KOALA
A PLATFORM FOR OS-LEVEL POWER MANAGEMENT
D.C. Snowdon    E. Le Sueur    S.M. Petters    G. Heiser
KNOBS

- CPU frequency
- CPU voltage
- CPU sleep states
- Memory and bus frequency
- Power states of IO devices (not considered here)
SIMPLISTIC POLICIES
SIMPLISTIC POLICIES
REALITY CHECK

Figure 1. Normalised energy use of two benchmarks under DVFS on a Latitude laptop for run-time phase detection and prediction [Isci 2006]. We see this work on workload prediction as being highly complementary to our approach, since we presently employ a very simple workload predictor. In the same manner, machine learning techniques [Kephart 2007] could be used to improve the system’s predictive ability, as well as tune power and performance estimators using on-line feedback. [Mahesri 2004] found that a laptop CPU uses between 10 and 50% of the system’s power depending on workload. While the CPU is a significant contributor to overall power consumption, it does not necessarily dominate. Consequently, there has been work on developing complete system power models based on run-time statistics [Heath 2005, Economou 2006, Bircher 2007]. ECOsystem [Zeng 2005] was an attempt to build an explicitly energy-aware operating system, introducing a system-wide abstraction for the energy used. The purpose was to budget the energy available to individual processes. The models concentrated on I/O power and are therefore complementary to the work presented here. Virtualisation adds yet another dimension to energy and resource accounting [Stoess 2007].

Recently, the case has been made for improved hardware support for power management. [Barroso 2007] argued for lower-power idle modes, based on the observation that servers were nearly always less than 50% utilised. In addition, they pointed out the need for more active-power management mechanisms for devices such as memory, network cards and disks. NVIDIA have recently introduced such mechanisms [NVIDIA Corporation 2007]. Such active management features would result in systems with many interacting settings. We consider our energy-modelling approach as core to the effective management of such a system. [Peddersen 2007] investigated a methodology for detecting which events within a CPU should be used to estimate power consumption, which provides the basis for a hardware manufacturer to provide more suitable PMCs for energy estimation.

3. Power Management Challenges

Our investigation of a wide range of platforms and workloads demonstrated the shortcomings of commonly-used energy-management heuristics – they frequently fail to achieve their goals. Here we present our main observations, which were obtained with the methodology described in Section 5.

3.1 Workload dependence of DVFS response

Energy-management approaches are frequently based on simplifying assumptions which neglect the fact that the response to frequency scaling is highly dependent on workload characteristics. This can lead to very poor results, as shown in Figure 1. Here we compare the responses of the CPU-bound gzip and the memory-bound swim benchmark on a Dell Latitude D600 laptop. As discussed in the literature, the execution time of the CPU-bound program is proportional to the clock period (inverse frequency), while for the memory-bound program it is almost independent of CPU frequency. This results in the energy consumption shown in the figure: Total energy use for the CPU-bound benchmark is minimised by running at the highest frequency (race-to-halt works well) because this minimises the clock-independent memory energy and leakage losses in the CPU [Snowdon 2005]. In contrast, the memory-bound process minimises energy use at a low (but not the lowest!) frequency. Clearly, an approach that does not take workload characteristics into account will not be able to deliver a reasonable result for both programs.
Figure 3. Cycles vs. Frequency for various benchmarks on a Latitude D600 laptop

3.2 Multiple adjustable frequencies

Some platforms we have previously evaluated, such as those based on Intel PXA processors, allow multiple frequencies to be modified, typically CPU, bus(es) and memory. Different combinations of frequencies (which we call settings) lead to different results, as shown in Figure 2. This shows the execution time of gzip (which is memory-bound on this platform – the opposite of the D600!) on a PXA270-based machine for different frequency settings, showing the impact of memory frequency. CPU-bound applications are unaffected by memory frequency.

3.3 Variable memory system performance

The performance of the memory subsystem at a particular memory frequency may appear to depend on the core frequency [Kotla 2004]. This is a result of micro-architectural features, designed to improve performance, such as out-of-order execution or pre-fetching. These can hide some of the latency of cache misses (by overlapping them with instruction execution) but become less effective as the ratio of core to memory frequency increases. An example is shown in Figure 3. The behaviour of both the CPU-bound (gzip) and memory-bound (swim) benchmarks are well represented by straight lines (with swim extending above the upper bound of the graph), while intermediate workloads (especially mgrid) show significant non-linearity. This effect contributes to the error in our models, since the performance counters require to estimate these effects accurately are not available.

The memory configuration also has an effect on energy consumption as shown in Figure 4. The energy use for swim is quite different with and without dual-channel memory enabled in the system, which happens when adding a second memory DIMM. The figure also shows the effect of changing the memory frequency on the energy consumption — the irregular behaviour at 800MHz is due to a reduced memory frequency for that setting.
OVERHEAD

- switching frequency and power incurs CPU downtime
- Pentium-M
  - frequency change: 10µs
  - voltage change asynchronous
- Opteron
  - frequency and voltage change: 2ms (140µs out of spec)
3.6 Temperature effects

The temperature of the processor core affects the power consumption in two ways: leakage current is proportional to the voltage to another. These regulators often have a high efficiency, but that efficiency may depend on the operating conditions. During our experiments on the Latitude we ran into the perplexing situation where reducing the CPU frequency would increase the power drawn by the system. This was caused by a change in efficiency of the laptop's CPU power supply as the load changed, as shown in Figure 6. Such an effect provides a challenge to power management schemes based on simple analytical power models or heuristics. We investigated this issue further by examining the power consumption of a Dell Latitude D600 laptop running from the AC adapter.

Figure 6. System power vs. temperature for gzip at 600MHz on a Dell Latitude D600 laptop running from the AC adapter.

Figure 7. Actual vs. predicted input power for the Dell Latitude D600 laptop running from the AC adapter.

3.7 Frequency switching overheads

The platforms provide different levels of automation. On the Pentium-M the CPU is unavailable for the duration of the voltage switch. The overhead involved in these operations is highly hardware-specific.

On the Opteron the voltage must be scaled to the maximum before the frequency switch can be performed, and the only CPU downtime is during the frequency change. The overhead involved in these operations is highly hardware-specific.

The Pentium-M is fully hardware-sequenced, and therefore the time during which the CPU is unavailable during frequency switching varies considerably between platforms. Of the ones we tested it ranged from 10 to 50 microseconds for a so-called “turbo mode” switch.

The overhead is due to two operations — the voltage ramps which take 500 microseconds for a full frequency switch, compared with 20 microseconds (Pentium-M based) to 50 microseconds (Opteron) for switching from one such type of event, and, in a system with dynamic ticks, the scheduler clock ticks are asynchronously invoked, and the proportion of the number of clock ticks themselves contribute to the overall execution time.

Some events in the system occur at a rate that is not related to the system's active time spent processing those invocations. Therefore, running at a higher performance setting reduces both the number of scheduler invocations incurred, and the proportion of the system's clock frequencies. Scheduler clock ticks are asynchronous through 140 clock ticks, while really low power consumption, as embedded processors do. Note that even with a zero-power idle mode, race-to-halt is sub-optimal for all but the most CPU-bound applications.

3.8 Real-time dependencies

System power vs. temperature for gzip at 600MHz on a Dell Latitude D600 laptop running from the AC adapter.

TEMPERATURE

Input Power (W) vs. Temperature (degrees C)
### 3.5 Power-supply nonlinearities

Systems often use voltage regulators to convert from one voltage to another. These regulators often have a high efficiency, but that efficiency may depend on the operating conditions. During our experiments on the Latitude we ran into the perplexing situation where reducing the CPU frequency would increase the power drawn by the system. This was caused by a change in efficiency of the laptop's CPU power supply as the load changed, as shown in Figure 6. Such an effect provides a challenge to power management schemes based on simple analytical power models or heuristics. We worked around this issue by running the Latitude from its battery instead of the AC power adapter, however we expect other systems' energy efficiency to be affected by their power consumption. Martin's work regarding the efficiency of batteries [Martin 2001] clearly falls into a similar category. Our approach naturally deals with these effects, given an appropriate model for the power supply efficiency.

### 3.6 Temperature effects

The temperature of the processor core affects the power consumption in two ways: leakage current is proportional to the temperature of the silicon, and the power required for active cooling (fan) is significant. The result is that higher frequencies (which cause the system to run at a higher temperature) use disproportionately more energy, and that the relative energy benefits of the frequency setpoints change when the CPU is at an elevated temperature. This effect is shown in Figure 7.

### 3.7 Frequency switching overheads

The time during which the CPU is unavailable during frequency switches varies considerably between platforms. Of the ones we tested it ranged from 10 $\mu$s (Pentium-M based Latitude, not including the voltage change which happens asynchronously) through 140 $\mu$s worst-case for the Opteron. This is pure overhead, since the machine consumes energy without doing any useful processing. Some other platforms examined exhibited interesting features: the PXAs take 500 $\mu$s for a full frequency switch, compared with 20 cycles for a so-called “turbo mode” switch. The overhead is due to two operations — the voltage and the frequency change. The overhead involved in these operations is highly hardware-specific. The platforms provide different levels of automation. On the Opteron, software controls the voltage ramps, and so the CPU is unavailable for the duration of the voltage switch. The Pentium-M is fully hardware-sequenced, and therefore the only CPU downtime is during the frequency change. On the Opteron the voltage must be scaled to the maximum before the frequency switch can be performed, and subsequently to the target voltage. The period of CPU unavailability is then dependent on both the previous and next frequencies. For experimental purposes, the Opteron's DVFS driver was tuned to run faster than the specification, allowing the above worst-case performance of 140 $\mu$s. The worst case when run within the specification was $\sim 2$ ms.

### 3.8 Real-time dependencies

Some events in the system occur at a rate that is not related to the system's clock frequencies. Scheduler clock ticks are one such type of event, and, in a system with dynamic ticks, do not occur while in idle modes. Therefore, running at a higher performance setting reduces both the number of scheduler invocations incurred, and the proportion of the system's active time spent processing those invocations. The number of clock ticks themselves contribute to the overall running time.
TIME MODEL

\[ T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \ldots \]

\[ C_{bus} = \alpha_1 PMC_1 + \alpha_2 PMC_2 + \ldots \]
\[ C_{mem} = \beta_1 PMC_1 + \beta_2 PMC_2 + \ldots \]
ENERGY MODEL

\[ E_{dyn} \propto cyc \times V^2 \]

\[ E = V_{cpu}^2 (\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem}) \Delta t + \\
V_{cpu}^2 (\alpha_0 PMC_0 + \cdots + \alpha_m PMC_m) + \\
\gamma_4 f_{mem} \Delta t + \beta_0 PMC_0 + \cdots + \beta_m PMC_m + \\
P_{static} \Delta t \]
UNIFIED POLICY

\[ \eta = P^{1-\alpha} T^{1+\alpha} \]

forces highest performance
UNIFIED POLICY

\[ \eta = P^{1-\alpha}T^{1+\alpha} \]

minimises energy-delay product
UNIFIED POLICY

\[ \eta = P^{1-\alpha} T^{1+\alpha} \]

\( \alpha \)

minimises energy
UNIFIED POLICY

\[ \eta = P^{1-\alpha}T^{1+\alpha} \]

\( \alpha \)

minimises power consumption
IMPLEMENTATION

• recent Linux kernel (2.6.24.4)

• per-process collection of relevant statistics

• policy-decision when process blocks or preempts
  • use data from previous time slice to predict optimal setting
  • assumes temporal locality

• uses logarithmic tables to simplify calculation (no float)
EVALUATION

The Laptop

- Dell Latitude D600
- Pentium-M 0.8 – 1.8 GHz
- 0.98 – 1.34 V
- three sleep states
- measured at battery

The Server

- AMD Opteron 246
- 0.8 – 2 GHz
- 0.9 – 1.5 V
- high switching overhead
- measured at wall socket
CHARACTERISATION

The Laptop
- number of completed burst transactions
- number of lines removed from L2 cache
- correlation 0.98 / 0.96

The Server
- quadword write transfers
- L2 cache misses
- dispatch stalls due to reorder buffer being full
- DRAM accesses due to page conflicts
- correlation 0.98 / 0.98
MODEL ACCURACY

Figure 8. Koala behaviour for the first 1000 time slices of swim on the server with and without latency terms.

5.4 Model Accuracy

Figure 9 shows the performance and energy use of 27 SPEC CPU2006 benchmarks under the minimum-energy policy $\alpha = 0$ in Equation 5) on the server. The benchmarks omitted for space reasons are all CPU-bound and thus uninteresting for this platform. The energy saving is between 0 and 15% of the total system energy. The latitude showed even more significant energy savings (see Figure 11). For some benchmarks (the memory-bound swim) Koala was able to save 29% of the energy for only a 3% loss in performance at the minimum energy setting. This is an estimated 45% saving of the dynamic energy (estimated by subtracting the idle power).

For most benchmarks there is good agreement, generally within a few percent, between the actual performance and energy use and the estimates produced by our model, which indicates that the approach generally works well. However, there is a single case where the model fails spectacularly, mispredicting performance of the LBM benchmark by 25% (107 vs. 86) and energy by 20% (68 vs. 85). The system still saves energy on this benchmark — while the models fail to predict accurately, they still provide a good heuristic for frequency selection. More accurate models would allow more reliable, predictable energy savings. LBM was the only such case we observed where the models failed in this way.
MODEL ACCURACY

5.4 Model Accuracy

Figure 9 shows the performance and energy use of 27 SPEC CPU2006 benchmarks under the minimum-energy policy $\alpha = 0$ in Equation 5) on the server. The benchmarks omitted for space reasons are all CPU-bound and thus uninteresting for this platform. The energy saving is between 0 and 15% of the total system energy. The latitude showed even more significant energy savings (see Figure 11). For some benchmarks (the memory-bound swim) Koala was able to save 29% of the energy for only a 3% loss in performance at the minimum energy setting. This is an estimated 45% saving of the dynamic energy (estimated by subtracting the idle power).

For most benchmarks there is good agreement, generally within a few percent, between the actual performance and energy use and the estimates produced by our model, which indicates that the approach generally works well. However, there is a single case where the model fails spectacularly, mispredicting performance of the LBM benchmark by 25% (107 vs. 86) and energy by 20% (68 vs. 85). The system still saves energy on this benchmark — while the models fail to predict accurately, they still provide a good heuristic for frequency selection. More accurate models would allow more reliable, predictable energy savings. LBM was the only such case we observed where the models failed in this way.

5.5 Policies

Figure 10 shows how Koala implements the maximum-degradation policy (see Section 4.6). Curves in the top graph show the actual performance of representative benchmarks under varying performance goals. The thick diagonal line represents the ideal response, under perfect operation all curves should be on or just above this line. It can be seen that actual performance mostly gets close to the target. Some benchmarks run at slightly less than the...
UNIFIED POLICY

Figure 10. Maximum-degradation policy on the Latitude target performance, this results from the discrete setpoints, inaccurate performance estimation, and Koala's adjustments lagging behind changes of workload behaviour.

The horizontal lines extending to the left of the graph are a result of the limited frequency range available — the processor cannot be throttled well enough to reach the lower performance targets. This effect is particularly strong for the memory-bound benchmarks.

The bottom graph in Figure 10 shows the corresponding energy use. We can see that the maximum-degradation policy saves significant energy (up to about 25%) on memory-bound benchmarks, but actually wastes energy on CPU-bound benchmarks, clearly indicating that this policy is not suitable for a wide range of workloads.

The reason is that a CPU-bound benchmark executes in a constant number of cycles, irrespective of the core frequency. Lower frequency leads to a longer overall execution time, which increases the static energy components (leakage currents in the processor and memory). This is the effect shown in Figure 1, which indicates that race-to-halt is the best policy for CPU-bound workloads.

Figure 11 shows that our generalised energy-delay policy produces much better results. As expected, $\alpha = 1$ yields the highest performance while $\alpha = 0$ produces the lowest energy consumption (with a slight aberration of the pathological $lbm$ benchmark), and intermediate values produce intermediate results. The graphs also show that the standard energy-delay policy ($\alpha = 0.33$) produces, for most benchmarks, an energy use close to that of the minimum-energy setting, for a moderate performance degradation. Negative values of $\alpha$ are not useful for energy management, although small negative values can be used to throttle power dissipation for thermal management.

Figure 11 also shows that some benchmarks, specifically the notorious $lbm$, fail to reach more than about 90% performance at $\alpha = 1$. This is obviously a result of incorrect performance estimates leading Koala to choosing an incorrect setting. (This is confirmed by $lbm$ also failing to reach its maximum-frequency energy use at $\alpha = 1$).

The strength of the generalised energy-delay policy with its single global parameter is particularly evident when comparing the CPU-bound $povray$ with the memory-bound $milc$ (Figure 12). $povray$ is not slowed down at all for positive $\alpha$, since there is no energy to save. For the same $\alpha$ values, $milc$ is scaled in order to save energy. The policy only sacrifices performance when there is a corresponding energy benefit. Below $\alpha = 0$, $povray$ is scaled aggressively to re-
UNIFIED POLICY

Figure 10. Maximum-degradation policy on the Latitude.

The horizontal lines extending to the left of the graph are a result of the limited frequency range available — the processor cannot be throttled well enough to reach the lower performance targets. This effect is particularly strong for the memory-bound benchmarks.

The bottom graph in Figure 10 shows the corresponding energy use. We can see that the maximum-degradation policy saves significant energy (up to about 25%) on memory-bound benchmarks, but actually wastes energy on CPU-bound benchmarks, clearly indicating that this policy is not suitable for a wide range of workloads.

The reason is that a CPU-bound benchmark executes in a constant number of cycles, irrespective of the core frequency. Lower frequency leads to a longer overall execution time, which increases the static energy components (leakage currents in the processor and memory). This is the effect shown in Figure 1, which indicates that race-to-halt is the best policy for CPU-bound workloads.

Figure 11 shows that our generalised energy-delay policy produces much better results. As expected, $\alpha = 1$ yields the highest performance while $\alpha = 0$ produces the lowest energy consumption (with a slight aberration of the pathological lbm benchmark), and intermediate values produce intermediate results. The graphs also show that the standard energy-delay policy ($\alpha = 0.33$) produces, for most benchmarks, an energy use close to that of the minimum-energy setting, for a moderate performance degradation. Negative values of $\alpha$ are not useful for energy management, although small negative values can be used to throttle power dissipation for thermal management.

Figure 11 also shows that some benchmarks, specifically the notorious lbm, fail to reach more than about 90% performance at $\alpha = 1$. This is obviously a result of incorrect performance estimates leading Koala to choosing an incorrect setting. (This is confirmed by lbm also failing to reach its maximum-frequency energy use at $\alpha = 1$).

The strength of the generalised energy-delay policy with its single global parameter is particularly evident when comparing the CPU-bound povray with the memory-bound milc (Figure 12). povray is not slowed down at all for positive $\alpha$, since there is no energy to save. For the same $\alpha$ values, milc is scaled in order to save energy. The policy only sacrifices performance when there is a corresponding energy benefit. Below $\alpha = 0$, povray is scaled aggressively to res-
BATTERY-AWARE POLICY

Figure 12. Generalised energy-delay policy on the server.

Reduce the system power consumption, but with a corresponding increase in energy used.
The policy applies equally well when the model includes the system's idle energy, and fairly trades performance and execution in this different context. Again, we demonstrate the idle energy models using well-behaved benchmarks.

Enabling the switch overhead model, we see the number of frequency switches reduced for most policies and benchmarks (in the case of swim on the server, this is about 9%) because the model predicts a higher performance for the incumbent frequency, which it therefore favours slightly. We also see the energy savings and model accuracy increase when using these models. We use well-behaved benchmarks here to highlight the effect of the switch overhead model.

5.6 Multi-tasking

Figure 13 shows the effect of running a multi-tasking workload consisting of memory-bound swim and CPU-bound gzip. The top part of the figure shows that the energy and performance predictions of the combined workload under the minimum-energy policy is about as good as for separate executions, and the energy saved is about the average of the two individual loads, as can be expected. The bottom graph shows how Koala adapts the setting for the two processes independently.

5.7 Higher-level Policies

One advantage of the generalised energy-delay policy is that the single parameter ($\alpha$), allows the system to adapt to changing energy-management objectives. As a demonstration we implemented a daemon which monitored the laptop's battery state of charge using ACPI. At capacities greater than 70%, the daemon sets $\alpha$ to 1, and the system runs at maximum performance. As the battery is depleted, the daemon lowers $\alpha$ until the battery gets below 0.6.

Figure 14. Using the Latitude's battery state of charge to drive the power management policy.

Another high-level policy on top of the generalised energy-delay policy emulates the ondemand governor in Linux: CPU scaling is based on the available idle time. During periods of low utilisation, $\alpha$ is lowered towards 0 (the minimum energy setting), and in times of high load, $\alpha$ is increased toward 1.0 (the maximum performance setting).

5.8 Calculation overheads

A major concern when developing Koala was the overhead introduced, since this could reduce the energy savings and be detrimental to performance. In order to minimise the...
DISCUSSION

- practicality
- cooperation from vendors, built-in power measurement
- energy management by hardware or software
- hints from applications
- is this too fine-grained
- dumb component shutdown (both software and hardware)