Intelligent Offload to Improve Battery Lifetime of Mobile Devices

Ranveer Chandra
Microsoft Research
Phones are Energy Constrained

- Energy: A critical issue in smartphones
  - Limited battery lifetime

- Battery energy density only doubled in last 15 years

- Smartphone capability has increased drastically
  - Multiple Components: GPS, 3G, retina display, ....
Where Does the Energy Go?

Android G1 energy consumption
“An Analysis of Power Consumption in a Smartphone”, USENIX 2010

Suspended State
(68 mW)

Video Playback
(450 mW + backlight)

Network, Display & CPU are the main energy hogs!
Efforts to Improve Battery Life

- **Battery**: bigger, more energy density
  - Challenge: Lighter phones

- **CPU**: low power cores, parking individual cores
  - Challenge: multi-core, faster processors

- **Network**: low power cellular & Wi-Fi states
  - Challenge: LTE, 802.11ac

- **Display**: energy-efficient display, e.g. AMOLED
  - Challenge: larger & brighter displays, video, animation
Mission & Big Bets

“Lasting a week without charge under normal usage”

- Profile energy use of each component
  - Application energy profiler
  - Energy debugging

- Big Bets:
  - **Offload**: Intelligently utilize available resources
  - **Energy-aware UI**: based on OLED energy models
  - **Adaptive battery usage**: OS controlled multi-battery systems
OFFLOAD
Computational Offload

- Move computation away from main processor without degrading user experience
  - Such that SoC is in low power states for longer

SoC (~1W when awake, ~10 mW when asleep)
Where to Offload?

- Low power subsystem on SoC
  - Already shipping with TI and other SoC vendors

SoC (~1W when awake, ~10 mW when asleep)
Where to Offload?

- Low power subsystem on SoC
- Low power processor connected to NIC

SoC (~1W when awake, ~10 mW when asleep)
Where to Offload?

- Low power subsystem on SoC
- Low power processor connected to NIC
- Cloud or another machine
Use Cases

- When display is off
  - Push e-mail
  - Continuous sensing
  - Large downloads
  - Instant Messaging
  - P2P file sharing
- When display is on
  - Gaming
  - Speech translation
  - ...
OFFLOAD TO LOW-POWER PROCESSOR

SoC (~1W when awake, ~10 mW when asleep)
100’s of apps using the accelerometer

Not using the full potential of sensors
Responsive Sleeping Challenge

- Sensor (accelerometer): 0.56mW
- Phone (mainly processor): ~600mW
- Wakeup + sleep time: 1200ms
- @ 1Hz sampling, processor can’t sleep

Solution: Offload sampling/processing sensor data to a low-power processor
Energy Efficient Responsive Sleeping

Phone Application Processor

Sensor buses (I2C, SPI etc.)

Sensors

Current Phones

Phone Application Processor

High-speed serial bus + GPIO

Low-power proc.

Sensor buses (I2C, SPI etc.)

Sensors

EERS

Bodhi, Jie, Dimitrios (2010)
Hardware Prototype

- Interfaced directly to phone’s AP
- Processor: MSP430F5438
  - 16KB RAM, 256KB Flash
  - Active: 6.6mW @16MHz; sleep: 10μW
  - Wakeup time: 4μs
- Sensors:
  - Temperature
  - Pressure
  - 3D compass
  - 3D accelerometer
  - 3D gyro
  - Capacitive touch sensing (x16)
- Reprogrammable over the phone
- Leakage (sleep) power: 270mW
- Extensible: can accommodate more sensors, radios etc.
- Can interrupt and turn on/off the phone
- Interfaced directly to phone processor (SPI bus + GPIOs)
- Directly powered from the phone’s battery
Summary

- Low Power processor on SoC can drive sensors
  - Key application: Continuous sensing

TI announces OMAP 5: two high-performance and two low-power cores, devices next year

By Chris Ziegler posted Feb 7th 2011 12:24PM

Windows 8: Microsoft shows off "sensor fusion" support
OFFLOAD TO NETWORK-CONNECTED LOW-POWER PROCESSOR

SoC (~1W when awake, ~10 mW when asleep)

Collaborators: Yuvraj Agarwal, Steve Hodges, James Scott, Victor Bahl
Power/Energy Efficiency are Key Drivers Today

Battery Powered Computers

**Lenovo X61 laptop**
- Power: 0.74W (sleep) to 16W (active)
- Goal: improve battery lifetime

“Wall Powered” Computers

**Dell Optiplex 745 desktop**
- Power: 1.2W (sleep) to >140W (active)
- Goal: reduce energy costs and impact to the environment

Energy efficiency: do more work for less power or energy
IT Equipment Consumes Significant Power

- Yet, shutdown opportunities are rarely used

- Studies show that:
  - 67% of office PCs are left on after work hours
    - “Sleep” modes used in less than 4% of these PCs! [1]
  - Home PCs are left on for 34% of the time
    - 50% of the time they are not being used

- Confirmed by our measurements: CSE@UCSD
  - 600+ desktops left always on (total=700+ )
    - @150W each $\Rightarrow$ 100kW (25% of total energy bill)

- Propriety solutions at WaMu, Dell and GE have reported savings of millions dollars per year
  - Thousands of tons of CO$_2$ emission avoided!!!

Saving Power Runs into Usability

- Reasons why users do not switch off their PCs
  - Maintain state: desktop and applications preferences
  - Occasional access
    - Remote desktop/SSH, accessing files
    - Administrative: updates, patches, backups
  - Active applications running
    - Maintaining presence: e.g. incoming Skype call, IM
    - Long running applications: Web downloads, BitTorrent

Cannot be handled by low-power modes (e.g. Sleep, Hibernate)
Power Management vs. Use Models

- Current design trends in power management:
  - Hosts (PCs): either *Awake* (Active) or *Sleep* (Inactive)
    - Power consumed when Awake = 100X power in Sleep!
  - Network: Assumes hosts are always “Connected” (Awake)

- What users really want:
  - Provide functionality of an *Awake* (active) host...
    ....While consuming power as if in *Sleep* mode
  - Resume host to Awake mode only if needed

Change the fundamental distinction between Sleep and Active states...
Augment PC’s Network Interface

- **Objective:** Make PCs responsive even when asleep
  - Maintain availability across the entire protocol stack
    - E.g. ARP (layer 2), ICMP (layer 3), SSH (Application layer)
  - Without making changes to the infrastructure or user behavior

Active State: >140 W
Idle State: 100 W
Sleep State: 1.2 W

**Requirements:**
- Functionally similar - masquerade as the host
- Much lower power

**Secondary Processor**
(Power in active state ~1W)

- Low Power CPU
- DRAM
- Flash Memory Storage
Somniloquy*: PCs Talk in their Sleep

- Augment network interfaces:
  - Add a separate power domain
    - Powered on when host is asleep
    - Processor + Memory + Flash Storage + Network stack
  - Same MAC/IP Address
- Wake up Host when needed
  - E.g. incoming connection
- Handle some applications while PC remains asleep
  - Using “application stubs”
Supporting Stateless Apps: Filters

- Wake up host on any user defined “filter”
  - E.g. incoming Skype call, Remote Desktop request
  - Wake-on-LAN either impractical or affects usability

- Specified at any layer of the network stack
  - E.g. from a particular IP (layer 3) or MAC (layer 2)
  - E.g. wake up on finding “MSFTWLAN” Wi-Fi network
Supporting Stateful Apps: Stubs

- Applications actively maintain state
  - E.g. background web downloads, P2P file sharing (BitTorrent)
  - Need application specific code on the secondary processor

- Challenge: secondary processor limited in resources
  - CPU, memory, flash storage
  - Cannot run the full application

- Offload part of the applications: i.e. “stub” code
  - Generate “stub” code manually
  - Stubs for BitTorrent, Web downloads, IM
Software Components

Somniloquy States

Timer-based or user-initiated sleep

Wake up on incoming network event or timer-based/user-initiated action
Somniloquy Prototype

- Prototype uses “gumstix” platform
  - PXA270 processor with full TCP/IP stack
  - USB connection to PC for sleep detection/wakeup trigger, power while asleep, and IP networking for data

- Wired and wireless prototypes
  - *-1NIC version follows initial vision of augmented NIC, where all data goes via gumstix even when PC is awake
  - *-2NIC version uses PC’s internal interface while it is awake, and allows for simpler legacy-friendly support
Prototype

USB Interface
(Wake up Host + Status + Debug)

USB Interface
(power + USBNNet)

SD Storage

Processor

100Mbps Ethernet Interface
Evaluation Methodology

- Maintain network reachability

- Stateless applications (filter based):
  - Measure increase in “application layer” latency
  - Detailed power profile: Gumstix, Host PCs
  - Extend battery lifetime (Laptops), Energy Savings (Desktop)

- Stateful applications (stub based):
  - Measure energy savings
Maintaining Reachability

- Respond to “ping”, ARPs, maintain DHCP lease

*Break in ICMP responses are due to state transitions: Sleep ↔ Active*
Stateless Apps: “Setup” Latency

- Measured time till user-perceived response
- For each, incoming TCP SYN caused wakeup
- Additional latency: 3-10s for all prototypes
- As a proportion of the resulting session, this is OK
# Gumstix: Power Consumption

<table>
<thead>
<tr>
<th>Gumstix State</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WIRED VERSION</strong></td>
<td></td>
</tr>
<tr>
<td>1. Gumstix only – No Ethernet</td>
<td>210mW</td>
</tr>
<tr>
<td>2. Gumstix + Ethernet Idle</td>
<td>1073mW</td>
</tr>
<tr>
<td>3. Gumstix + Ethernet + Write to flash</td>
<td>1675mW</td>
</tr>
<tr>
<td><strong>WIRELESS VERSION</strong></td>
<td></td>
</tr>
<tr>
<td>4. Gumstix only - no Wi-Fi</td>
<td>210mW</td>
</tr>
<tr>
<td>5. Gumstix + Wi-Fi associated (PSM)</td>
<td>290mW</td>
</tr>
<tr>
<td>6. Gumstix + Wi-Fi Associated (CAM)</td>
<td>1300mW</td>
</tr>
</tbody>
</table>

- Our prototypes consume 290mW (Wi-Fi) to 1W (Ethernet)
  - Similar to power consumed by our test laptop (740mW) in the “sleep” state.
  ...and our test desktop (1.2W) in the “sleep” state.
Using Somniloquy:

– Power drops from >100W to <5W
– Assuming a 45 hour work week
  – 620kWh saved per year
  – US $56 savings, 378 kg CO₂
Laptops: Extends Battery Lifetime

Using Somniloquy:
– Power drops from >11W to 1W,
  – Battery life increases from <6 hours to >60 hours
– Provides functionality of the “Baseline” state
  – Power consumption similar to “Sleep” state
Energy Savings for Sample Workloads

- Use trace data from [Nedevschi-NSDI2009]
  - 24 desktop PCs: ON, sleep, idle and OFF durations

- Bin data into 3 categories based on % idle time

<table>
<thead>
<tr>
<th>% Idle Time</th>
<th>% Energy Saving using Somniloquy</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;25% of the time (7 PCs)</td>
<td>38%</td>
</tr>
<tr>
<td>25% - 75% of the time (6 PCs)</td>
<td>68%</td>
</tr>
<tr>
<td>&gt;75% of the time (9 PCs)</td>
<td>85%</td>
</tr>
</tbody>
</table>
Stateful Application: Energy Savings

- Web download “stub” on the gumstix
  - 200MB flash, download when Desktop PC is asleep
  - Wake up PC to upload data whenever needed

- 92% less energy than using the host PC for download
Summary:

- Somniloquy: augment network interfaces of PCs
  - Maintain reachability and availability transparently
  - Power consumption similar to a “sleep” state

- Incrementally deployable prototype
  - No changes to infrastructure, application servers

- Demonstrable savings
  - Desktops: reduced energy cost, carbon footprint
  - Laptops: extend battery lifetime
In the context of mobile devices

- Network-connected low power processor can:
  - Sync with e-mail
  - Perform IM tasks
  - Run Skype in the background
  - Download music

... all without waking up the main processor
CLOUD OFFLOAD

SoC (~1W when awake, ~10 mW when asleep)

Collaborators: Eduardo Cuervo, Aruna Balasubramanian, Alec Wolman, Stefan Saroiu, Victor Bahl
Mobile apps can’t reach their full potential.

- Speech Recognition and Synthesis: Slow, Limited or Inaccurate
- Interactive Games: Not on par with desktop counterparts
- Augmented Reality: Too CPU intensive
- Power Intensive: Limited
One Solution: Remote Execution

- Remote execution can reduce energy consumption

Challenges:
- What should be offloaded?
- How to dynamically decide when to offload?
- How to minimize the required programmer effort?
MAUI Contributions:

- Combine extensive profiling with an ILP solver
  - Makes dynamic offload decisions
  - Optimize for energy reduction
  - Profile: device, network, application

- Leverage modern language runtime (.NET CLR)
  - To simplify program partitioning
  - Reflection, serialization, strong typing
Roadmap

- Motivation
- **MAUI system design**
  - MAUI proxy
  - MAUI profiler
  - MAUI solver
- Evaluation
MAUI Architecture

Application

Maui Runtime

Client Proxy

Profiler

Solver

Smartphone

Maui Controller

Maui Runtime

Server Proxy

Profiler

Solver

Maui server

RPC

RPC
How Does a Programmer Use MAUI?

- Goal: make it dead-simple to MAUI-ify apps
  - Build app as a standalone phone app
  - Add .NET attributes to indicate “remoteable”
  - Follow a simple set of rules

```csharp
[Remoteable]
ArrayList GetValidMoves(Square s)
{
    if (s.IsEmpty())
    {
        return new ArrayList();
    }
    if (s.Piece.IsEnemyOf(active))
    {
        //this piece does not belong to the active side, no moves possible
        return new ArrayList();
    }
    //forward the call to the Rule-class
    return rules.getMoves(s);
}
```
Run-Time Support For Partitioning

- **Portability:**
  - Mobile (ARM) vs Server (x86)
  - .NET Framework Common Intermediate Language
- **Type-Safety and Serialization:**
  - Automate state extraction
- **Reflection:**
  - Identifies methods with [Remoteable] tag
  - Automates generation of RPC stubs
MAUI Proxy

- **Maui Runtime**: Provides runtime information and handles errors.
- **Profiler**: Intercepts application calls, synchronizes state, and chooses local or remote.
- **Solver**: Handles errors.
- **RPC**: Interacts between Maui Runtime and Client Proxy.

---

**Smartphone**
- **Maui Controller**: Intercepts application calls.
- **Application**: Receives and sends data to Maui server.

---

**Maui server**
- **Server Proxy**: Manages communication with Maui Runtime.
- **Profiler**: Monitors and reports application performance.

---

49
MAUI Profiler

- Callgraph
- Execution Time
- State size
- CPU Cycles
- Device Profile
- Network Latency
- Network Bandwidth
- Computational Power Cost
- Computational Delay
- Network Power Cost
- Network Delay
- Computational Delay

Annotated Callgraph

Network Power Cost
Network Delay
Computational Delay
MAUI Solver

A sample callgraph

Energy and delay for state transfer

Computation energy and delay for execution

A

B

C 5000 mJ 3000 ms

D 15000 mJ 12000 ms

1000mJ

10000 mJ

10000 mJ
Is Global Program Analysis Needed?

Yes! – This simple example from Face Recognition app shows why local analysis fails.

User Interface

1000mJ

FindMatch
900 mJ

Cheaper to do local

InitializeFace
Recognizer
5000 mJ

DetectAndExtract
Faces
15000 mJ

10000 mJ

10000 mJ
Is Global Program Analysis Needed?

Yes! – This simple example from Face Recognition app shows why local analysis fails.
Is Global Program Analysis Needed?

- User Interface: 1000mJ
- FindMatch
- InitializeFace Recognizer: 25900mJ
- DetectAndExtract Faces

Cheaper to offload
Adapting to Changing Conditions

- Adapt to:
  - Network Bandwidth/Latency Changes
  - Variability on method’s computational requirements

- Experiment:
  - Modified off the shelf arcade game application
  - Physics Modeling (homing missiles)
  - Evaluated under different latency settings
Adapting to Changing Conditions?

DoFrame → DoLevel → HandleEnemies → HandleBonuses → HandleMissiles

11KB + missiles

Required state is smaller

Complexity increases with # of missiles

*Missiles take around 60 bytes each
Case 1

- Zero Missiles
- Low latency (RTT < 10ms)

Offload starting at DoLevel

Computation cost is close to zero

*Missiles take around 60 bytes each
Case 2

- 5 Missiles
- Some latency (RTT = 50ms)

*Missiles take around 60 bytes each

Very expensive to offload everything

Little state to offload

Only offload Handle Missiles

Most of the computation cost
Roadmap

- Motivation
- MAUI system design
  - MAUI proxy
  - MAUI profiler
  - MAUI solver
- Evaluation
MAUI Implementation

- **Platform**
  - Windows Mobile 6.5
  - .NET Framework 3.5
  - HTC Fuze Smartphone
  - Monsoon power monitor

- **Applications**
  - Chess
  - Face Recognition
  - Arcade Game
  - Voice-based translator
Questions

- How much can MAUI reduce energy consumption?
- How much can MAUI improve performance?
- Can MAUI Run Resource-Intensive Applications?
How much can MAUI reduce energy consumption?

Face Recognizer

- Smartphone only
- MAUI (Wi-Fi, 10ms RTT)
- MAUI (Wi-Fi, 25ms RTT)
- MAUI (Wi-Fi, 50ms RTT)
- MAUI (Wi-Fi, 100ms RTT)
- MAUI* (3G, 220ms RTT)

An order of magnitude improvement on Wi-Fi

Big savings even on 3G
How much can MAUI improve performance?

Face Recognizer

Improvement of around an order of magnitude

- Smartphone only
- MAUI (Wi-Fi, 10ms RTT)
- MAUI (Wi-Fi, 25ms RTT)
- MAUI (Wi-Fi, 50ms RTT)
- MAUI (Wi-Fi, 100ms RTT)
- MAUI* (3G, 220ms RTT)
Latency to server impacts opportunities for fine-grained offload

Up to 40% energy savings on Wi-Fi
Can MAUI Run Resource-Intensive Applications?

CPU Intensive even on a Core 2 Duo PC

Can be run on the phone with MAUI
SUMMARY
Looking ahead...

- Mechanisms are in place
  - Low power cores in the SoC
    - TI and others ...
  - Big.Little processors from ARM
    - Expected by end of year
  - Smart Web Services

- Offload Policies: the next big move?