Vertigo: Automatic Performance-Setting for Linux

Krisztián Flautner
ARM Limited, Cambridge, UK
Trevor Mudge
The University of Michigan

Presented by Choi Hojung
Background(1)

- In 2002
  - need for low power and high performance processors
  - from embedded computers to servers
  - high performance
  - battery operated
Background(2) – Intel SpeedStep

- SpeedStep by Intel
  - No built-in performance-setting policy
  - A simple approach by the usage model
  - When on AC power, processor runs at higher speed.
  - When on battery power, processor runs at a slower speed, thus saving battery power
Background(3) - LongRun

- LongRun for Crusoe™, by Transmeta
  - power management that dynamically manages the frequency and voltage levels at runtime
  - use historical utilization to guide clock rate selection
  - in processor’s firmware
  - interval-based algorithm
Background(4) - LongRun

- Flaws & Questions
  - in Processor’s firmware
  - utilization periods can be obscured when all tasks are observed in the aggregate
  - not have any information about interactive performance in operating system level
  - a single algorithm perform well under all conditions
Background(5) - DVS

• Dynamic voltage scaling (DVS)
  – also called Dynamic Voltage and Frequency Scaling (DVFS)
  – reduces the power consumed by a processor by lowering its operating voltage

\[ P = C \cdot f \cdot V_{DD}^2 \]

– P: power consumption
– V: the supply voltage
– C: the capacitance
– f: operating frequency
Proposed

• What Vertigo proposed
  – Implemented in OS kernel level to use a richer set of data for prediction
  – to reduce the processor’s performance level only when it is not critical to meeting the deadline
  – ensure good interactive performance without undue delay
Design(1) - Vertigo

- Vertigo: multiple performance-setting algorithms

- a decision hierarchy
  - high
    - TOP
    - the interactive algorithm
      - an algorithm for automatically quantifying the performance requirements of interactive apps
  - MIDDLE
    - the application specific layer
      - where DVS-aware applications can submit requirements
  - low
    - BOTTOM
    - the perspective algorithm
      - attempts to estimate the future utilization of the processor based on past information
Design(2) - Keeping track of work

• full-speed equivalent work done
  – to estimate how long a given workload would take running at the peak performance of a processor.

\[
W_{\text{fse}} = \sum_{i=1}^{n} t_i p_i
\]

– \(i\) : one of the \(n\) different performance levels during a given interval
– \(t_i\) : non-idle time spent at that performance level in seconds
– \(p_i\) : frequencies specified as a fraction of peak performance.
Design(3) – Perspectives-based algo

• a perspectives-based algorithm
  – at the lowest level in the policy stack
  – aims to derive a rough approximation for the necessary performance level of the processor
  – computes performance predictions from the perspectives of each task
  – uses the combined result to control the performance-setting of the processor
Design(4) - Perspectives-based algo

- Measuring the utilization for task A

  Task A's utilization is computed over this interval
  a. current state of work counter
  b. current state of idle time counter
  c. current time
  d. a run bit that task has started running

  the per-task data structures initialized
  a. current state of work counter
  b. current state of idle time counter
  c. current time

  \[ \text{Perf} = \frac{\text{WorkEst}}{\text{Deadline}} \]
Design(5) - Interactive applications

• Strategy for ensuring good interactive performance
  – to find the periods of execution that directly impact the user experience
  – to ensure that these episodes complete without undue delay

• How Vertigo detects interactive applications
  – interactive episodes signified by a GUI event
  – by monitoring the appropriate system calls
Design(6) - Interactive applications

- the strategy for setting the performance level during an interactive episode
Design(7) – Policy stack

• How policy stack works
  – a mechanism for supporting multiple independent performance-setting policies in a unified manner
  – a policy requests performance level with command
  – When a new level request arrives, commands on the stack evaluated bottom to up to compute
Evaluation(1) - MPEG

### Execution statistics

<table>
<thead>
<tr>
<th></th>
<th>Length (s)</th>
<th>Idle</th>
<th>Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Danse De Cable</td>
<td>247.1</td>
<td>54%</td>
<td>23%</td>
</tr>
<tr>
<td>320x160 + audio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LongRun</td>
<td>247.1</td>
<td>54%</td>
<td>23%</td>
</tr>
<tr>
<td>Vertigo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legendary</td>
<td>19.4</td>
<td>33%</td>
<td>13%</td>
</tr>
<tr>
<td>LongRun</td>
<td>19.4</td>
<td>33%</td>
<td>13%</td>
</tr>
<tr>
<td>352x240 + audio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertigo</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MPEG decode

<table>
<thead>
<tr>
<th></th>
<th>Ahead (s)</th>
<th>Exactly on time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Danse De Cable</td>
<td>148.10</td>
<td>6</td>
</tr>
<tr>
<td>320x160 + audio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LongRun</td>
<td>148.10</td>
<td>6</td>
</tr>
<tr>
<td>352x240 + audio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertigo</td>
<td>7.20</td>
<td>19</td>
</tr>
<tr>
<td>Legendary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Fraction of time at each performance level

<table>
<thead>
<tr>
<th></th>
<th>300 Mhz</th>
<th>400 Mhz</th>
<th>500 Mhz</th>
<th>600 Mhz</th>
</tr>
</thead>
<tbody>
<tr>
<td>LongRun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Danse De Cable</td>
<td>6%</td>
<td>19%</td>
<td>33%</td>
<td>54%</td>
</tr>
<tr>
<td>Legendary</td>
<td>0%</td>
<td>3%</td>
<td>17%</td>
<td>79%</td>
</tr>
<tr>
<td>Vertigo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean perf level</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LongRun</td>
<td>89%</td>
<td>51%</td>
<td>45%</td>
<td>0%</td>
</tr>
<tr>
<td>Danse De Cable</td>
<td>89%</td>
<td>51%</td>
<td>45%</td>
<td>0%</td>
</tr>
<tr>
<td>Legendary</td>
<td>96%</td>
<td>51%</td>
<td>45%</td>
<td>0%</td>
</tr>
<tr>
<td>Vertigo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Mean performance reduction over LongRun

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Danse De Cable</td>
<td>34%</td>
</tr>
<tr>
<td>Legendary</td>
<td>15%</td>
</tr>
<tr>
<td>Vertigo</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation(2) – interactive workload
Conclusions

• LongRun
  – in processor’s firmware
  – interval-based algorithm

• Vertigo
  – first approach to implement DVFS in SW level
  – multiple performance-setting algorithms

• LongRun vs Vertigo
  – Both implement DVFS technology
  – a 11%~35% average performance level reduction by Vertigo over LongRun
  – Vertigo can makes decisions based on a richer set of run-time information
  – LongRun has crucial advantage of being OS agnostic
Questions?