensuring adequate spending opportunities.

⇒ Develop CPU scheduling that considers energy expenditures on non-CPU resources.

Currentcy-aware scheduling or energy-centric scheduling.

NSCU, September 2004
Problem: Scheduling/Allocation Interactions

- Allocation shares may be appropriately specified and consistent with demand, but the ability to spend depends on scheduling policies that control the opportunities to access resource.
- Priority Inversion – a task with small allocation but large CPU component can dominate a task with larger allocation but demands on other devices.
- Scheduling should be “aware” of currentcy expenditures throughout the system.

Energy-Centric Scheduling

- The next task to be scheduled for CPU is the one with the lowest amount of currentcy spent in this epoch relative to its share
  - Captures currentcy spent on any device.
- Dynamic share – weighted by the task’s static share divided by currentcy spent in last epoch.
  - Compensation for previous lack of spending opportunities
Energy-Centric Scheduling Experiment

- Workload:
  - Computationally intensive ijpeg
  - Image viewer, gqview, with think time of 10 seconds and disk access (700mW)
    - Performance levels out at 6500mW allocation.
  - Given equal allocation shares, total allocation varied
- Comparing against round-robin and stride based on static share value.
Energy-Centric Scheduling Results

Benefits of Currentcy

Currentcy abstraction

• Provides a concrete representation of energy supply and demand – allowing explicit energy/power management.

• Provides unified view of energy impact of different devices – enabling multi-device, system-wide resource management
  – Comparable, quantifiable, tradeoffs can be expressed

• Encourages analogies to economic models – motivating a rich set of policies.
Contributions

• ECOSystem is a powerful framework for managing energy explicitly as a first-class OS resource.
• Currentcy model is capable of formulating non-trivial energy goals and serving as the basis for solutions
  – Reducing residual battery capacity when lifetime reached
  – Ensuring that scheduling works with currentcy allocation towards proportional energy sharing
  – Smoothing out response time variation
  – Encouraging greater disk energy efficiency

Power Aware DRAM

• Memory with multiple power states has become available
  – Fast access, high power
  – Low power, slow access
• New take on memory hierarchy
• How to exploit this opportunity?
Power State Transitioning

Requests completion of last request in run

\[ \text{gap} \geq t_{h\rightarrow l} + t_{l\rightarrow h} + t_{\text{benefit}} \]

Ideal case: Assume we want no added latency

Threshold based - delays transition down

On demand case - adds latency of transition back up
Power-Aware DRAM Main Memory Design

- Assume we access & control each chip individually
- 2 dimensions to affect energy policy: HW controller / OS
- Energy strategy:
  - Cluster accesses to already powered up chips
  - Interaction between power state transitions and data locality

Power-Aware DRAM

- Read/Write Transaction
- Rambus RDRAM Power States
- Active 300mW
- Standby 180mW
- Power Down 3mW
- Nap 30mW
**Dual-state HW Power State Policies**

- All chips in one base state
- Individual chip Active while pending requests
- Return to base power state if no pending access

![Diagram of Dual-state HW Power State Policies]

**Quad-state HW Policies**

- Downgrade state if no access for threshold time
- Independent transitions based on access pattern to each chip
- Competitive Analysis
  - rent-to-buy
  - Active to nap 100’s of ns
  - Nap to PDN 10,000 ns

![Diagram of Quad-state HW Policies]
Page Allocation and Power-Aware DRAM

- Physical address determines which chip is accessed
- Assume non-interleaved memory
  - Addresses 0 to N-1 to chip 0, N to 2N-1 to chip 1, etc.
- Entire virtual memory page in one chip
- Virtual memory page allocation influences chip-level locality

Page Allocation Polices

Virtual to Physical Page Mapping
- Random Allocation – baseline policy
  - Pages spread across chips
- Sequential First-Touch Allocation
  - Consolidate pages into minimal number of chips
  - One shot
- Frequency-based Allocation
  - First-touch not always best
  - Allow (limited) movement after first-touch
The Design Space

<table>
<thead>
<tr>
<th>Dual-state Hardware</th>
<th>Quad-state Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Allocation</td>
<td>Sequential Allocation</td>
</tr>
<tr>
<td>1 Simple HW</td>
<td>2 Can the OS help?</td>
</tr>
<tr>
<td>3 Sophisticated HW</td>
<td>4 Cooperative HW &amp; SW</td>
</tr>
</tbody>
</table>

Evaluation Methodology

• Metric: Energy*Delay Product
  – Avoid very slow solutions
• Energy Consumption (DRAM only)
  – Processor & Cache do affect runtime
• Trace-Driven Simulation
  – Windows NT personal productivity applications (Etch traces from U. Washington)
  – Simplified processor and memory model
• Execution-Driven Simulation
  – SPEC benchmarks (subset of integer)
  – SimpleScalar w/ detailed RDRAM timing and power models
### Summary of Simulation Results

*(Energy*Delay product, RDRAM, ASPLOS00)*

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Random Allocation</th>
<th>Sequential Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual-state</td>
<td>Nap is best dual-state policy 60%-85%</td>
<td>Additional 10% to 30% over Nap</td>
</tr>
<tr>
<td>Quad-state</td>
<td>Improvement not obvious, Could be equal to dual-state</td>
<td>Best Approach: 6% to 55% over dual-nap-seq, 80% to 99% over all active.</td>
</tr>
</tbody>
</table>

**Other Questions**

- How to determine the best thresholds in memory controller design?
- Are more sophisticated OS page allocation (or migration) policies useful?
- How do power-state components (power-aware DRAM) and dynamic voltage scaling (processors) interact?
- Is there a policy based on adaptive thresholds for transitioning power-state devices (in general -- memory, disks, wireless)?
Naïve Power-awareness

- Lowest energy achieved at 400MHz
  - Memory remains powered on too long in low frequencies
  - CPU energy too high in high frequencies
- Result conflicts with conventional DVS
  - Memory has to be taken into account
Aggressive Power-awareness

- Lowest frequency wins again
  - CPU energy becomes dominant
  - Memory energy greatly reduced and stabilizes
- Effective power-aware memory contributes to realizing the potential of DVS
Contributions

- Demonstrated dramatic improvements in energy*delay for power-aware page allocation
- Frequency-based allocation little impact
- Device-level general power management
  - Based on histogram of gaps in moving window to capture non-stationarity in access pattern
  - Efficient tree algorithm updates energy and searches threshold space
- DVS and Power-aware memory interactions explored
  - Technique for DVS to choose optimal frequency with the consideration of memory effect

FaceOff

- Goal: to reduce system energy consumption by using low power sensors to match I/O behavior more directly to user behavior and context.
  - A display is only necessary if someone is looking at it.
Prototype

- IBM ThinkPad T21 running RedHat Linux
  - Base + max CPU power consumption = 18 Watts
  - Display = 7.6 Watts
- Logitech QuickCam Web Cam
  - Power Consumption = 1.5 Watts
- X10 ActiveHome Wireless Motion Sensor and Receiver
- Software components:
  - Image capture, face detection, display power state control (ACPI)
Face Detection

• Simple skin detection used for prototype

Feasibility Study

• What is the potential for energy savings?
  – “Best case” scenarios to measure opportunity
    • Assume perfect accuracy
    • User behavior – start it and leave, return on completion.

• What is the effect on System Performance
  – Network file transfer (113 MB)
  – CPU intensive process (Linux kernel compile)
  – MP3 Song (no display necessary)

• How responsive is the system?
File Transfer

Tradeoff of energy costs: CPU image processing plus camera power vs. display energy during idle timeout.

Energy and Time Comparisons

<table>
<thead>
<tr>
<th>Energy (J)</th>
<th>Default</th>
<th>With FaceOff</th>
<th>%Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>File transfer</td>
<td>6795</td>
<td>4791</td>
<td>29.5</td>
</tr>
<tr>
<td>Kernel compile</td>
<td>12507</td>
<td>11023</td>
<td>11.9</td>
</tr>
<tr>
<td>MP3</td>
<td>4714</td>
<td>3403</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Default</th>
<th>With FaceOff</th>
<th>%Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>File transfer</td>
<td>348.6</td>
<td>351.3</td>
<td>.8</td>
</tr>
<tr>
<td>Kernel compile</td>
<td>575</td>
<td>603.5</td>
<td>4.9</td>
</tr>
<tr>
<td>MP3</td>
<td>No effect on playback</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NSCU, September 2004
Responsiveness Timing

- polling latency
- detection latency

Face arrives (or departs) → Image acquired → detection complete → display signaled → Total responsiveness latency

Detection Latency Under Load

<table>
<thead>
<tr>
<th>Workload</th>
<th>Average (99% Confidence)</th>
<th>Maximum</th>
<th>Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Transfer</td>
<td>175±7ms</td>
<td>305ms</td>
<td>116ms</td>
</tr>
<tr>
<td>Kernel Compile</td>
<td>230±5ms</td>
<td>669ms</td>
<td>51ms</td>
</tr>
<tr>
<td>MP3</td>
<td>154±3ms</td>
<td>229ms</td>
<td>84ms</td>
</tr>
</tbody>
</table>
**On-going Work on FaceOff**

- Continue work on optimizing responsiveness & overhead
- Comprehensive user study
  - Survey of usability
  - Characterization of “real deployment” usage patterns
    - End-to-end experiment
  - Energy measurement under realistic usage

---

**Milly Watt Project: Future Directions**

![Diagram](image)

New energy goals: efficiency; application cooperation

New platforms: Motes with TinyOS+ currentcy

New devices & policies: integrating the display; economics-based file system

ECOSystem

Distributed systems: sensor networks

NSCU, September 2004
For More Information

www.cs.duke.edu/ari/millywatt/

email: carla@cs.duke.edu