Koala
A Platform for OS-Level Power Management

David C. Snowdon[12] Etienne Le Sueur [12] Stefan M. Petters [12] Gernot Heiser [123]

[1]The University of New South Wales [2]NICTA[3]Open Kernel Labs
Sydney, Australia
David.Snowdon@nicta.com.au

Abstract

Managing the power consumption of computing platforms is a complicated problem thanks to a multitude of hardware configuration options and characteristics. Much of the academic research is based on unrealistic assumptions, and has, therefore, seen little practical uptake. We provide an overview of the difficulties facing power management schemes when used in real systems.

We present Koala, a platform which uses a pre-characterised model at run-time to predict the performance and energy consumption of a piece of software. An arbitrary policy can then be applied in order to dynamically trade performance and energy consumption. We have implemented this system in a recent Linux kernel, and evaluated it by running a variety of benchmarks on a number of different platforms. Under some conditions, we observe energy savings of 30% for a 4% performance loss.

Categories and Subject Descriptors D.4.8 [*Operating Systems*]: Performance Modelling and prediction

General Terms Design, Experimentation, Measurement

Keywords Power, Energy, Efficiency, Modelling, Dynamic voltage scaling, Power management, Operating systems

1. Introduction

Managing power consumption of computing platforms is becoming increasingly important. On the one hand, power is becoming important for servers. Given the cost of power for computation and cooling [Scaramella 2006] there is increasing interest in reducing their energy consumption, and power dissipation needs to be managed in order to prevent the cores from overheating [Kumar 2006]. On the other hand, the ubiquity of battery-powered embedded devices, such as mobile phones, is driving interest in energy management in order to maximise battery life time [Martin 2001]. While superficially similar, the different scenarios have different requirements and constraints, which require different policies.

Properly managing power is difficult for a number of reasons. One is the availability of a range of hardware mechanisms, which can be used to influence power consumption. A potentially effective mechanism is *dynamic voltage and frequency scaling* (DVFS), which is based on the fact that the dynamic power dissipated by a CMOS circuit is proportional to the clock frequency and the square of the supply voltage. Then there are idle (or "sleep") states with reduced power consumption available for CPU, memory and peripherals. The number of available mechanisms presents a challenge to effective power management, as each has a different impact on power consumption and different trade-off between performance and energy use. Furthermore, the impact on power usage as well as performance is generally non-linear, platform-dependent and workload-dependent, and at times, counter-intuitive.

Power management in present main-stream operating systems tends to be simplistic. Standard policies are either the "race-to-halt" approach, which runs the workload to completion at the maximum performance setting and then transitions into a low-power mode. Alternatively, the assumption is made that the highest energy savings can be achieved by running at the lowest performance setting. We show that such simplistic approaches lead to sub-optimal results on real hardware.

In order for the OS to manage power effectively it must be possible to predict the impact these mechanisms have on both performance and power. This implies the need for an accurate model of how the hardware mechanisms impact on performance and energy consumption, which can then be used to adjust the operating conditions to meet the performance or energy targets defined by the system's power-management policy.

We had earlier developed an approach that allows us to generate such a model, taking into account the properties of the hardware platform. The model can then be used to predict the power and performance response of a workload [Snowdon 2007].

Based on that model we developed Koala[1], a platform that lets the OS manage power according to an overall policy guided by predicted power and performance. Koala collects per-process performance statistics that characterise an application's behaviour. At each scheduling event, the behaviour is matched against the system policy and the most appropriate operating conditions are determined. In general this means that the power settings change at each time slice, according to the characteristics of the process that is being scheduled. Koala also provides accurate per-process accounting of energy use. This can be used to implement other policies, such as overall or per-process energy budgets, or limiting the average power use over a certain time window.

We have implemented Koala in Linux and evaluated its operation on a range of platforms, from embedded processors typical for mobile phones up to high-end server platforms.

This paper makes the following contributions:

- an investigation of factors affecting power use on a range of platforms;
- a generic model that accurately predicts for each process the performance and energy response when changing power settings;
- an improved energy-management policy, called the generalised energy-delay policy, which incorporates previously-used policies and provides a single parameter for tuning the system to an overall energy-management objective;
- an implementation of the model in Linux, called Koala, that manages power near-optimally for multi-tasking workloads on a range of platforms.

The remainder of this paper is structured as follows. In Section 2 we discuss related work. Section 3 discusses, based on measurements on actual computing systems, the challenges facing OS-level power management. In Section 4 we discuss Koala, our implementation of a model-based power-management platform in Linux. Section 5 presents an evaluation of Koala on several hardware platforms and Section 7 presents the conclusions we draw from the results.

2. Related Work

The energy savings possible via DVFS have been under active investigation since the pioneering work of Weiser et al. [Weiser 1994], who used the actual system utilisation in a feedback loop to adjust the system frequency setting in order to minimise idle time. An empirical evaluation of DVFS algorithms concluded that the algorithms used at the time did not save significant amounts of energy[Grunwald 2000].

[1] The Koala is a marsupial native to Australia known for its extremely efficient energy management.

[Miyoshi 2002] showed that the decrease in idle time resulting from execution at lower frequency can offset any savings from DVFS. That work, however, was based on a simplistic execution-time model. Our prior work has shown that the most energy-efficient setting is dependent on workload characteristics, and a good energy-management policy requires on-line characterisation of workloads [Snowdon 2005].

[Bellosa 2000] correlated hardware *performance monitoring counters* (PMCs) with energy consumption, obtaining an estimate of energy use at the current frequency setting. [Weissel 2002] then used performance counters for instructions, memory-accesses and CPU cycles to index a pre-computed table of frequency settings, with the aim of minimising energy use with a 10% performance degradation. [Bircher 2005] used a similar technique on the Pentium 4, which has many PMCs.

Other research (including our own) has led to more flexible methods of selecting the frequency which gives a specified performance loss [Hsu 2004, Kotla 2004, Choi 2005, Snowdon 2007]. By limiting the change in total execution time, these methods implicitly limit the change in energy used by the rest of the system, concentrating on savings in CPU power. Our previous work [Snowdon 2007] extended PMC-based energy estimation for predicting performance and energy use at different frequencies, and incorporated memory power. We also presented a systematic way for selecting the most appropriate PMCs to use on a particular platform.

[Kotla 2004] observed that the performance of the memory subsystem can vary thanks to the effectiveness of micro-architectural techniques like pre-fetching and instruction-level parallelism.

DVFS policies too numerous to discuss have been developed. The policy developments attempt to minimise the performance loss (how much longer software takes to execute), minimise the energy for a given workload, or minimise the energy-delay product (which places equal value on both energy savings and performance loss) have been presented. [Pénzes 2002] introduced the idea of geometrically weighting the $E \times D$ product to provide a variable weighting on energy and delay. We have extended this concept for a more flexible use in the context of DVFS.

Predicting the workload to be executed plays a crucial role in all on-line DVFS methods. [Hsu 2003] implemented DVFS-aware compiler techniques which analyse code to be run and insert DVFS calls. This has the advantage of being able to synchronise DVFS scaling with real workload changes. As we have demonstrated, off-line information alone is insufficient to choose the optimal frequency setting. The information provided by these analyses would, however, enhance the energy-saving ability of our system when available (with our present method used for non-DVFS-aware workloads) by allowing improved workload prediction and frequency-switch timing. Others have developed algorithms

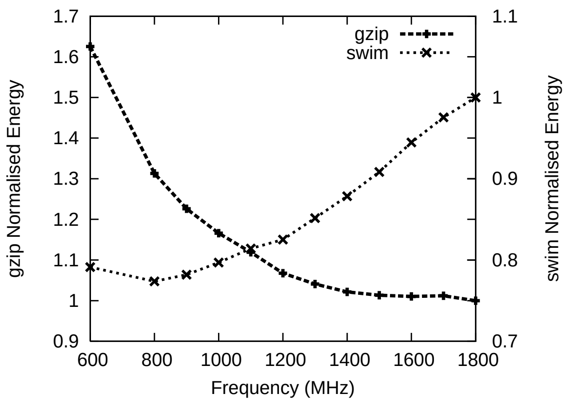


Figure 1. Normalised energy use of two benchmarks under DVFS on a Latitude laptop

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.

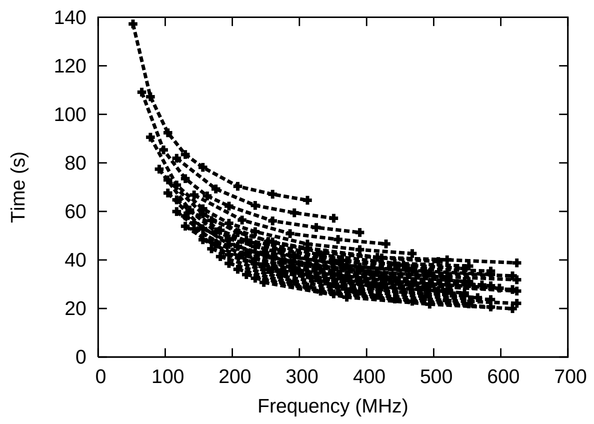


Figure 2. Performance of a memory-bound application (`gzip`) under frequency scaling on a PXA270-based platform. Lines connect settings with the same memory but different core frequencies.

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 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.

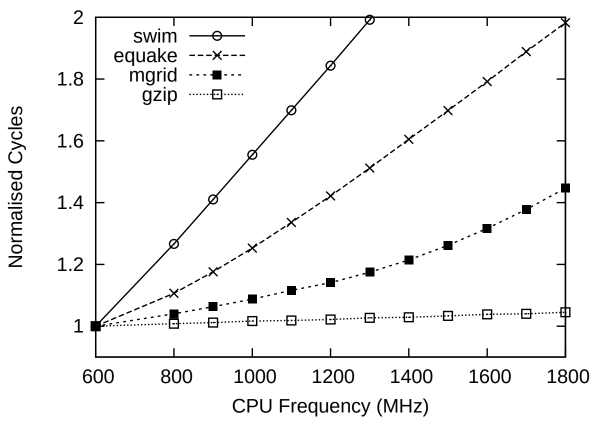


Figure 3. Cycles vs. Frequency for various benchmarks on a Latitude D600 laptop

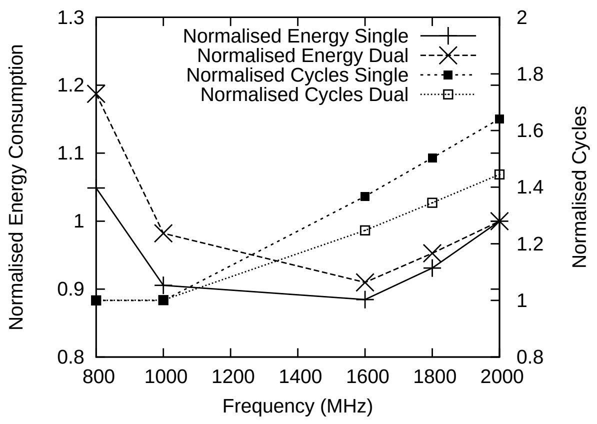


Figure 4. Comparison of cycles and energy use on an AMD64 Server with and without dual channel memory for `swim`.

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 required 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.

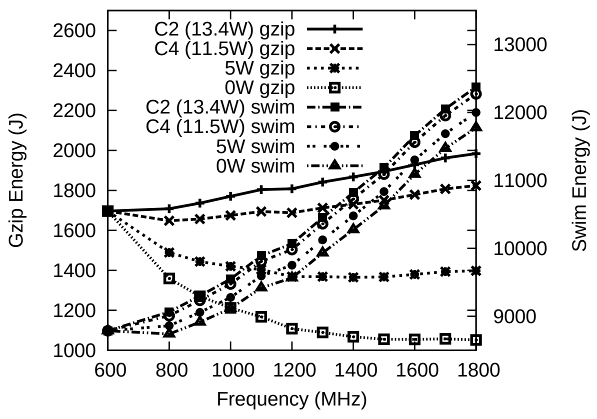


Figure 5. Total energy for the CPU-bound `gzip` and memory-bound `swim` application on the Latitude, using different idle states.

3.4 Idle modes

Reducing the frequency (and thus performance) reduces idle time, unless there is no idle time (as in heavily loaded servers). Modern CPUs have low-power modes which the OS can invoke when the system is idle. Idle modes still consume power in most cases, and take time to enter and exit (the deeper, i.e. lower-power, the idle mode is, the more time it takes to enter and exit). As discussed by Miyoshi [Miyoshi 2002], the power consumed while idle must be taken into account if overall energy saving is the goal.

Figure 5 shows total energy consumption when accounting for the idle energy used in various low-power modes during the faster runs (for a total time equal to the slowest run time). Besides the actual low-power modes supported ("C states"), we also show hypothetical 5W and 0W states (active power is 22–30W). We see again a significant difference between memory-intensive and CPU-intensive pro-

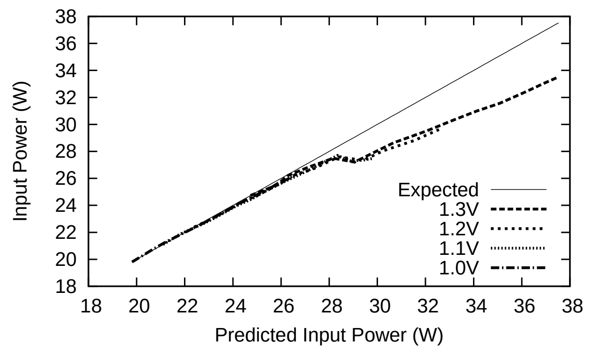


Figure 6. Actual vs. predicted input power for the Dell Latitude D600 laptop running from the AC adapter.

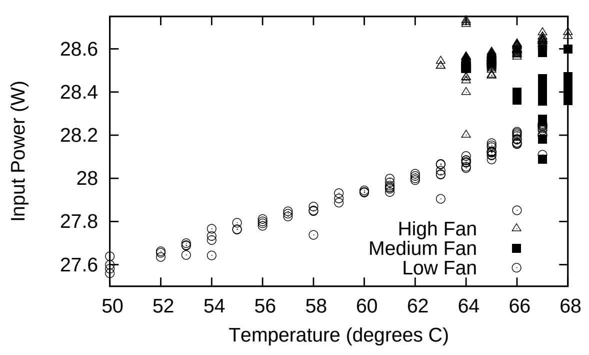


Figure 7. System power vs. temperature for gzip at 600MHz on a Dell Latitude D600

cesses, although for the hardware-supported idle modes, running at the lowest frequency always results in the lowest total energy on this platform. As the hypothetical states demonstrate, this will change once the hardware offers idle states of 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.

3.5 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 10μs (Pentium-M based Latitude, not including the voltage change which happens asynchronously) through 140μs 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 500μs 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 140μs. The worst case when run within the specification was $\sim$2ms.

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.

Specifically, a timer tick frequency of f_{tick} and a tick processing overhead of C_{tick} cycles will lengthen the interrupt-free execution time T_{work} of a workload to T_{tot} according to

$$\frac{T_{tot}}{T_{work}} = \frac{1}{1 - C_{tick} f_{tick} / f_{cpu}}. \tag{1}$$

For all the platforms and operating system we tested, $f_{cpu} \gg C_{tick} f_{tick}$ and thus the effect was negligible.

4. Koala Power-management Framework

As argued in the introduction, effective power management needs to be able to predict the system's power and performance response to the use of the hardware's power controls. Given the complexities outlined above, such a model must be specific to the hardware platform, and must take into account the characteristics of the workload. As such characteristics change over time, the model must adapt at run time.

Furthermore, there is significant benefit from separating the mechanisms enabling power management (including the modelling of application behaviour) from the policies. The choice of policy depends not only on a system's purpose (managing power on a server vs. energy on a battery-powered device) but may even change over time (dependent on load, battery charge or real-time deadlines) and may need to be integrated with other OS policies.

The approach taken in Koala therefore cleanly separates the modelling from the management. The model is specific to the hardware platform and provides an abstraction of platform details. The management component provides a generalised policy that includes commonly-used policies as special cases, and can be adapted at run time.

4.1 Basic model

The basic model, originally developed on PXA-based embedded platforms, has been presented previously [Snowdon 2007] and is summarised below.

In the following we normalise application *performance* and *energy* use to their values at the maximum frequency (i.e. highest performance) setpoint. Hence a performance of 90% means that the execution time is $1/0.9 = 1.11$ times the minimum. Similarly, 110% energy use means 10% higher than at the maximum frequency. Obviously, normalised performance never exceeds 100%, while normalised energy can be larger or smaller than 100% (as in Figure 5).

Our approach represents a program's execution time and total energy use as a linear combination of quantities measurable at run time. In the case of execution time, these are the inverses of the various clock frequencies (CPU, busses, memory), and can easily be extended to multiple clock domains, as long as their parameters are accessible to the OS. The energy model contains static energy (static power times execution time), dynamic energy terms (clock frequency times square of the respective voltage), other voltage- and frequency-dependent terms, as well as performance counters which measure energy-relevant events (see Section 5.2 for details).

4.2 Model Extensions

Some platforms, such as the Dell Latitude D600, have a thermal sensor which can be read by the operating system. This affects the model and should be included, as the leakage power is proportional to temperature. However, on the Latitude D600, the overhead of reading the sensor is substantial, and omitted in the results included here (the experiments were run with a constant ambient temperature). Other OS-observable factors could also be included, such as speed of the CPU's cooling fan. We expect that this approach is also suitable for including other factors, such as interrupts and DMA [Bircher 2007]. However, detailed modelling of I/O is beyond the scope of this paper, having been addressed to some degree by prior work (e.g. [Zeng 2005]).

The energy used during any extra idle time can be added. This makes sense in the case where the system is under-utilised, or where the system will not be shut down following a period of work. In this case, running at a higher-than-minimal frequency leads to more time spent in an idle mode. We therefore add this extra energy into the model as

$$E_{total} = E_{active} + (T_{max} - T) P_{idle}. \tag{2}$$

We also added the frequency-switch overhead (see Section 3.7) when predicting the performance during the next time slice, which has the effect of giving a slight preference not changing the frequency from the present setting (which in general has been selected for a different process since frequency selection happens at the beginning of a time slice). We do not make any assumption that a switch may be amortised over several time slices. On the laptop, the model is trivial — a constant $10\mu s$ overhead per switch. On the server, a more complex model is required:

$$T_{switch} = \frac{2V_{max} - V_{current} - V_{next}}{V_{step}} T_{step} + T_{lock}, \tag{3}$$

where V_{step} is the size of the step in the ramp, T_{step} is the latency incurred at each step and T_{lock} is the PLL locking delay. We make the approximation that the power remains constant during the switch.

4.3 Platform characterisation

While modern processors provide performance counters for measuring a large number of different events, the number of events that can be counted concurrently is typically quite small (2–4). Hence we need an approach to determine the most appropriate counters. We also need a systematic way of determining the appropriate weights of all the observed quantities.

We solve this by running a representative set of benchmarks (the *platform characterisation workloads*) on the system. The set is chosen to encompass a wide range of characteristics, in order to represent any future workload.

Each benchmark is run individually with no other activities on the system, once for each frequency setting, while measuring execution time and energy consumption (using a wattmeter). We also collect all available performance-counter events, which requires many identical runs per benchmark program and per setting in order to sample all counters (but this effort can be reduced somewhat, see Section 5.1.2).

We then perform an exhaustive analysis of the results to determine the set of performance counters that gives the most accurate prediction of energy use across the whole characterisation workload and all settings. The regression also produces the correct weights without making arbitrary assumptions.

Assuming this model is good, it should allow us to predict the response of a workload to changes in operating conditions. Specifically, from observing the performance and energy use at one DVFS setting, it should allow us to predict performance and energy use at a different setting. We confirmed this with a separate set of benchmarks (the *validation workloads*). On the embedded platform, we found that we could predict performance at any other setting with an average error of less than 2%, and energy use with an average error of around 6% [Snowdon 2007].

4.4 Idle power characterisation

On many systems the power and transition latency for C states are stored by the manufacturer in ROM (accessible via ACPI). When this information is not available, or the system is in a different state from that measured by the manufacturer (e.g. a different screen brightness setting), or one simply does not trust ACPI, the idle powers can be characterised by running benchmarks that put a system with no foreground activities to sleep for varying periods of time. A complication is that the OS uses some policy for choosing idle modes as a function of sleep time, which we did not want to interfere with. Instead we used the OS's accounting of time spent in various sleep states, and used linear regression on the average idle power

$$P_{ave} = \sum_i \frac{T_{C_i}}{T_{total}} P_{C_i}, \qquad (4)$$

where T_{C_i} is the time spent in idle state C_i, $T_{total} = \sum_i T_{C_i}$, and P_{C_i} the power drawn in state C_i. We avoid having to deal with a potential frequency and voltage dependence of the idle power by always switching to the lowest frequency when idling. We have not yet integrated into Koala a prediction of the amount of time spent in each idle state.

4.5 Selection of settings

During normal execution time we collect the relevant statistics separately for each process. Each time a process blocks or is pre-empted, we use the power and performance models to estimate and record its energy usage during the last execution period. The next time the process is scheduled, assuming temporal locality, we use the data gathered during the previous time slice to determine the optimal setting according to the system's (then active) power-management policy.

Even in the worst case, the frequency switch overhead is a small percentage of the timeslice (about 1.5%, for the Opteron), and therefore could be ignored in many cases. However, modelling the switch overhead according to Section 4.2 allows reducing the number of frequency switches, resulting in better performance and energy savings.

4.6 Policy

We implemented two policies which between them include and generalise all policies we have found in the literature. The first, which we call the *maximum-degradation policy*, chooses the lowest frequency which keeps (estimated) performance above a certain threshold. A threshold of 90% was empirically found to give reasonable results [Weissel 2002].

The second policy, which we call the *generalised energy-delay policy*, minimises the quantity

$$\eta = P^{1-\alpha} T^{1+\alpha}, \qquad (5)$$

where P is the power consumption, T the execution time (inverse performance) and α is a parameter that can be varied between -1 and 1. Special choices of α result in a number of particular policies found in the literature. Specifically:

$\alpha = 1$ maximises performance (forces highest frequency)

$\alpha = 0$ minimises energy ($E = PT$)

$\alpha = -1$ minimises power consumption

$\alpha = 1/3$ minimises the energy-delay product [AbouGhazaleh 2008] $ET = PT^2$.

Other values map to other policies used in the literature [Pillai 2001, Pénzes 2002]. Our approach allows the OS to easily adapt the power-management policy to changed operating conditions.

Note that $0 < \alpha < 1$ will throttle different processes differently, depending on their characteristics: memory-bound processes will execute at lower frequency than CPU-bound processes, as the former achieve relatively high energy savings for relatively low performance degradation.

In order to minimise run-time costs and avoid floating-point arithmetic in the kernel, a more suitable representation of the policy function is

$$\log_2 \eta = (1-\alpha)\log_2 E + 2\alpha \log_2 T. \qquad (6)$$

This can be implemented in fast fixed-point arithmetic using the `clz` instruction and a look-up table. Since $\log$ is a monotonic function, minimising $\log_2 \eta$ also minimises η.

5. Evaluation

5.1 Experimental Procedure

5.1.1 Platforms

We evaluated Koala on ten different platforms, ranging from ARM-based embedded systems platforms (PXA255/270, iMX31) to laptop-class systems (using Atom or Pentium-M processors), to server-class computers (Opteron, Xeon). Due to space limitations we focus on the two platforms producing the most interesting results, representative of both the strengths and limitations of our approach. These are a laptop and a server, both x86-based.

Some platforms are less interesting because they have long frequency switching delays, and thus gain only limited benefit from our methodology. Others were dominated by CPU external power consumption like network devices or screen and thus enable only marginal benefits from frequency scaling. Others were uninteresting in that a small ratio of CPU to memory speed makes most benchmarks CPU-bound.

All experiments were conducted in an air-conditioned environment to eliminate large temperature variations.

The laptop is a Dell Latitude D600 [Del 2004] based on an 1.800 GHz Pentium-M processor paired with the Intel 855PM chipset. It has 1GB of DDR266 memory with a 133 MHz clock rate, and a frontside bus frequency of 100 MHz but quad pumped to 400 MHz. The core frequency clock is varied from 0.8 to 1.8 GHz in 100 MHz steps and the core voltage is varied from 0.98 V to 1.34 V. We switch frequencies by accessing the respective registers directly rather than via ACPI, as ACPI did not export all possible frequencies/voltages. The LCD backlight was switched off to reduce the system's static power, and improve savings gained from the CPU for the experiments.

In order to help keep each benchmark run deterministic and consistent with the others, the network interface was removed from the kernel during measurements. This reduces any unpredictable wakeups because of network interface interrupts. The parallel port was used for triggering energy measurements and the serial port was used as a console terminal.

The Pentium M has a cycle counter and two user-configurable counters which can each measure one of several hundred different events, of which we used 164 potentially-relevant events to create our models. During a frequency switch, the CPU is unavailable for $10\mu s$ while the frequency-synthesis circuitry (PLL) re-locks. The voltage is automatically ramped up and down by the hardware prior to or following the frequency switch, respectively. This means that for a short period of time following the switch request, the processor may operate at a frequency different than the requested one.

The system has three sleep states which were measured and use 18.5 W in C2, 13.1 W in C3 and 11.1 W in C4.

Power consumption was measured in the battery supply line using a custom-designed device (called Echidna). Measuring at the battery reflects the importance of minimising the battery energy use (rather than wall-socket energy) while disconnected from an external supply. The Echidna has a 5 mW accuracy, sampling at 4.7 kHz. It has an on-board microcontroller which integrates the power to obtain the energy over a period, triggered by the parallel port of the machine under test.

The server is based on an AMD Opteron 246 processor clocked at 2 GHz. Using a custom driver, we were able to put the CPU in five different settings ranging from 0.8 GHz at 0.9V to 2 GHz at 1.5 V.

The system has 512 MB of DDR400 SDRAM, and while the memory frequency was fixed at 200 MHz, we noted that the memory controller changed the memory bus frequency down to 160 MHz at the lowest CPU frequency of 800 MHz; this was built into the model for the platform.

The overhead for a switch on this platform varies depending on the current and target frequencies, ranging from as low as $15\mu s$ when dropping the frequency from 2 GHz to any value, to $140\mu s$ when increasing from the minimum to the maximum frequency. This is due to delays while ramping the core voltage to the required level, and then the re-locking delay incurred by the PLL of the clock generator.

The processor has a cycle counter and four user-configurable counters which, like the Latitude, can each measure one of several hundred events, of which we examined 177.

Power measurement was performed using a commercial AC power meter with an accuracy of 0.9% sampled at 2.5 Hz. This was inserted between the wall socket and the machine's power plug and thus measured the system's total AC power consumption (a reduction of the total AC power consumption being the main goal for server power management).

5.1.2 Approach

We used benchmarks from SPEC CPU2000 for characterisation and CPU2006 for evaluation. The input data sizes were varied based on platform performance and time constraints.

All experiments were conducted on Linux 2.6.24.4, compiled with GCC version 4.1. Benchmarks were run in single-user mode and several sources of interrupts and wakeups within the kernel were removed. On the Latitude, the timer-interrupt frequency was reduced to 20 Hz during characterisation in order to further minimise jitter caused by interrupt handling on the overall benchmark results during the parameter selection and model generation.

All benchmarks were conducted in an air-conditioned environment to minimise error due to temperature variation.

As the calibration of our model requires many identical runs, differentiated only by the events measured by the performance counters, deterministic execution times are essential. We found that this is not easy to achieve in Linux — in some cases the execution times of supposedly identical runs differed by as much as 20%. We traced this to L2 cache miss rates changing by several orders of magnitude between runs, apparently due to conflict misses resulting from changed physical-memory layout. We worked around this problem by rebooting the system before each benchmark run.

To further ensure that both performance and energy results for each benchmark were consistent between runs, all benchmarks were run from a `tmpfs` filesystem (i.e. from RAM) to eliminate disk I/O. Output data was discarded via `/dev/null` to avoid console output and interrupts. Buffer cache writebacks were disabled and disk power management was disabled to ensure that the disk remained spinning (avoiding spin-up and spin-down energy and interrupts).

The characterisation can take a considerable amount of time depending on the platform specifications. For example, with 10 frequencies, 20 benchmarks, 150 performance events, 2 performance counters, and an average benchmark runtime of 100 seconds, the time for characterisation would take nearly 18 days. We reduced this somewhat by using incomplete benchmark sets (subset of benchmarks and settings) to close in on the most relevant performance events, which were then thoroughly evaluated using the full sets.

5.2 Characterisation

While the procedure for selecting performance-counter events, as described in Section 4.3, is systematic and fully determined by the benchmarks, we are left with one relatively arbitrary decision: We need to determine the best performance-counter events for two models, performance and power, but only have a small number of events that can be observed concurrently.

Given that performance and power consumption for a given setpoint are both essentially determined by a process's memory accesses, it makes sense to share the same events between both models. In theory it would be possible to perform a regression analysis for the combined model. However, we found that we get good results if we use regression on the performance model to select the events.

On the server platform, our characterisation procedure selected the following set of events for the four available performance counters, all of which are intuitively related to memory-boundedness:

- quadword write transfers
- L2 cache misses (datacache fill)
- dispatch stalls due to reorder buffers being full
- DRAM accesses due to page conflicts.

The performance model resulting from these events plus the cycle counter fits the characterisation data with a coefficient of determination of $R^2 = 0.98$, which indicates an excellent fit. The power model, based on the same performance counters and the basic fV^2 term, also results in a coefficient of determination of $R^2 = 0.98$.

For the laptop, the selection procedure chose the following counters:

- Number of completed burst transactions
- Number of lines removed from the L2 cache

This resulted in a performance model with $R^2 = 0.98$ and a power model with $R^2 = 0.96$, which, given the limited number of performance counters, can also be considered a good fit.

The normal Linux sleep-state policies (implemented by `cpuidle`) operate when idle time is available. In the case where frequency scaling affects the idle time, the associated energy used can be added in to the model.

The power in each of the idle modes on the laptop was characterised according to Section 4.4. The standard Linux mechanism for dealing with idle modes (`cpuidle` with the `menu` governor) was enabled, and the percentage of time spent in each mode while idle was varied using a synthetic benchmark to repeatedly sleep for different periods, causing `cpuidle` to use different sleep states. The results fit Equation 4 with $R^2 > 0.99999$.

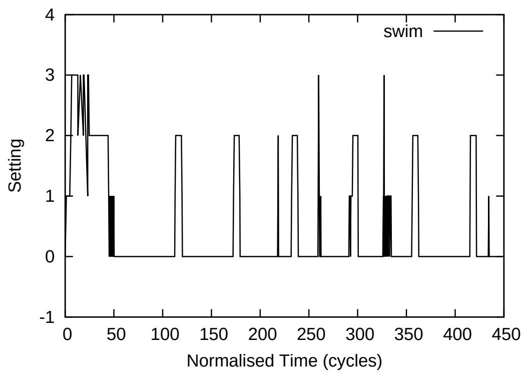
We also measured the average idle power for both the laptop and server. When idle for long periods, the laptop resides almost exclusively in C4. We therefore use the power for C4 as the idle-power when the laptop is under-utilised. The task of predicting the percentage of time spent in each of the idle states has been left to future work in workload prediction.

5.3 Adaptation to workload

At each time slice, the Koala implementation selects a frequency setting based on the system's energy-management policy as described in Section 4.5. The policy can be selected at run time via a `/proc` interface.

Figure 8 shows how Koala adapts to the memory- and CPU-bound phases of the `swim` benchmark (using a minimum-energy policy). The lower graph ofFigure 8 shows how the addition of the frequency switch latency terms greatly reduces the number of frequency switches, by introducing a bias against switching. This improved accounting saves an additional 1% of energy in this case.

Three time-slice lengths were trialled, the commonly-used 100 Hz and 250 Hz, as well as 1000 Hz (sometimes used in real-time applications). The system worked well at both 100 Hz and 250 Hz. At 1000 Hz the accuracy decreased markedly. We attribute this to two effects: Firstly, the frequency switch overhead is becoming more significant compared to the time slice. Secondly, averaging over a shorter period makes the result more sensitive to short-term fluctuations in application behaviour. We did not investigate this issue further, as the (more standard) longer time slices worked well.



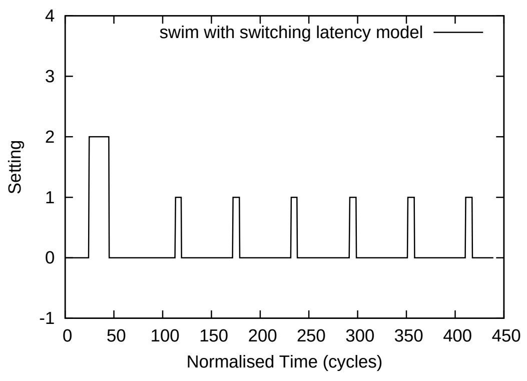
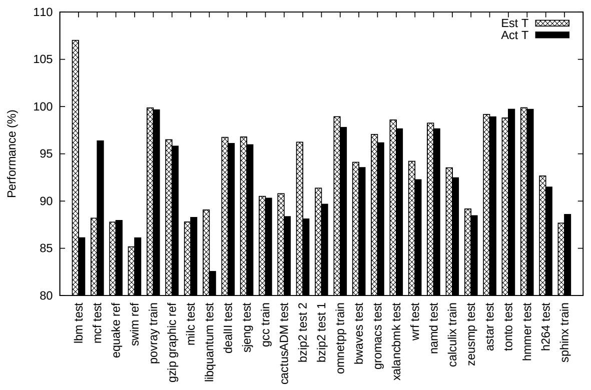


Figure 8. Koala behaviour for the first 1000 time slices of `swim` on the server with and without latency terms.

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.



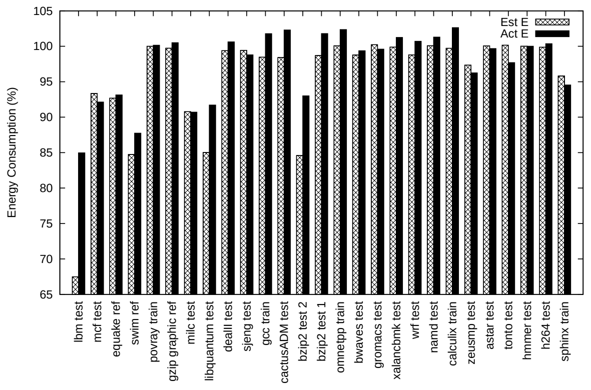


Figure 9. Comparison of estimated vs. actual performance (top) and energy (bottom) for the minimum energy policy on the server platform.

There are two possible explanations for this behaviour. For one, the characterisation benchmark set (CPU2000) may not cover a wide-enough range of behaviours, as such, failing to produce a parameter/weight selection that allows prediction of LBM's behaviour. The second is that the available set of performance-counter events may just not be suitable for accurate prediction of the behaviour of this benchmark.

We enabled idle energy in the model (which adjusts the energy for any extra idle time created thanks to frequency increases), and ran benchmarks over a fixed time period. In this case, on both platforms, the energy-optimal frequency is almost always the minimum.

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

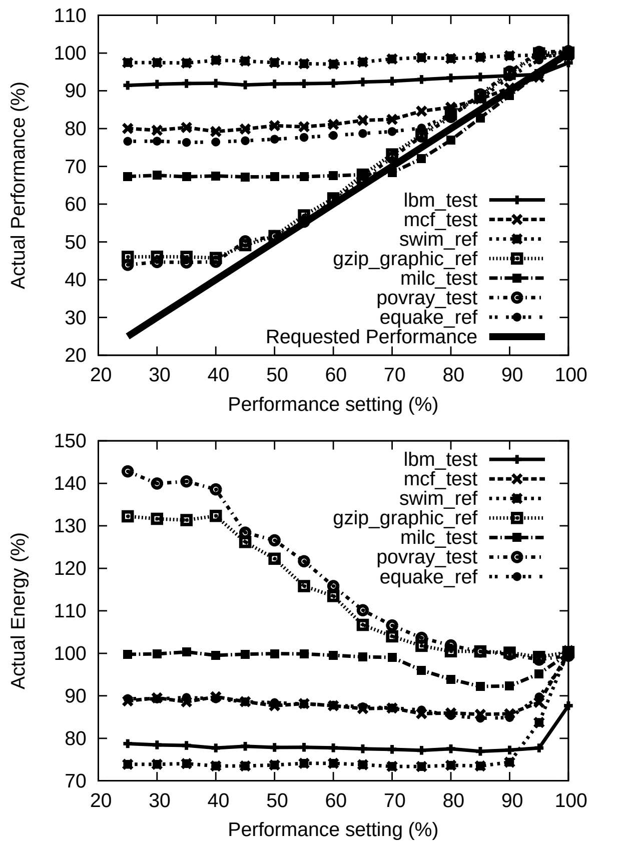


Figure 10. Maximum-degradation policy on the Latitude

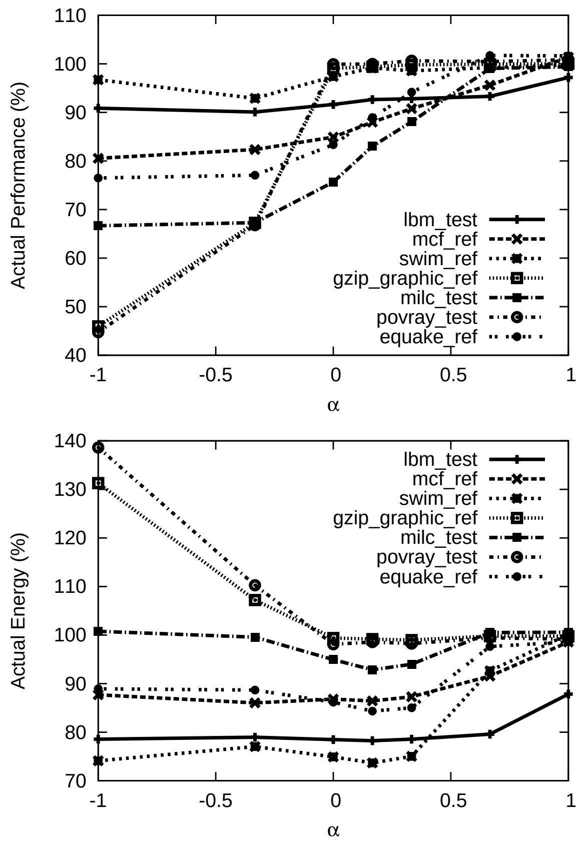


Figure 11. Generalised energy-delay policy on the Latitude.

target performance, this results from the discrete setpoints, inaccurate performance estimation, and Koala's adjustment 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 α 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 α, since there is no energy to save. For the same α 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-

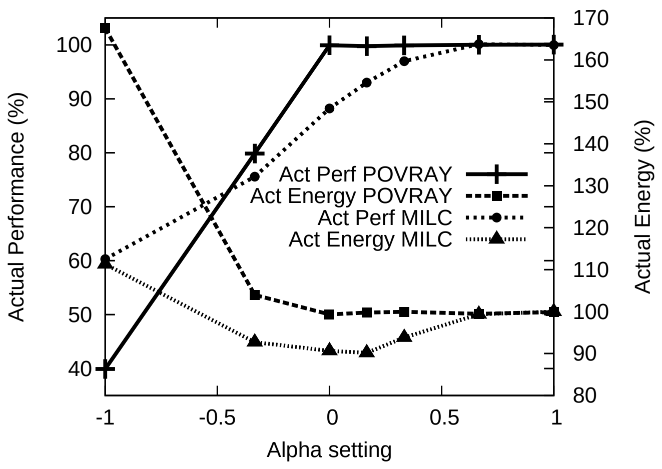


Figure 12. Generalised energy-delay policy on the server.

duce 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 (α), 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 α to 1, and the system runs at maximum performance. As the battery is depleted, the daemon lowers α until the battery gets below

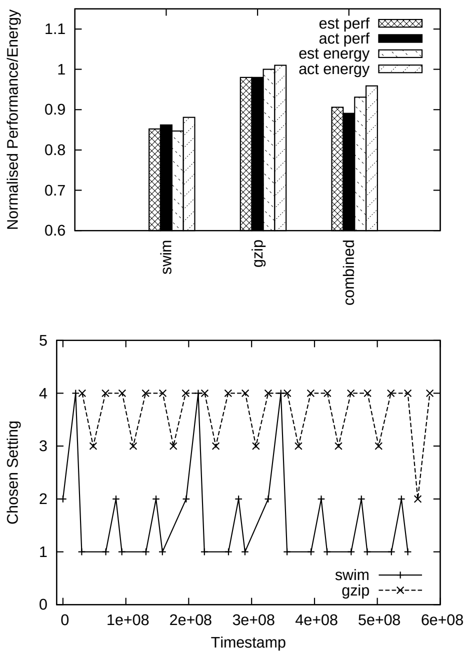


Figure 13. Koala multi-tasking on the server

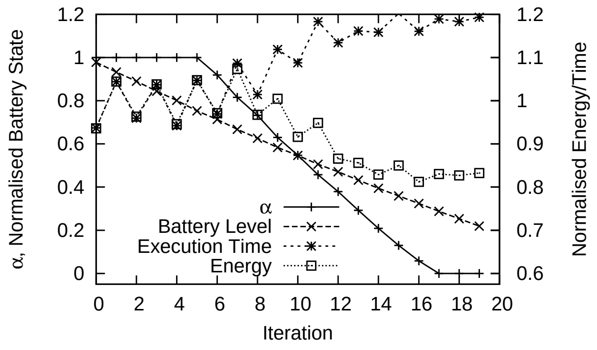


Figure 14. Using the Latitude's battery state of charge to drive the power management policy.

30% and then α is set to 0, i.e. minimum energy. Figure 14 shows how the performance-energy tradeoff changes as the battery depletes while running `mcf`

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, α is lowered towards 0 (the minimum energy setting), and in times of high load, α 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 overhead, all calculations were performed in fixed point, and pre-calculated lookup-tables used where appropriate.

Reducing the number of setpoints considered in the calculations also reduces the overhead. Therefore, setpoints which are never chosen should be excluded.

We ran the set of benchmarks on kernels with and without Koala enabled. On both the laptop and the server, for both a single task executing, and for two concurrent tasks, the mean performance difference was well within the standard deviation of the benchmarks. To emphasise the overheads, the timer tick frequency was increased up to 1000Hz, but the mean difference in performance between the two kernels was still unmeasureable.

6. Main-stream practicality

How Koala would be implemented in main-stream operating systems running on varying hardware remains an open problem. The issues are varied, but considered manageable, particularly if hardware vendors were to become involved.

One problem is the effort required in determining the models for a given system. This effort would be significantly reduced with hardware-vendor support in providing appropriate models and inputs, theoretically derived from the design. These would be both more accurate than the derived models as above, and easier to characterise. In-built power measurement hardware would allow for characterisation either as the system runs, adapting the models online, or automatically at boot or burn-in. Components without manufacturer-provided models could be modelled as described above.

Koala stores the models as Linux modules generated from parameters in the Linux source tree. While this is practical on a small scale, it is impractical for vendors. An alternative is to store models (or components of the model) in ROM or using ACPI tables/methods.

7. Conclusions and Future Work

Effective power management by the OS is of increasing importance for a wide class of computing platforms, from small mobile devices to high-end servers. Present approaches are ad-hoc mechanisms and policies which, as we have shown, often leads to poor results.

We have presented the Koala approach, which has two components. Firstly, we provide the OS with the right tools to perform effective power management, by allowing it to gain insight into the relevant properties of an application. We have presented a model that allows the OS in most cases to obtain accurate estimates of a running process's power consumption, as well as the performance and power response to frequency scaling. We have also demonstrated that benchmarks have sufficient temporal locality to use this information to predict their behaviour under changed frequency settings.

The second component is a policy that allows the OS to tune the system's operation towards an energy-management objective. The generalised energy-delay policy contains a single parameter which the OS can use to run the system at maximum performance, minimum energy, reduced thermal load, or intermediate values representing trade-offs between performance and energy. The OS can tune this parameter to adapt to a changing energy-management objective.

Rather than treating every process the same, Koala adjusts individual processes differently in order to achieve the best overall result. Specifically, when in an intermediate energy regime, memory-bound processes where small reductions in performance result in large energy savings are throttled more than CPU-bound processes where small energy savings come at the cost of a significant performance penalty.

The main shortcoming we found was a single benchmark whose behaviour was poorly predicted by the system. While energy was still saved for this benchmark, the significant errors in the prediction of its performance and energy use certainly lead to sub-optimal performance. We have not (yet) determined whether this failure is a result of an insufficient characterisation set, and could be fixed by more a orthogonal characterisation workload, or whether it due to the hardware-provided performance events not being sufficient to predict the behaviour of that particular workload. In any case, the experience shows that operating systems would benefit from hardware manufacturers providing statistics tailored to predicting energy use.

In the future, we intend to improve both the modelling accuracy, and the workload prediction capabilities within Koala. While our present techniques are sufficient for effective energy management, improvements will allow even more energy to be saved, and a more accurate trade-off with performance. We intend to generalise to more platforms, taking into account multi-core systems, I/O power, and the other power management challenges that an even more varied set of platforms will present.

Acknowledgments

Part of the work for this paper was carried out while David Snowdon was an intern at Intel Corporation. He gratefully acknowledges the support and feedback given by members of Intel's Corporate Technology Group.

NICTA is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

References

[AbouGhazaleh 2008] Nevine AbouGhazaleh, Bruce R. Childers, Daniel Mosse, and Rami G. Melhem. Integrated CPU cache power management in multiple clock domain processors. In *3rd HIPEAC*, Göteborg, Sweden, Jan 27-29 2008.

[Barroso 2007] Luiz Andre Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE Comp.*, 40(12):33–37, Dec 2007.

[Bellosa 2000] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *9th SIGOPS Eur. WS*, Kolding, Denmark, Sep 17–20 2000.

[Bircher 2005] Lloyd Bircher, Madhavi Valluri, Jason Law, and Lizy John. Runtime identification of microprocessor energy saving opportunities. In *Int. Symp. Low Power Electron. & Design*, 2005.

[Bircher 2007] W. Lloyd Bircher and Lizy K. John. Complete system power estimation: A trickle-down approach based on performance events. In *Int. Symp. Performance Analysis Syst. & Softw.*, pages 158–168. IEEE Computer Society, 2007.

[Choi 2005] Kihwan Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *Trans. CAD ICAS*, 24(1):18–28, Jan 2005.

[Del 2004] *Dell Latitude D600 Systems User's Guide*. Dell Inc., Nov 2004. URL `http://support.ap.dell.com/support/edocs/systems/latd600/`.

[Economou 2006] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system power analysis and modeling for server environments. In *2nd WS Modeling, Benchmarking & Simul.*, pages 158–168, Boston, MA, June 2006.

[Grunwald 2000] Dirk Grunwald, Philip Levis, Keith I. Farkas, Charles B. Morrey III, and Michael Neufeld. Policies for dynamic clock scheduling. In *4th OSDI*, pages 73–86, San Diego, CA, USA, Oct 2000.

[Heath 2005] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *10th PPOPP*, pages 186–195, 2005.

[Hsu 2003] Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. *SIGPLAN Not.*, 38(5):38–48, 2003.

[Hsu 2004] Chung-Hsing Hsu and Wu chun Feng. Effective dynamic voltage scaling through CPU-boundedness detection. In *2004 WS Power-Aware Comp. Syst.*, pages 135–149, 2004.

[Isci 2006] Canturk Isci, Gilberto Contreras, and Margaret Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *39th MICRO*, Nov 2006.

[Kephart 2007] J.O. Kephart, H. Chan, R. Das, D.W. Levine, G. Tesauro, F. Rawson, and C. lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *ICAC*, 2007.

[Kotla 2004] Ramakrishna Kotla, Anirudh Devgan, Soraya Ghiasi, Tom Keller, and Freeman Rawson. Characterizing the impact of different memory-intensity levels. *2004 Int. WS Workload Charact.*, pages 3–10, 2004.

[Kumar 2006] Amit Kumar, Li Shang, Li-Shiuan Peh, and Niraj K. Jha. HybDTM: a coordinated hardware-software approach for dynamic thermal management. In *43rd DATE*, Jul 2006.

[Mahesri 2004] Aqeel Mahesri and Vibhore Vardhan. Power consumption breakdown on a modern laptop. In Babak Falsafi and T. N. Vijaykumar, editors, *2004 WS Power-Aware Comp. Syst.*, volume 3471 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2004.

[Martin 2001] Thomas L. Martin. *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Melon University, 2001.

[Miyoshi 2002] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *16th Int. Conf. Supercomp.*, pages 35–44, New York, NY, USA, 2002. ACM Press.

[NVIDIA Corporation 2007] NVIDIA Corporation. PowerMiser 7.0: Intelligent power management technology for NVIDIA GeForce 8M series and Quadro NVS/FX notebook GPUs, May 2007. URL `http://www.nvidia.com/object/feature_powermizer.html`.

[Peddersen 2007] Jorgen Peddersen and Sri Parameswaran. CLIPPER: Counter-based low impact processor power estimation at run time. In *12th ASPDAC*, Yokohama, Japan, Jan 2007.

[Pénzes 2002] Paul I Pénzes and Alain J. Martin. Energy-delay efficiency of VLSI computations. In *12th ACM Great Lakes Symp. VLSI*, pages 104–111, New York, NY, USA, 2002. ACM. ISBN 1-58113-462-2.

[Pillai 2001] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th SOSP*, pages 89–102, Lake Louise, Alta, Canada, Oct 2001.

[Scaramella 2006] Jed Scaramella. Worldwide server power and cooling expense: 2006-2010 forecast. White paper 203598, IDC, Sep 2006. `http://www.sun.com/service/eco/IDCWorldwideServerPowerConsumption.pdf`.

[Snowdon 2005] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. Power management and dynamic voltage scaling: Myths and facts. In *2005 WS Power Aware Real-time Comput.*, New Jersey, USA, Sep 2005.

[Snowdon 2007] David C. Snowdon, Stefan M. Petters, and Gernot Heiser. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *7th Int. Conf. Emb. Softw.*, Salzburg, Austria, Oct 2007.

[Stoess 2007] Jan Stoess, Christian Lang, and Frank Bellosa. Energy management for hypervisor-based virtual machines. In *2007 USENIX*, Santa Clara, CA, Jun 2007.

[Weiser 1994] Mark Weiser, Brent Welch, Alan J. Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *1st OSDI*, pages 13–23, 1994.

[Weissel 2002] Andreas Weissel and Frank Bellosa. Process cruise control—event-driven clock scaling for dynamic power management. In *CASES*, Grenoble, France, Oct 8–11 2002.

[Zeng 2005] Hang Zeng and Carla S. Ellis Alivn R. Lebeck. Experiences in Managing Energy with ECOSystem. *Pervasive Comput.*, 4(1):62–68, 2005.