Mobile Multicores: Use Them or Waste Them

Aaron Carroll and Gernot Heiser
NICTA and University of New South Wales
{Aaron.Carroll,gernot}@nicta.com.au

Abstract

Energy management is a primary consideration in the design of modern smartphones, made more interesting by the recent proliferation of multi-core processors in this space. We investigate how core offlining and DVFS can be used together on these systems to reduce energy consumption. We show that core offlining leads to very modest savings in the best circumstances, with a heavy penalty in others, and show the cause of this to be low per-core idle power. We develop a policy in Linux that exploits this fact, and show that it improves up to 25% on existing implementations.

1 Introduction

Energy efficiency is a first-class concern in mobile embedded systems, such as the smartphone, due to battery constraints. At the same time, multi-core processors are emerging in the embedded space, with high-end smartphones now shipping with quad-core application processors. Such systems present new challenges and opportunities for energy management.

Modern processors provide mechanisms to control power consumption. *Offlining* allows the operating system to power-down individual cores, allowing the remaining cores to continue processing. DVFS, *dynamic voltage and frequency scaling*, provides for the reduction of CPU operating frequency (at the cost of performance), thus reducing dynamic power per the equation $P \propto fV^2$.

In this paper we investigate the combined efficacy of these two mechanisms on a smartphone applications processor. In particular we focus on slack management, i.e., managing an under-utilised processor to reduce energy without adversely affecting performance.

We start in Section 2 by measuring the energy consumption of a range of synthetic workloads while varying the core frequency and number of active cores. We analyse this data in Section 3, and conclude that offlining cores is generally ineffective as a power management policy.

We propose high-level principles for designing a unified offline/DVFS policy, and in Section 4 describe the implementation of such a policy, *medusa*, in the Linux kernel. In Section 5 we perform a series of benchmarks to evaluate its effectiveness, comparing it against existing policies.

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version will be published in Proceedings of the 5th Workshop on Power-Aware Computing and Systems (HotPower'13).

2 Motivation

To motivate our design, we first measure the energy consumption of a series of workloads, keeping the total work constant but varying the CPU frequency and number of online cores; this pair forms the *operating point*, or OP. *loadcpu* is designed to emulate periodic real-time workloads. It consists of 4 processes executing a tight busy-loop for a certain number of iterations, and then sleeping for the remainder of the 50 ms period. The number of iterations is set to achieve the desired total CPU load: we used 10, 25, 50 and 75% of maximum capacity (i.e. 4 cores running at maximum frequency). The total execution time and amount of work performed are both constant, with duty cycle varying with the OP. We run each workload only at OPs that can sustain the required throughput (i.e. utilisation $\leq$ 100% without over-running the 50 ms interval).

loadmem is similar to *loadcpu*, but rather than executing a busy-loop, it strides through memory performing read-modify-write on buffers equal in size to the L2 cache. We also ran a fully CPU-bound workload (*spin*), which executes a fixed number of iterations of a busy-loop across 4 processes, and ran it across the full range of OPs. Finally, we use a software video decoder playing an H.264-encoded video across OPs able to decode all frames with no overrun. In all cases, the average of 3 iterations is reported, with worst-case relative standard deviation of 6%.

We run these workloads on 2 embedded platforms: the Samsung Galaxy S III (SGS3), a latest-generation off-

Characteristic	SGS3	MDP
SoC	Exynos 4412	APQ8064
CPU	Cortex-A9	Krait
cores	4	4
Frequency (MHz)		
min	200	384
max	1400	1512
Cache (KiB)		
L0 (I/D)	—	4 / 4
L1 (I/D)	32 / 32	16 / 16
L2 (shared)	1024	2048
OS	Android 4.0.4	

Table 1: Platform characteristics.

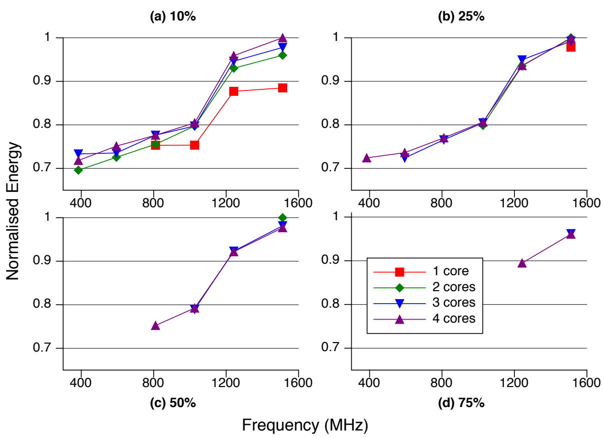

Figure 1: MDP loadcpu energy at 10, 25, 50 and 75% load.

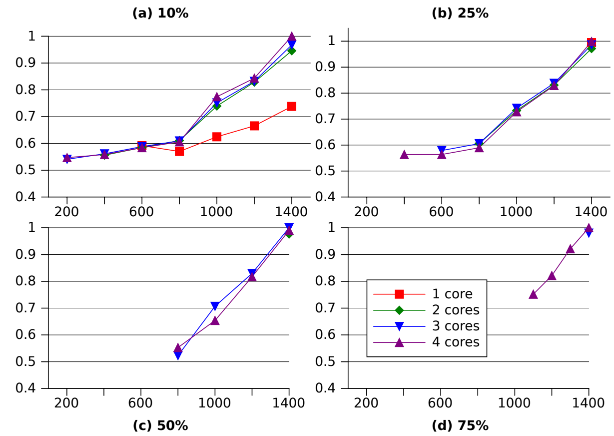

Figure 2: SGS3 loadcpu normalised energy vs. frequency (MHz) at 10, 25, 50 and 75% load.

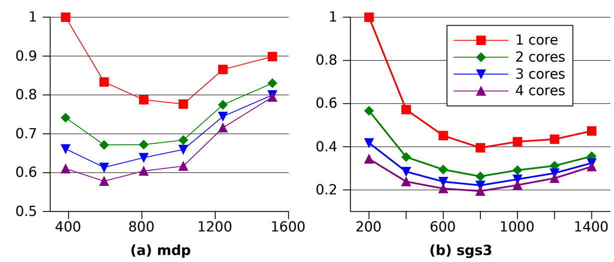

Figure 3: spin normalised energy vs. frequency (MHz).

the-shelf smartphone; and the Qualcomm Snapdragon S4 Pro MDP/T tablet development kit (MDP). These platforms represent two SoC vendors (Samsung and Qualcomm, respectively) and two different implementations of the ARMv7 architecture (ARM Cortex-A9 and Qualcomm Krait), and hence give us reasonable coverage of the high-end mobile CPU space. The characteristics of these platforms are summarized in Table 1. While the MDP platform allows independent frequency settings for each core, for the present work we force all cores to run at the same frequency.

The results of *loadcpu* on both platforms are shown in Figures 1 and 2. The behaviour is similar on both platforms. Energy consumption is very weakly dependent on the number of cores used, except at low load (10%), where it is significantly more efficient to use a single core at mid to high frequency. For any number of cores, energy is an increasing function of frequency (Fig. 2(a) 1 core @ 800 MHz being a single exception). Minimum energy is observed at minimum frequency, and the reverse is also true: maximum energy at maximum frequency. We omit the results of *loadmem* but note virtually identical behaviour on MDP, with SGS3 showing a stronger positive correlation between number of cores and energy. Our observations otherwise continue to hold.

The results of *spin* on MDP and SGS3 are shown in Figure 3, and again show common features across platforms. At any frequency, energy *decreases with* increasing number of cores; hence, energy is minimised with 4 online cores, and maximised with 1 core (at minimum frequency). The optimal frequency varies with the number of cores, but note that with additional cores, the optimal frequency never increases, but sometimes decreases.

Figure 4 shows the video decode results on SGS3, displaying similar behaviour to *loadcpu*, as does the same benchmark running on MDP (not shown).

We also performed similar measurements on the Pandaboard, a development platform based around the OMAP4 SoC, featuring a dual-core Cortex-A9 CPU. However, this platform is less interesting since fewer operating points are supported (2 cores and 4 frequencies). For brevity we omit the results, but note that they are similar to the above, and our observations hold.

3 Analysis

Clearly, the energy cost of choosing an incorrect operating point is substantial. Indeed, this result is well known from the single-core DVFS literature [SLSPH09]. Our results

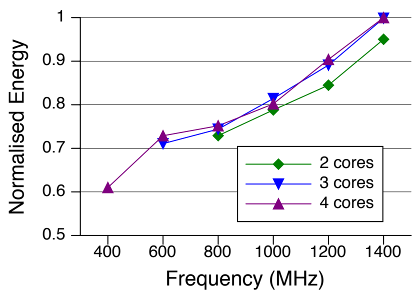

Figure 4: SGS3 video playback energy.

show that the multi-core processor only exacerbates this problem, both by increasing the penalty of incorrect OP selection, and by increasing the size of the optimisation problem with the additional “number of online cores” dimension. Moreover, the results show that the offline and DVFS mechanisms are inherently tied: one cannot be optimised independently of the other. Applying the naive wisdom that lower power implies lower energy, i.e. that fewer cores result in lower energy consumption, leads to catastrophic results.

In the periodic case, reducing frequency tends to reduce energy consumption. Furthermore, and maybe surprisingly, for a particular frequency there is little variation of energy consumption with the number of online cores. However, the increased computation power of a larger number of cores makes it possible to run the workload at a lower frequency, and thus reduce energy consumption—the opposite of the naive expectation!

This is a consequence of the low idle power of each core, as we can show with a simple model. Consider an n-core CPU at fixed frequency:

$$P_{\text{CPU}} = P_{\text{uncore}} + n(P_{\text{active}} + P_{\text{idle}}) \;, \tag{1}$$

where P_{idle} is the power consumed for an online but idle core (i.e. in a shallow sleep state), and P_{active} is the additional power when computing. P_{uncore} is the power consumed by the remainder of the CPU (buses, last-level cache, etc.) Substituting T for the period of the workload, and t for the per-period execution time (where $t < T$), we get per-period energy of

$$E_{\text{CPU}} = P_{\text{uncore}}T + n(P_{\text{active}}t + P_{\text{idle}}T) \;. \tag{2}$$

For a workload with good scalability (i.e. $t = \alpha/n$), we get

$$E_{\text{CPU}} = P_{\text{uncore}}T + \alpha P_{\text{active}} + nP_{\text{idle}}T \;, \tag{3}$$

and hence

$$E_{\text{CPU}} = (P_{\text{uncore}} + nP_{\text{idle}})T + k \;. \tag{4}$$

This shows that the CPU energy consumption at a fixed frequency is independent of n if the idle power of an online core is low.

For CPU-bound workloads it is always more energy-efficient to run with more cores online, because increasing throughput reduces execution time and thus reduces the accumulation of static CPU energy. This is an example of the race-to-idle policy, which is well-documented in the DVFS literature [MLH+02]. However, with DVFS, dynamic power is super-linear in frequency and hence race-to-idle is not necessarily optimal. On the other hand, core power is linear in the number of online cores, and thus additional cores are always more efficient. Again this can be demonstrated with a simple model,

$$P_{\text{CPU}} = P_{\text{uncore}} + nP_{\text{core}} \;. \tag{5}$$

Assuming scalability ($t \propto 1/n$) where t is the execution time, we get

$$E_{\text{CPU}} \propto \frac{P_{\text{uncore}}}{n} + P_{\text{core}} \;. \tag{6}$$

The scalability requirement can be relaxed by replacing the assumption of workload-independent dynamic power by the approximation that P_{dynamic} is proportional to instructions per cycle [SKK11], from which it follows that E_{dynamic} is proportional to the number of executed instructions, which is constant for a fixed workload. Expanding P_{active} into its dynamic and static components, from Equation 1 we get

$$E_{\text{CPU}} = (P_{\text{uncore}} + n(P_{\text{static}} + P_{\text{idle}}))t + nE_{\text{dynamic}} \;. \tag{7}$$

If we execute i instructions spread across n cores, then from the above assumption,

$$E_{\text{dynamic}} \propto i/n \;, \tag{8}$$

hence

$$E_{\text{CPU}} = (P_{\text{uncore}} + n(P_{\text{static}} + P_{\text{idle}}))t + k \;. \tag{9}$$

Thus, if $P_{\text{static}} + P_{\text{idle}}$ is small compared with P_{uncore}, then increasing n has negligible impact on E_{CPU}, even for workloads with sub-linear scalability. Note that for the purposes of this analysis we can treat P_{uncore} as constant—adding the dynamic uncore contribution would only strengthen our argument.

Intuitively it is reasonable that per-core idle power is low on smartphone-class embedded processors. They typically have little on-core state, and a modern ARM core such as the Cortex-A9 heavily clock-gates, even in the lightest sleep state, which “reduces the power drawn to the static leakage current, leaving a tiny clock power overhead requirement to enable the device to wake up” [ARM10]. In Figure 5 we show this directly with a plot of idle power consumption as a function of cores online, at both maximum and minimum frequency, on the SGS3. The equations of linear fit show that P_{idle} is approximately 6 and 25 mW per core at minimum and maximum frequency, respectively. The corresponding uncore power is 263 mW and 415 mW. Hence, $P_{\text{uncore}}/P_{\text{idle}}$ is > 16 at f_{max} and > 40

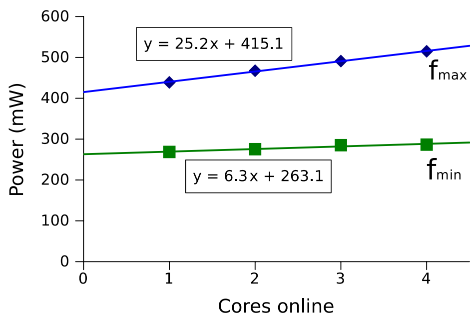

Figure 5: SGS3 idle power consumption at minimum and maximum core frequencies.

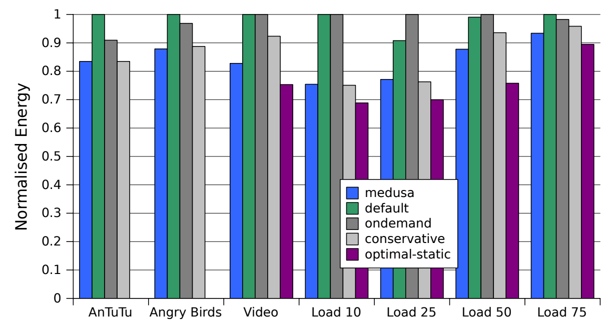

Figure 6: Normalised energy consumption of benchmarks under each energy-management policy.

at $f_{\min}$. Further, we can determine P_{static} for one core by solving

$$\begin{aligned} P_{\text{CPU}} &= P_{\text{uncore}} + P_{\text{idle}} + P_{\text{dynamic}} + P_{\text{static}} && (10)\\ &= P_{\text{uncore}} + P_{\text{idle}} + \beta f V^2 + P_{\text{static}}\ , && (11)\end{aligned}$$

using the values of P_{uncore} and P_{idle} from Figure 5, and P_{CPU} from our *spin* results (322 mW at $f_{\min}$ and 1092 mW at $f_{\max}$). This yields a P_{static} of approximately 10 mW per core. Thus we expect to see, as we have indeed shown, that having more online cores reduces energy consumed.

From this analysis, we propose the following principles for design of an energy management policy: *scale out (online cores) before scaling up (increasing frequency); offline cores conservatively; and reduce frequency aggressively.*

4 Medusa: an offline-aware governor

Based on the principles outlined in Section 3, we have implemented such an energy management policy, *medusa*, in the Linux kernel running on the MDP platform. It aims to reduce energy consumption without affecting performance by manging slack time in the CPU. Medusa functions as a "cpufreq" DVFS governor, and additionally controls the number of online cores.

At a high level, the policy runs regularly (every 200 ms by default) and selects a new operating point by checking the following conditions, executing the first that applies.

1. If the number of runnable threads exceeds online cores, online additional cores (if available).
2. If any online core is at maximum utilisation, increase frequency to the next highest.
3. If all cores are under-utilised, set the frequency to that which maximises utilisation.
4. If any core is below 5% utilisation, offline it.

To improve OP stability, we apply a small amount of hysteresis, and average load across 3 update cycles for the purposes of reducing frequency and offlining cores. Furthermore, to improve responsiveness, frequency is increased more aggressively after several consecutive increases, or when executing a new workload from a quiescent state.

The full implementation, including extensive logging, configuration, and debug support, is about 1000 lines of code. The only CPU-specific information used is the available frequencies and number of available cores.

5 Evaluation

To evaluate medusa, we ran a number of workloads and compared the energy consumption to that of 3 existing policies. *Default* is the built-in policy shipped with the MDP device, which uses the ondemand DVFS governor combined with a (closed-source and undocumented) userspace tool called *mpdecision*. *Ondemand* and *conservative* are standard Linux DVFS governors, but since they control only frequency, we repeated each experiment statically setting the number of active cores between 1 and 4, and report the result yielding minimum energy. Finally, where feasible we report the *optimal-static* OP; that is, the minimum energy consumption across all OPs when fixed for the full run. For ondemand, conservative and optimal-static, we discard any data where our OP constraint causes the benchmark to exceed nominal runtime.

In total we used 7 benchmarks of 4 types. *AuTuTu* is a 3D-intensive graphics benchmark. *Angry Birds* is a 2D game, and the scenario involves a short (2 minute) play through the game, using an input trace/replay methodology for repeatability. *Video* is playback of an H.264-encoded video using a software decoder. Finally, *loadgen* is the micro-benchmark from Section 2, again used at the same 4 load levels.

Figure 6 shows the normalised full-system energy consumption for each policy and benchmark, averaged over 3 iterations. Across all benchmarks, medusa performs equally or better than the dynamic policies, but is never more efficient than optimal-static. This is due to the elasticity of the workloads; that is, temporary overload is acceptable since the extra work can be completed later, when load is lower, without causing overrun. Since the dynamic policies (including medusa) are work-conserving, they can not exploit such properties. Moreover, it is not clear how such a policy could be implemented without affecting performance for some applications.

On average, medusa consumes 85% the energy of the default and ondemand policies, 75% in the best case. Compared with conservative, medusa uses 97% on average, and 90% best case. However, note that for ondemand and conservative we have selected the optimal number of cores manually, whereas medusa does so automatically, and so can adapt as the workload changes. Clearly this is important, since the optimum varies significantly with those policies: 4 cores 50% of the time, 3 cores 29%, 2 cores 14%, and 1 core in only 7% of cases.

6 Related work

Ghasemazar et al. [GPP10] takes a theoretical approach to the problem of combining offlining and DVFS. They develop a control-theoretic feedback algorithm to select an operating point, and evaluate it via simulation of an Alpha-like processor. Compared with a baseline of all online cores and open-loop DVFS, they show a 17% improvement in energy consumption. They demonstrate that increasing the number of online cores always decreases the optimum frequency. As noted earlier, we see a similar though small effect. They claim that offlining unused cores is important, our results in most cases do not reflect this.

Li and Martinez [LM06] develop a number of heuristics to reduce the optimisation search space and algorithms to search for the optimal operating point. The policy is reactive feedback-based, and hence depends on online power measurement. They evaluate these on a simulated Alpha, and show performance close to optimal for a range of workloads. They claim that the optimal operating point depends heavily on the power-performance curve of the particular processor. We note that within the particular class of CPUs in our study, variation in behaviour is is somewhat limited.

Gupta et al. [GBK+12] consider the cost of the uncore component of power consumption on heterogeneous multi-core processors. They show that on desktop-class systems, uncore consumes a significant fraction of CPU energy, varying 20–80% of total depending on workload, and that core and uncore power are approximately equal at idle. Our data shows that uncore contributes a much larger proportion of idle power in embedded-class processors, demonstrating why our results differ from those in the desktop/server space.

7 Conclusions and Future Work

We have shown that, due to the low per-core idle power consumption of embedded applications processors, offlining of cores makes little sense for energy management if work is available to run on them. This occurs for 2 reasons: one, onlining cores allows access to lower, more energy-efficient frequencies for equivalent throughput; and two, completing the work quicker can reduce accumulation of static CPU “uncore” power, which is significant in such processors. A corollary of this for developers is that writing multi-threaded applications can be effective for reducing energy, even if the additional CPU throughput is not required.

Using these observations, we have implemented *medusa*, a Linux CPU frequency governor, that significantly improves energy efficiency compared with several existing implementations: up to 25% reduction with no cases of increased energy.

In the present work, we have concentrated on management of idle time to reduce energy consumption without affecting performance. In future, we hope to extend this work by incorporating active energy management using multi-core-aware DVFS optimisation. The most promising path seems to be to extend the approach taken by Koala [SLSPH09]: use a parameterised hardware model, characterised offline, which observes the application behaviour and uses performance counters to predict on-line the system's performance and energy response to changes in operating points. Such a system would allow the trading of performance for reduced energy consumption.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[ARM10] ARM Ltd. *Cortex-A9 Technical Reference Manual*, r2p2 edition, 2010. ARM DDI 0388F.

[GBK+12] Vishal Gupta, Paul Brett, David Koufaty, Dheeraj Reddy, Scott Hahn, and Karsten Schwan. The forgotten ‘uncore’: On the energy-efficiency of heterogeneous cores. In *2012 USENIX ATC*, Boston, MA, USA, Jun 2012.

[GPP10] M. Ghasemazar, E. Pakbaznia, and M. Pedram. Minimizing energy consumption of a chip multiprocessor through simultaneous core consolidation and DVFS. In *ISCAS*, pages 49–52. IEEE, 2010.

[LM06] J. Li and J.F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *12th HPCA*, pages 77–87. IEEE, 2006.

[MLH+02] 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, Jun 2002. ACM Press.

[SKK11] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive DVFS. In *IGCC*, pages 1–8. IEEE, Jul 2011.

[SLSPH09] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for OS-level power management. In *4th EuroSys Conf.*, Nuremberg, Germany, Apr 2009.