Adaptive and Reflective Middleware and OS Services for Mobile Applications

Managing Power in Dynamic Distributed Environments through cross-layer adaptation

Rajesh Gupta
University of California, San Diego

MESL . UCSD . EDU
The Real Culprits

- Meta-Data and Its Use in Component Compositions
  - BALBOA Project
    - Sudipta Kundu, Frederic Doucet, Ingolf Krueger

- Awareness For Energy, Location
  - Energy awareness:
    - Zhen Ma, Yuvraj Agrawal, Zhong Yi Jin
    - Jihong Kim, SNU
    - Nalini Venkatasubramaniam, Nikil Dutt, Alex Nicolau, UC Irvine
  - Location awareness
    - SPATIAL PROGRAMMING
      - Ryo Sugihara, R. K. Shyamasundar, IRL & TIFR, India
    - DYNAMIC RESOURCE DISCOVERY
      - Jeffrey Namkung, Chalermek Intanononvihat, Amin Vahdat
RADIOS ARE EVERYWHERE…

AND SO IS THE NEED FOR POWER
Low Power Has Been A Design Focus

- Speed power efficiency has indeed gone up
  - 10x / 2.5 years for μPs and DSPs in 1990s
    - between 100 mW/MIP to 1 mW/MIP since 1990
  - IC processes have provided 10x / 8 years since 1965
  - rest from power conscious IC design in recent years
- Another 20X is possible.

<table>
<thead>
<tr>
<th>Processor</th>
<th>MHz</th>
<th>Year</th>
<th>SPECint-95</th>
<th>Watts</th>
</tr>
</thead>
<tbody>
<tr>
<td>P54VRT (Mobile)</td>
<td>150</td>
<td>1996</td>
<td>4.6</td>
<td>3.8</td>
</tr>
<tr>
<td>P55VRT (Mobile MMX)</td>
<td>233</td>
<td>1997</td>
<td>7.1</td>
<td>3.9</td>
</tr>
<tr>
<td>PowerPC 603e</td>
<td>300</td>
<td>1997</td>
<td>7.4</td>
<td>3.5</td>
</tr>
<tr>
<td>PowerPC 604e</td>
<td>350</td>
<td>1997</td>
<td>14.6</td>
<td>8</td>
</tr>
<tr>
<td>PowerPC 740 (G3)</td>
<td>300</td>
<td>1998</td>
<td>12.2</td>
<td>3.4</td>
</tr>
<tr>
<td>PowerPC 750 (G3)</td>
<td>300</td>
<td>1998</td>
<td>14</td>
<td>3.4</td>
</tr>
<tr>
<td>Mobile Celeron</td>
<td>333</td>
<td>1999</td>
<td>13.1</td>
<td>8.6</td>
</tr>
</tbody>
</table>

Source: ISI/USC, DARPA PACC Program
There are basically two ways to save power…(true for pretty much everything on-chip)

- **Shutdown** through choice of right system & device states
  - Multiple sleep states
  - Also known as Dynamic Power Management (DPM)
- **Slowdown** through choice of right system & device states
  - Multiple active states
  - Also known as Dynamic Voltage/Frequency Scaling (DVS)
- **DPM + DVS**
  - Choice between amount of slowdown and shutdown
“System Design” for Low Power

- Energy efficiency (has to) cut across all system layers
  - circuit, logic, software, protocols, algorithms, user interface, power supply...
  - Computation versus Communication; Node versus network

- Trade-off between energy consumption & QoS
  - optimize energy metric while meeting “quality” constraint
And thus began our love affair with ‘awareness’

- Knowing an application’s intent one can do a lot of power saving tricks at all levels: architecture, compiler, OS, middleware.

- Conversely, if the awareness for power/energy is seeped into all these levels, one can reduce power significantly.

- Together they can create a new contract in the computing system!

- Since power is important in radios, things with radios move (or they monitor things that move), location awareness is even more phenomenal for power reduction.
Outline:
Bringing energy awareness in application, OS and Middleware

A. Application

B. OS

C. Middleware
What does it mean to be ‘aware’?

- That the application and the services know about energy, power
  - File system, memory management, process scheduling
  - Make each of them energy aware

How does one make software to be “aware”?  
- Use “reflectivity” in software to build adaptive software
- Ability to reason about and act upon itself (OS, MW)
Reflection and Introspection: A HW Guy’s Way of Looking At It

- **Component:**
  - A unit of re-use with an interface and an implementation

- **Meta-information:**
  - Information about the structure and characteristics of an object

- **Reification:**
  - A data structure to capture the meta-information about the structure and the properties of the program

- **Reflection:**
  - An architectural technique to allow a component to provide the meta-information to himself

- **Introspection:**
  - The capability to query and modify the reified structures by a component itself or by the environment
Building HW Components W/ Meta-data

1. Start with SystemC descriptions of IP blocks
   - Multi-level (RTL, TL) descriptions

2. Capture meta information of these IP into XML
   - Mostly structural information for now.

3. Generate library of ‘XMLized’ IP blocks
   - Schema to match datatype and protocol type information across IP blocks
   - Create DOM model and constraints for the library

4. Develop methods for IP selection, composition, verification, synthesis
   - Automated methods for IP instantiation, interface generation

IP Selection through an Introspective Composer
- IP matching and connection
  - Insertion of bridges
  - Validation of functionality
  - Create an executable specification
Applying Reflection: performance, energy

- Use Meta data to represent resource demands, dynamic behavior of the program carrying it.
  - Resources: Memory (R/W, Cache), Processor (IPC)
- Enables energy-performance tuning by exploring resource demand variations throughout programs’ execution
  - Example: Profile of application over memory banks
  - Vary frequency of processor based on IPC demand
Example: Rambus DRAM (RDRAM)

- High bandwidth (>1.5 GB/sec)
- Each RDRAM module can be activated/deactivated independently
- Read/write can occur only in the active mode
- Three low-power operating modes:
  - Standby, Nap, Power-Down
Approach

1. Characterize application offline
   - Divide an application into phases of execution
     - A group of program intervals executing similar code
   - Each phase has similar demand on resources
     - Similar code, similar resource demands (memory, IPC)

2. Annotate source code
   - Phase signatures

3. Enable OS (and hardware) to recognize signature
   - Smart hardware and/or online learning techniques

4. Dynamically tune the power manager
   - As application moves from one phase to another.
1 Understanding application behavior

- Divide an application into phases of execution
  - A group of program intervals executing similar code
- Each phase has similar demand on resources
  - Similar code, similar resource demands
- Demand for resources varies during the execution of application
  - As it moves from one phase to another.
- Phases identified using BBV or LBV
  - Keep track of loop branches
2 Offline analysis of application

- A data structure with the summary of the information of interest for each phase is attached to the program
  - Fixed location for the program metadata
  - OS support to access metadata.

- Also a signature of each phase is attached to the program.
  - No. of times the loops of the program are executed in the particular phase
3 Runtime Analysis

- **Challenge:**
  - Detect in which phase the program is running
- **Learning, signature construction, partial matches**
  - Match the signature created offline with a online signature using Manhattan distance.
    - If distance < threshold a match is found.
    - Threshold tells how similar two intervals of execution are.
  - The same technique used for splitting the program in phases offline is applied online but using partial signatures for matching with the offline computed signatures.
4 Matching signature at runtime

- Use performance counters:
  - Can be programmed to generate an interrupt on specified counts

- ISR provides matching with the meta data and mode changes
  - Every S*10,000 loop branches try a match
  - Phase matching can also be done in hardware

- Notify power manager to trigger proper action
Adaptation for Memory Behavior

- Number of engineering optimizations
  - Frequency of adaptation
  - Granularity of analysis (phase granularity)
  - Tradeoff against cost of adaptation.
Results – Normalized to NAP

Average among bzip, mpeg, ghostscript and ADPCM
Results - overheads

- Approx. 350K instructions for every 10,000 loop branch instructions
- Number of instructions executed by the match algorithm at every 10,000 loop branches to match a partial signature (500 instructions per phase)

<table>
<thead>
<tr>
<th># of phases</th>
<th># instructions</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2,580</td>
<td>0.7%</td>
</tr>
<tr>
<td>10</td>
<td>4,500</td>
<td>1%</td>
</tr>
<tr>
<td>20</td>
<td>8,280</td>
<td>2%</td>
</tr>
<tr>
<td>30</td>
<td>12,060</td>
<td>3%</td>
</tr>
</tbody>
</table>

- Size overhead. 4 bytes per inter arrival estimate per bank / phase. 4 x 16 x 10 = 640 bytes assuming 16 banks and 10 phases.
- The signatures take 1280 bytes for 10 phases. Total of 2KB of meta data
Outline:
Bringing energy awareness in application, OS and Middleware

A Application
B OS
C Middleware
Power Aware OS and Middleware

- Make it adaptive to respond to application requirements
  - and to dynamically smooth the imbalances between demand and availability of energy resource

- Use reflectivity enable dynamic scaling of data, choice of algorithms or parameters
  - (e.g., transcoding algorithms, compression parameters)

- Use brokerage service to negotiate quality demands
Enable Power/Energy Dialogue

- Application can tell OS task information (type, deadlines, WCET estimates)
- OS can update these estimates based on runtime conditions.
- To do this, we need
  - **API**: Provide ways by which Application, OS and Hardware can exchange energy/power and performance related information efficiently.
  - **Middleware**: Facilitate a continuous dialogue / adaptation between OS / Applications.
  - **HAL**: Facilitate the implementation of power aware OS services by providing a software interface to low power devices.
Power Aware Parts

- **PA-API (Power Aware API)**
  - interfaces applications and OS making the power aware OS services available to the application writer.

- **PA-OSL (Power Aware Operating System Layer)**
  - implements modified OS services and active components such as a DPM manager.

- **PA-HAL (Power Aware Hardware Abstraction Layer)**
  - interfaces OS and Hardware making the power control knobs available to the OS programmer.
OS Services

- **PA-API** - Power aware function calls available to the application writer.
  - Some functions of this layer are specific to certain scheduling techniques.
- **PA-Middleware** - Power aware services
  - Implemented on the top of the OS (power management threads, data handling, etc...).
- **POSIX** - Standard interface for OS system calls.
  - This isolates PA-API and PA-Middleware from OS.
- **PA-OSL** - Power aware OS layer.
  - Calls related to modified OS services should go through this level. Also isolates OS from PA-API and PA-Middleware.
- **PA-HAL** - Power Aware Hardware Abstraction Layer.
  - Isolates OS from underlying power aware hardware.
- **Modified OS services**
  - Implementation / modification of OS services in a power related fashion. Ex: scheduler, memory manager, I/O, etc.
## Layer Functionality

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PA-API</strong></td>
<td>paapi_dvs_create_thread_type(), paapi_dvs_create_thread_instance()&lt;br&gt;paapi_dvs_app_started(), paapi_dvs_get_time_prediction()&lt;br&gt;paapi_dvs_set_time_prediction(), paapi_dvs_app_done(),&lt;br&gt;paapi_dvs_set_adaptive_param()&lt;br&gt;paapi_dvs_set_policy(), paapi_dpm_register_device()</td>
</tr>
<tr>
<td><strong>PA-OSL</strong></td>
<td>paosl_dvs_create_task_type_entry(), paosl_dvs_create_task_instance_entry(),&lt;br&gt;paosl_dvs_killer_thread(), paosl_dvs_killer_thread_alarm_handler(),&lt;br&gt;paosl_dpm_register_device(), paosl_dpm_deamon()</td>
</tr>
<tr>
<td><strong>PA-HAL</strong></td>
<td>pahal_dvs_initialize_processor_pm(), pahal_dvs_get_frequency_levels_info()&lt;br&gt;pahal_dvs_get_current_frequency(), pahal_dvs_set_frequency_and_voltage()&lt;br&gt;pahal_dvs_pre_set_frequency_and_voltage(),&lt;br&gt;pahal_dvs_post_set_frequency_and_voltage()&lt;br&gt;pahal_dvs_get_lowpower_states_info(), pahal_dvs_set_lowpower_state()&lt;br&gt;pahal_dpm_device_check_activity(), pahal_dpm_device_pre_switch_state()&lt;br&gt;pahal_dpm_device_switch_state(), pahal_dpm_device_post_switch_state()&lt;br&gt;pahal_dpm_device_get_info(), pahal_dpm_device_get_curr_state()&lt;br&gt;pahal_battery_get_info()</td>
</tr>
</tbody>
</table>
DVS Related Functions

paapi_dvs_create_thread_type(), paapi_dvs_create_thread_instance()
creates type and instance of a task respectively

paapi_dvs_app_started(), paapi_dvs_app_done()
delimits execution of useful work in a thread. Tell the OS whether the task has finished execution or not.

paapi_dvs_get_time_prediction(), paapi_dvs_set_time_prediction()
get current execution time prediction for a given thread

paapi_dvs_set_adaptive_param()
set the parameters of the adaptive policy (it will be described later) for a given task.

paapi_dvs_set_policy()
chooses the policy to be using for DVS
DVS Related Functions (contd.)

paosl_dvs_create_task_type_entry(), ...
create a type and an instance of a thread in the kernel internal tables of type and instance respectively

paosl_dvs_killer_thread()
kills a thread that missed a deadline

pahal_dvs_initialize_processor_pm()
initialize structures for processor power management

pahal_dvs_get_current_frequency(),
pahal_dvs_set_frequency_and_voltage()
pahal_dvs_pre_set_frequency_and_voltage(),
pahal_dvs_get_frequency_levels_info()
pahal_dvs_post_set_frequency_and_voltage()
functions to switch processor among possible frequencies levels

pahal_dvs_get_lowpower_states_info(),
pahal_dvs_set_lowpower_state()
functions to switch processor among low power states
DPM Functions

- **paapi_dpm_register_device()**
  - just register the device to be power managed

- **paosl_dpm_deamon()**
  - implements the actual policy for a specific device. This daemon uses PA-HAL functions to decide on how to switch devices among all possible states.

- **pahal_dpm_device_switch_state()**
  - switch device's state

- **pahal_dpm_device_check_activity()**
  - check whether the device has been idle and for how long. This function needs support from the device driver.

- **pahal_dpm_device_get_info(), pahal_dpm_device_get_curr_state()**
  - gets information about the device and about its current state respectively

- **Others**
  - functions for helping implementing power policies. For example:
    - **pahal_battery_get_info()** – gets battery status
Prototype Implementation

- Platforms
  - eCOS RTOS:
    - open source, Object oriented and highly configurable RTOS (by means of scripting language)
  - Hardware platforms we are currently working with:
    - Linux-synthetic (emulation of eCos over Linux - debugging purposes only)
    - Compaq iPaq Pocket PC, Accelent IDP
Using Power Aware OS

- The scheduler adapts frequency according to the real time parameters passed in as parameter on the thread type.
- The frequency is adjusted by means of slowdown factors (a factor can also speed up the processor if it is > 1).

```c
void main()
{
    mpeg_decoding_t = paapi_dvs_create_thread_type(100,30,100,hard);
    paapi_dvs_set_policy(SHUTDOWN | STATIC
                         DYNAMIC | ADAPTIVE);
    paapi_dvs_create_thread_instance(     mpeg_decoding_t, mpeg_decode_thread);
}
```

- **void mpeg_decode_thread()**
  ```c
  {
      for (; ;) {
          paapi_dvs_app_started();
          /* original code */
          mpeg_frame_decode()
          paapi_dvs_app_done();
      }
  }
  ```
Directing DPM (and DVS)

```c
void threshold_policy_deamon(device_info_t dev) {
    unsigned idleness;
    for (;;) {
        /* check for how long the device has been idle */
        idleness = dev->check_activity(dev);
        /* if idle for longer than the threshold switch to next state */
        if (idleness > dev->check_state()->threshold) {
            dev->check_state()->switch_state(dev, dev->check_state,
                                              dev->check_state()->next);
        }
        /* sleep until next period for checking idleness */
        sleep(dev->policy_info->th_policy->period);
    }
}
```
OS-directed DVS Results

Energy Consumption for each scheme

<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>WCET (us)</th>
<th>Std Dev (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>MPEG2 (wg_gdo_1.mpg)</td>
<td>30700</td>
<td>3100</td>
</tr>
<tr>
<td>T2</td>
<td>MPEG2 (wg_cs_1.mpg)</td>
<td>26300</td>
<td>2100</td>
</tr>
<tr>
<td>T3</td>
<td>ADPCM</td>
<td>9300</td>
<td>3300</td>
</tr>
<tr>
<td>T4</td>
<td>FFT</td>
<td>15900</td>
<td>0</td>
</tr>
<tr>
<td>T5</td>
<td>FFT (gaussian dist.)</td>
<td>13600</td>
<td>800</td>
</tr>
</tbody>
</table>
Outline:
Bringing energy awareness in application, OS and Middleware

A Application
B OS
C Middleware
Energy Aware Middleware in FORGE

[Jointly with Nalini Venkatsubramaniam, UC Irvine.]

- Use reflective middleware services to continuously monitor and keep track of application needs
  - Use a rule base and directory service to carry out its functions.

- Example:
  - Multimedia streaming from a server to a set of mobile “nodes”
  - Use a proxy server to adapt video stream to specific nodes

- Node device: sends device info to proxy, connects video stream and network parameters to lower layers
- Proxy: admission control, real-time transcoding, network traffic regulation.
Cross-layer Energy Awareness

- Experiments using iPAQs demonstrate viability of dynamic adjustments of video quality
  - based on changing battery conditions and client devices.
  - [Mohapatra et al, ACM MM03]
Reducing Backlight for Lower Power

- **Identify “Groups of Scenes” with little variance in luminosity**

- **SBC: Simple Backlight Compensation**
  - Only identify GOS, reduce backlight on handheld
  - No video stream contrast enhancement

- **CBVLC: Constant Backlight with Video Luminosity Compensation**
  - Backlight level set once at start of video stream
  - Video stream is enhanced (dynamically at the proxy)

- **DCA: Dual Compensation Approach**
  - Backlight level is dynamically changed based on GOS
  - Video stream is enhanced based on Backlight level decision

<table>
<thead>
<tr>
<th>Backlight Modes</th>
<th>Power Consumed (in Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super Bright</td>
<td>2.80</td>
</tr>
<tr>
<td>High Bright</td>
<td>2.51</td>
</tr>
<tr>
<td>Medium Bright</td>
<td>2.32</td>
</tr>
<tr>
<td>Low Bright</td>
<td>2.16</td>
</tr>
<tr>
<td>Power Save</td>
<td>1.72</td>
</tr>
</tbody>
</table>

Power consumed at various backlight levels during streaming multimedia playback on the Compaq iPAQ
### Characteristics of video streams used in experiment

<table>
<thead>
<tr>
<th>MPEG Video</th>
<th>Resolution</th>
<th>FPS</th>
<th>Duration (sec)</th>
<th>Luminosity Variation</th>
<th>Video Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bipolar.mpg</td>
<td>320 x 240</td>
<td>30</td>
<td>41</td>
<td>Little</td>
<td>Dark, 3D animation</td>
</tr>
<tr>
<td>iceegg.mpg</td>
<td>240 x 136</td>
<td>30</td>
<td>59</td>
<td>Moderate</td>
<td>Bright, 3D animation</td>
</tr>
<tr>
<td>intro.mpg</td>
<td>160 x 120</td>
<td>30</td>
<td>59</td>
<td>Very High</td>
<td>Flashy, TV show clip</td>
</tr>
<tr>
<td>simpsons.mpg</td>
<td>192 x 144</td>
<td>30</td>
<td>27</td>
<td>High</td>
<td>Colorful, 2D animation</td>
</tr>
</tbody>
</table>

Snapshots of MPEG-1 video streams used in experiments
Results for Backlight Compensation

Super Bright

High Bright

Medium Bright

<table>
<thead>
<tr>
<th>Backlight Modes</th>
<th>Power Consumed (in Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super Bright</td>
<td>2.80</td>
</tr>
<tr>
<td>High Bright</td>
<td>2.51</td>
</tr>
<tr>
<td>Medium Bright</td>
<td>2.32</td>
</tr>
<tr>
<td>Low Bright</td>
<td>2.16</td>
</tr>
<tr>
<td>Power Save</td>
<td>1.72</td>
</tr>
</tbody>
</table>
New Computing Spaces & Fabrics

- New Computing Spaces challenge energy efficiency
  - Computers with radios, mobility

- New Computing Fabrics challenging architectures
  - To enable high hardware integration, and yet allow for reasonable software development across multiple heterogeneous processors
  - Yet, not fast enough. With the multiple processor cores on chip, the interaction of application behavior with core utilization will be even more critical to maintain energy profile.

- Awareness (environmental, location, QoS) may make it possible to achieve extreme dutycycles.

- Systematic understanding and use of awareness is crucial to engineering optimization.
Example

- **T₀**: 3 users (1, 2, 3)
- **T₁**: user 4 joins
  - System readjusts quality levels
- **T₂**: residual power on user 1 decreases
  - Quality level is decreased for 1
- **T₃**: user 5 joins
  - All levels go down to accommodate 5
- **T₄**: user 3 finishes streaming
  - Quality levels rise back
- **T₅**: user 1 finishes streaming
  - Even higher levels

**Video Quality**

- User₁
- User₂
- User₃
- User₄
- User₅

**Timeline**

- T₀: No optimizations
- T₁: With optimizations

Radu Cornea, UCI