Unifying DVFS and Offlining in Mobile Multicores

Aaron Carroll and Gernot Heiser
NICTA and UNSW, Sydney, Australia
{Aaron.Carroll,gernot}@nicta.com.au

***Abstract*—Energy efficiency is a primary design criterion of the modern smartphone due to limitations in battery capacity. Multi-core processors are now commonplace in these devices, which adds a new dimension, the number cores used, to energy management. In this paper we investigate how the mechanisms of frequency scaling and core offlining interact, and how to use them to reduce energy consumption. We find surprising differences in the characteristics of latest-generation smartphones, specifically in the importance of static power. This implies that policies that work well on one processor can lead to poor results on another.**

We propose a simple policy that integrates core offlining with frequency scaling and implement it in a Linux-based frequency governor called medusa. We show that, despite its simplicity, medusa obtains energy savings that are as good or better than governors presently shipping on the studied phones and approaches the static optimal setting.

I. 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.

The modern multi-core applications processor provides several mechanisms to control energy consumption. One of these is *offlining*, which allows the operating system to switch off individual cores, reducing the per-core power consumption to zero, but allowing the remaining cores to continue processing. Another is the well-studied DVFS, or *dynamic voltage and frequency scaling*, which provides for the reduction of CPU operating frequency at the cost of performance. The use of these two mechanisms in tandem presents an interesting tradeoff: to respond to increasing system load, either the frequency of online cores can be increased or additional cores can be onlined (and vice versa for decreasing load). The energy-optimal decision depends, among other things, on the power/performance response of these two mechanisms.

Past research has observed the increasing importance of static power, and the resulting importance of sleep states in reducing energy consumption [2], [11]. When the per-core static power is significant, a larger number of active cores means more static power, and thus a higher energy consumption. In particular, deep sleep states mean that the *race-to-halt* approach may be beneficial, as it minimises the accumulation of static energy loss. Consequently, running at minimum frequency may not minimise energy use [14].

In this paper we revisit these assumptions and observations in the context of latest-generation smartphone applications processors. We extend our previous work [3] in investigating the combined efficacy of core idle power states and DVFS for maximising energy efficiency. We do not investigate trading off performance for energy, and instead attempt to minimise energy consumption without affecting performance (i.e. slack management). We find a surprising diversity in characteristics of latest-generation ARM processor implementations: from very significant to completely negligible static power. This has a significant impact on energy management, which does not seem to be well understood: phones ship with energy-management policies which produce highly non-optimal results.

We start our investigation in Section II by measuring the energy consumption of a range of synthetic workloads while varying the core frequency and number of active cores. In Section III we analyse these data from both an empirical and theoretical perspective, and show that running with more online cores generally reduces energy consumption and thus aggressive offlining of processors cores is an ineffective energy management strategy. In Section IV we describe an implementation of an energy management policy in the Linux kernel called *medusa*, which exploits these insights to reduce energy consumption. In Section V we perform a series of benchmarks to evaluate medusa's effectiveness, comparing it against existing policies and exploring how it can be adapted for particular platforms.

II. Motivation

To understand the problem and motivate a solution, we first measure the energy consumption of a series of simple microbenchmarks while varying the CPU frequency and number of online cores. This pair forms the *operating point* (OP).

A. Platforms

We run our benchmarks on two versions of the Samsung Galaxy S4 smartphone. The first, *S4-E*, is the GT-I9500 model which features the Samsung Exynos 5410 system-on-chip (SoC) with a quad-core Cortex-A15 and a quad-core Cortex-A7 CPU. These run in an ARM big.LITTLE "task migration" configuration [7] whereby one of the quad-core CPUs (called *clusters*) can be active at any time. Switching between the clusters is controlled by the operating system, and on our device this is done in the DVFS subsystem by "virtualising" the frequency control. When the OS requests a particular operating (virtual) frequency, this is used to decide which cluster to activate, and the virtual frequency is mapped to the physical frequency that the cluster will actually run at. Virtual frequencies in the range 250–600 MHz will cause the Cortex-A7 cluster to be active, running at twice the virtual frequency. For virtual frequencies at or above 800 MHz, the Cortex-A15 cluster will be active at a physical frequency equal to the virtual frequency. This is the default behaviour that ships with the GT-I9500 device, and for the purposes of our work we retain this behaviour and, unless otherwise noted, we use virtual frequencies in the remainder of the paper.

TABLE I. CHARACTERISTICS OF THE TWO GALAXY S4 PLATFORMS. †THE *S4-E* HAS 8 CPU CORES, BUT ONLY 4 ARE ACTIVE AT ANY TIME.

Characteristic	*S4-E*	*S4-S*
Model	GT-I9500	GT-I9505
SoC	Exynos 5410	Snapdragon 600 (8064T)
CPU	Cortex-A15/A7	Krait 300
cores	4†	4
Frequency (MHz)		
min	250	384
max	1600	1890
Voltage (V)		
min	0.9	0.9
max	1.15/1.2125	1.2125
Cache (KiB)		
L0 (I/D)	—	4 / 4
L1 (I/D)	32 / 32	16 / 16
L2 (shared)	1024	2048
OS	Android 4.2.2	

Our second platform is the GT-I9505 model featuring a Qualcomm Snapdragon 600 SoC with a quad-core Krait 300 CPU; we call this the *S4-S*. Table I summarises the relevant parameters for these two platforms. On *S4-E*, all cores run at the same voltage and frequency. The *S4-S* allows cores to run at independent voltages and frequencies, but for the purposes of this work we force all active cores to run at the same setting. Herbert and Marculescu [9] show that this is not a significant limitation.

These devices are latest-generation, off-the-shelf commodity smartphones. They represent two SoC vendors (Samsung and Qualcomm) and two independent implementations of the ARMv7 architecture (ARM Cortex and Qualcomm Krait), and hence give us good coverage of the high-end mobile CPU space.

B. Measurement

We measure power consumption of the devices by inserting a 20 mΩ current sense resistor between the battery and the device. We use a National Instruments NI-6229 [15] data acquisition system to sample the voltage drop across the sense resistor, and the battery supply voltage, from which power can be determined. To minimise interference from non-CPU components, we run the devices in airplane mode with all unrelated components disabled, including the screen. When the screen is required, as in the medusa evaluation benchmarks, we run it at minimum brightness.

C. Benchmarks

We use three workloads that emulate periodic tasks, designed so that the work performed and total execution time are fixed, but varying in duty cycle (i.e. the proportion of time spent computing vs. sleeping) by varying the OP (frequency and online cores). These benchmarks are *loadcpu*, *loadcache* and *loadmem*, which exercise the CPU, L2 cache, and main memory, respectively. For each, we use four levels of intensity, corresponding to 10, 25, 50 and 75% of the total system capacity (i.e. 100% means running all cores at maximum frequency, 50% means all cores at half frequency, or two cores at maximum frequency, etc.) Each benchmark consists of four identical processes executing the load for a certain number of iterations, and then sleeping for the remainder of the 50 ms period. We vary the number of iterations to control the load intensity, but for each benchmark the total execution time and amount of work performed are 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).

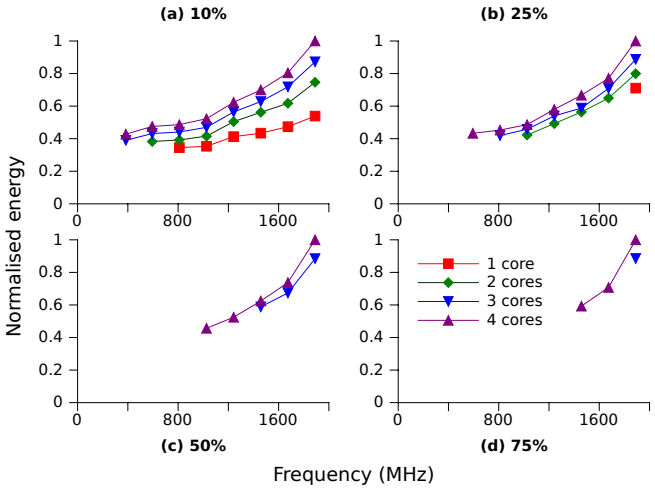

Fig. 1. *S4-S loadcpu* normalised energy vs. frequency (MHz) at 10, 25, 50 and 75% load.

In *loadcpu*, the workload is a simple busy loop with no memory accesses. For *loadcache*, each process strides through a memory region performing read-modify-write cycles on successive cache lines. The size of the region is 2× the L1 cache size per process, for a total working set of 128 KiB on *S4-S* and 256 KiB on *S4-E*. This results in significant cache pressure, with few accesses to main memory. For *loadmem* we do the same, but increasing the total working set size to 2× L2 cache size, resulting in many accesses to main memory. We use another microbenchmark, *spin*, to investigate the properties of CPU-bound workloads, i.e. those with no periods of idleness. *spin* simply executes a busy-loop for a fixed number of iterations, so in varying OP, the execution time changes but the total work performed is constant.

For all microbenchmarks, the work is split over four processes which are scheduled by the default Linux task scheduler. While this does not enforce any particular assignment of processes to cores, we observe that in practice, the work performed by each core involved in the benchmark is approximately equal. In all cases we report the average of three iterations of the workload, and for all data reported, the relative standard deviation across iterations is less than 6%.

D. Results

The results of *loadcpu* on both platforms are shown in Figures 1 and 2. For a fixed number of online cores, energy is an increasing function of frequency; that is, the lowest energy generally occurs at the lowest frequency attainable for a given number of cores. At a fixed frequency, energy is very weakly dependent on the number of online cores on *S4-E*. However, on the *S4-S*, there is significant energy difference varying the number of cores at fixed frequency, a factor of two in some cases.

On both platforms, at fixed frequency, using a single core (where load is low enough to allow this) is more efficient than 2, 3 or 4 cores, and this disparity increases with frequency (although this effect is much more pronounced on the *S4-S* than the *S4-E*). This is due to a CPU-wide sleep state that

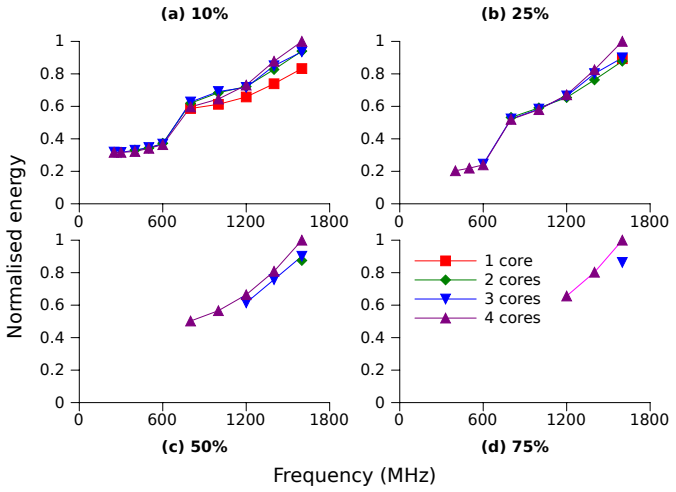

Fig. 2. *S4-E loadcpu* normalised energy vs. frequency (MHz) at 10, 25, 50 and 75% load.

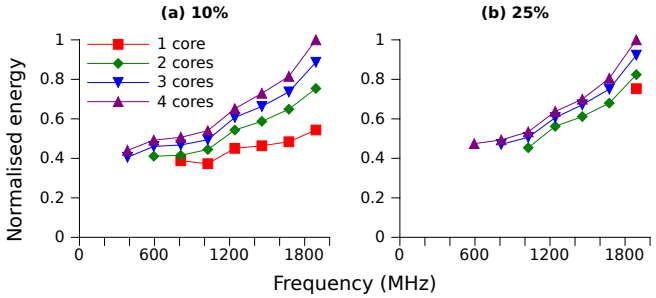

Fig. 3. *S4-S loadcache* normalised energy vs. frequency (MHz) at 10 and 25% load.

significantly reduces power, but can be entered only when cores 1–3 are offline, and core 0 becomes idle. We say more about this in Section III-C. Minimal energy is achieved with more than the minimal number of cores active in cases where this allows frequency to be further reduced. On *S4-E*, note a discontinuity from 600 to 800 MHz; this is where the processor switches from the little (A7) to big (A15) CPU cluster.

In Figures 3–5 we show a subset of the *loadcache* and *loadmem* results. On *S4-S*, adding significant cache pressure does not noticeably change the shape of the graphs. However, with added main memory access in Figure 4, we observe that

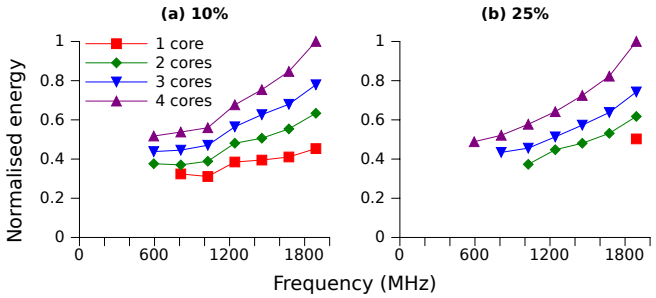

Fig. 4. *S4-S loadmem* normalised energy vs. frequency (MHz) at 10 and 25% load.

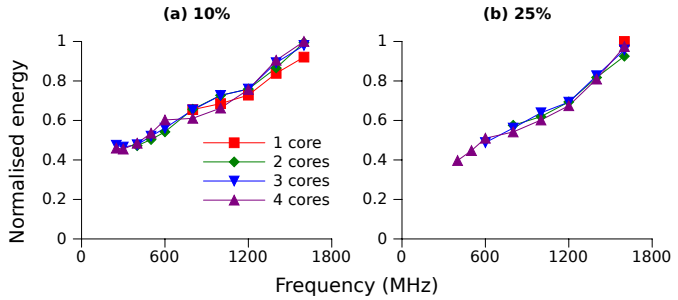

Fig. 5. *S4-E loadcache* normalised energy vs. frequency (MHz) at 10 and 25% load.

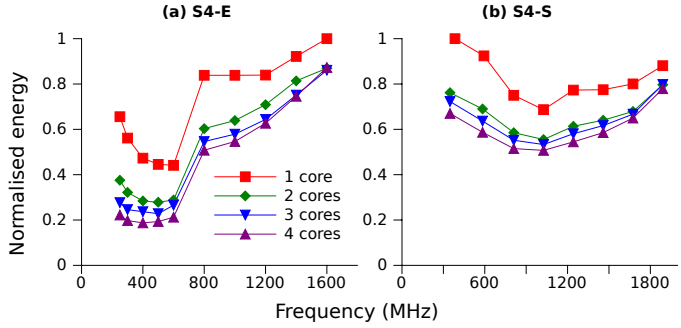

Fig. 6. *spin* normalised energy vs. frequency (MHz) for (a) *S4-E*, and (b) *S4-S*.

the energy difference between the number of online cores has increased approximately uniformly across the full range of frequencies. Furthermore, under cache pressure, the discontinuity on *S4-E* from 600–800 MHz caused by the CPU cluster switch does not appear. In other respects, the *loadcache* and *loadmem* results do not differ from *loadcpu* in any significant aspect.

The results of the *spin* benchmark for both platforms are shown in Figure 6. While energy is a complex function of frequency, maximising the number of online cores is always optimal for this workload, both globally and at each frequency. The discontinuity in (a) is again due to the CPU cluster switch from 600 to 800 MHz. While the power curve is reasonably continuous across the switch (see Figures 2 and 5), the performance curve shows a significant discontinuity, resulting in this effect on the energy curve. The use of little cores can significantly improve energy efficiency, more than a factor of two if the correct OPs are selected. However this of course results in an increased workload runtime.

III. Modelling and Analysis

In this section we analyse the results from Section II, and provide a theoretical framework in which to understand them.

A. CPU Model

We use a simple multi-core CPU power model in which each core can be in one of three states, *active*, *idle* and *offline*. Active is the state in which instructions are executing. The offline state is the deepest sleep state in which the core and all supporting circuitry is fully powered down. Idle is the shallowest sleep state in which the core is ready to perform computation but not actually executing instructions. These states correspond roughly to ACPI C-states C0, C1–C(x-1) and

Cx respectively (if x is the deepest C-state). Each core may be in a different power state, but we assume that the CPU has a single voltage and frequency plane, and hence all cores are clocked at the same frequency and supplied with the same voltage. We assume that the idle state can be entered and exited instantaneously with no performance or energy overhead, and that this is done immediately when no computation is available to run. Further we assume that the offline state has a high entry and exit cost and thus is used at a coarse granularity under control of the system-wide power management policy.

We model the power consumption of an n-core CPU at frequency f as:

$$P_{\text{CPU}} = P_{\text{uncore}} + n(P_{\text{dynamic}} + P_{\text{static}}) \ . \tag{1}$$

P_{static} is the power consumed by a core when it is online but otherwise idle. It is workload independent, but varies with core voltage. P_{dynamic} is the additional power consumed when a core is active, and is dependent on workload and voltage/frequency. The sum of the dynamic and static components forms the per-core contribution to total power consumption. The remaining CPU power consumption is independent of the number of online cores, and hence called the *uncore*, denoted P_{uncore}. The uncore must remain powered as long as any core is online, and typically includes last-level caches, buses, etc. We can express P_{dynamic} by the well-known equation

$$P_{\text{dynamic}} = C_{\text{eff}} f V^2 \ , \tag{2}$$

where C_{eff} is the effective switching capacitance, f is the core frequency, and V is the core voltage.

This model is based on Gupta [8], but others have used similar models [19], [20]. Importantly, we are using a *functional* power model, and not a physical one: we are interested in how power appears to be consumed from the OS perspective, rather than the location or subsystem to which the corresponding circuit belongs. For example, if the transistor leakage power of a core persists whether or not that core is online, such as if all cores share a power plane, then functionally we consider this uncore power, since it is independent of n. We discuss the validity of this model in Section III-C.

B. Analysis

The energy cost of choosing an incorrect operating point can be substantial, a result well known from the single-core DVFS literature [17]. Our results 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, can lead to catastrophic results, in some cases (e.g. Figure 4(a)) a doubling of energy consumption!

As the experiments of Section II-D show, energy is generally minimised by running at the lowest possible frequency on both platforms. However, to meet performance requirements, cores must be onlined to offset the capacity lost by reducing frequency. Put another way, onlining cores allows us to run workloads at lower, more energy-efficient frequencies for equivalent throughput. This is the primary mechanism by which running more cores can reduce energy consumption.

We can develop an understanding of how this occurs using the above CPU model. Treating the workload as periodic, using Equation 1 and 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

$$E_{\text{CPU}} = P_{\text{uncore}} T + n(P_{\text{dynamic}} t + P_{\text{static}} T) \ . \tag{3}$$

Using the approximation [18] that

$$P_{\text{dynamic}} \propto \frac{\text{instructions}}{\text{cycles}} \tag{4}$$

for fixed frequency f, it follows that E_{dynamic} is proportional to the number of executed instructions, which is constant for a fixed workload. So if we execute i instructions spread across n cores, then

$$E_{\text{dynamic}} \propto i/n \ , \tag{5}$$

and substituting into Equation 3, we get

$$E_{\text{CPU}} = (P_{\text{uncore}} + nP_{\text{static}})T + ie \ , \tag{6}$$

where e is the proportionality constant of Equation 4, corresponding to the per-instruction energy, and ie is constant for a given workload. Thus, if P_{static} is small compared to P_{uncore}, then E_{CPU} is independent of n. In the following sections, we show this to be true on ***S4-E***, and then deal with the case where it is not. Note that for the purposes of this analysis we can treat P_{uncore} as constant, since adding the dynamic uncore contribution would only increase the uncore/static ratio.

For workloads with no periods of idleness (i.e. compute-bound, such as our *spin*), 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 effect, which is well-documented in the DVFS literature [14]. However, with DVFS dynamic power is super-linear in frequency (since $P \propto fV^2$ and V is monotonic in f) and hence race-to-idle with frequency 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. Assuming scalability[1] ($t \propto 1/n$ where t is the execution time), then from Equation 1 we get

$$E_{\text{CPU}} \propto \frac{P_{\text{uncore}}}{n} + P_{\text{dynamic}} + P_{\text{static}} \ , \tag{7}$$

and hence increasing n will always decrease E_{CPU}, for workloads with ideal scalability. The results of Figure 6 show this clearly. Determining the optimal frequency however, depends on the relative values of the power terms, and we claim this requires a full multi-core dynamic power model, which is currently an open problem.

C. Model validation

In the previous section, we established that if P_{static} is low, then CPU energy is minimised by onlining as many cores as required to run at the lowest possible frequency. To validate this assumption, we directly measure the static power on both platforms.

[1] We use the term *scalability* specifically in terms of the effect of hardware resource sharing, such as L2 cache and memory buses. The algorithmic scalability of a workload is not relevant to this discussion.

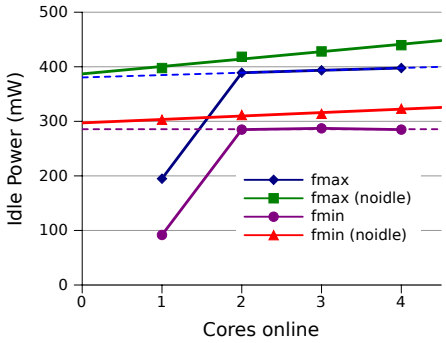

Fig. 7. *S4-E* idle power for the Cortex-A15 (big) CPU cluster at minimum (800 MHz) and maximum (1600 MHz) frequencies, and with idle states enabled and disabled (noidle).

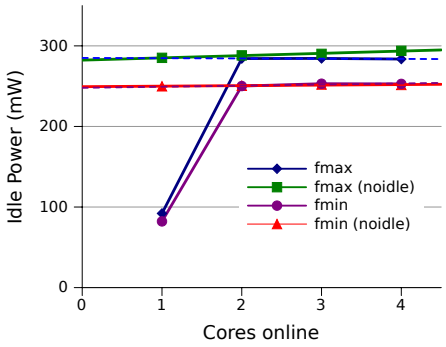

Fig. 8. *S4-E* idle power for the Cortex-A7 (little) CPU cluster at minimum (500 MHz physical) and maximum (1200 MHz physical) frequencies, and with idle states enabled and disabled (noidle).

Figure 7 shows idle power consumption as a function of the number of online cores for the *S4-E* platform running with the big (A15) cores only, at maximum (1600 MHz) and minimum (800 MHz) frequencies for that cluster. From such a graph, the per-core static power consumption, P_{static}, can be determined from the gradient of the curve, while the uncore power, P_{uncore}, corresponds to the y-intercept. Shown are two data sets for each; one in the normal configuration, and one with all idle states disabled.

With idle states disabled, we see that P_{static} is approximately 14 mW per core at $f_{\max}$, and 6 mW at $f_{\min}$, determined by dividing the difference in power consumption from 2–4 cores by 2. This is a very small fraction of uncore power, only 3.5% and 2.1% respectively, so this platform indeed has low static power. From our previous analysis we thus expect to see that on *S4-E*, onlining cores consumes *less* energy, and Figures 2, 5 and 6 demonstrate exactly that. Furthermore, the small difference in power consumption between the normal and idle-disable cases (worst-case of 42 mW for 4 cores at $f_{\max}$) shows that our earlier assumption of a 3-state core (offline, idle, active) holds on this platform. Moreover, this validates our assumption that idle state entry and exit cost is negligible, since disabling them entirely does not significantly impact power.

We measured the same data for the small (A7) CPU cluster on the *S4-E*, running at maximum (1200 MHz) and minimum (500 MHz) physical frequency, plotted in Figure 8. This shows negligible P_{static} even with idle states disabled (3 mW per core @ $f_{\max}$), and a worst-case difference between idle and noidle of 10 mW.

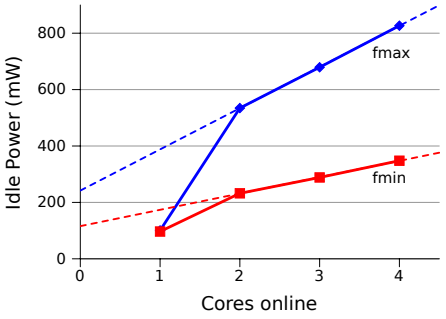

Fig. 9. *S4-S* idle power at minimum (386 MHz) and maximum (1890 MHz) core frequencies.

Figure 9 shows idle power as a function of online cores for the *S4-S* platform, at maximum (1890 MHz) and minimum (386 MHz) frequency. The per-core static power is 146 mW per core at $f_{\max}$, and 58 mW at $f_{\min}$, corresponding to 60% and 50% of P_{uncore}, respectively. This is certainly significant, and hence we do not necessarily expect to see that maximising the number of online cores yields minimum energy, and our microbenchmark results reflect this, particularly Figures 1, 3 and 4 at 10 and 25% load levels. We deal with this issue in Section III-D. This figure shows the measurements with all core idle states enabled, however disabling them increases the idle power by at most 2%. This further validates our 3-state CPU model.

As the idle states provide little power saving, race-to-idle is not beneficial. For periodic workloads, this implies that minimising power is equivalent to minimising energy. Moreover, since all power terms are *increasing* functions of f, then for fixed n, energy is minimised by minimising f. The same does not apply however to CPU-bound loads, because the entire CPU can be powered down and hence race-to-halt remains potentially effective.

It is certainly surprising to see such a big difference in static power with two latest-generation implementations of the same architecture, by manufacturers who are both major players in the smartphone processor market! One explanation of this is that in the case of *S4-E*, all cores share a power plane (i.e. they are connected to the same power supply), so they are powered regardless of the power state of the core. This means that the leakage power, a significant aspect of overall power consumption, can not be reduced by offlining a core. On the other hand, the *S4-S* powers each core from independent supplies and hence leakage can be reduced in offline cores. The latter is consistent with per-core DVFS, where cores require different supply voltages when running at different frequencies.

The significant reduction in power seen on both platforms with a single idle core is due to the previously mentioned "package idle state", which requires core 0 to be idle, and cores 1–3 to be offline. The existence of low-power idle states such as this means that the energy-optimal OP may in fact not be at minimum frequency, due to race-to-idle reducing static energy loss. Specifically, there is a trade-off between the energy saved by completing the workload faster and entering the low-power state as soon as possible, and the additional dynamic energy consumption caused by running at a higher frequency. We claim that in general, solving such an optimisation requires an

TABLE II. AVERAGE COST OF OPERATING POINT TRANSITION PAIR.

	Time (ms) ±1%		Power (mW) ±4%	
	S4-E	*S4-S*	*S4-E*	*S4-S*
offline at $f_{\min}$	22.6	11.7	357	454
offline at $f_{\max}$	16.5	4.09	1305	1926
$f_{\min}$ to $f_{\max}$	2.10	11.1	1033	420
$f_{\min}$ to $f_{\min+1}$	0.39	6.05	320	317
$f_{\max}$ to $f_{\max-1}$	1.20	0.24	1400	838

online dynamic power model (such as Koala [17]). However, on our platforms, running above minimum frequency does not result in appreciable energy savings, and this can be observed directly from the data in Section II.

We asserted earlier that the cost of transitioning between the online and offline states is significant, and thus the mechanism can only be used in a coarse-grained fashion. To validate this, we perform an experiment where we on- and off-line a core 1000 times, measuring the run time and power consumption. Table II shows the results. Also shown is the cost to transition between core frequencies: from minimum to maximum, between the two lowest frequencies, and between the two highest frequencies. In each case we repeat the experiment 3 times and show the average cost of a transition pair (i.e. online and offline or frequency increase and decrease); our methodology does not allow us to determine the individual cost of each operation. For all power data, the relative standard deviation is $< 4\%$, while for time it is $< 1\%$.

D. Adapting for high static power

We have shown both theoretically and empirically how to optimise the OP for a processor with low per-core static power. If static power is significant however, our previous argument does not apply. We now develop a heuristic that can be used to predict an efficient OP for such processors.

Earlier we argued that for periodic workloads, minimising P_{CPU} is equivalent to minimising E_{CPU}. If we treat n and f as continuous variables, then the optimal operating point can be determined by minimising $P_{\text{CPU}}(f, n)$. If we assume scalability (i.e. $n \propto 1/f$), then we can express n in terms of f as

$$n = \frac{n_{\max} f_{\max} u}{f} \equiv \frac{\gamma u}{f} \ , \tag{8}$$

where u is the total system utilisation. We can then minimise P_{CPU} by finding the f that solves

$$\frac{d}{df} P_{\text{CPU}}(f) = 0 \ . \tag{9}$$

Substituting Equations 2 and 8 into Equation 1, we get

$$P_{\text{CPU}} = P_{\text{uncore}} + \gamma u \left(C_{\text{eff}} V^2 + \frac{P_{\text{static}}}{f} \right) \ , \tag{10}$$

where P_{uncore}, P_{static} and V are functions of f, γ is a constant, and C_{eff} and u are workload-dependent. Differentiating with respect to f:

$$P'_{\text{CPU}} = P'_{\text{uncore}} + \gamma u \left(2 C_{\text{eff}} V V' + \frac{P'_{\text{static}} f - P_{\text{static}}}{f^2} \right) \ . \tag{11}$$

P_{uncore}, P_{static} and V can be measured, and their derivatives determined numerically, so the f that minimises P_{CPU} is a function of the remaining variables, u and C_{eff}, both of which are workload-dependent. u can be trivially measured online,

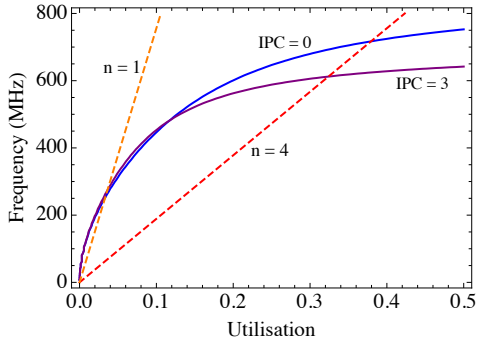

Fig. 10. Optimal operating frequency predicted by our power model for *S4-S* as a function of utilisation for minimum (0) and maximum (3) IPC. The dashed lines delimit valid operating points.

and C_{eff} can be approximated as a function of instructions per cycle (IPC), such as is done by Gupta [8].

We follow a methodology similar to that of Spiliopoulos et al. [18]: we run a series of compute-bound benchmarks at minimum and maximum frequency, measuring both IPC and power. We use the EEMBC telecommunications benchmark suite, which is characterised by minimal L2 cache and memory activity. This minimises any variation in P_{uncore} from the static measurements we made in Section III-C.

We verify this directly by measuring the L1 cache miss rate, but since the L2 and memory clocks do not scale proportionally to the CPU clock, we can also see this by comparing the measured IPC between the iterations at maximum and minimum frequency, which vary by at most 3%. Across all 16 benchmarks, we observe IPC values between 1.0 and 2.3. From the measurements, P_{dynamic} can be determined by subtracting the known P_{static} and P_{uncore} from the total power. Using Equation 2 and a least-squares linear regression, we then produce a function of IPC to C_{eff}, with $R^2 = 0.73$.

Finally, we add this model to Equation 11, and solve $P'_{\text{CPU}}(u, \text{IPC}) = 0$ for f; in other words, given a workload characterised by its IPC and utilisation u, we can predict the frequency f that minimises power, and hence energy consumption. Figure 10 plots the optimal frequency as a function of utilisation for the two IPC extremes, 0 and 3 (since the *S4-S* is a triple-issue pipeline). Note that not all points on the graph correspond to valid operating points. For example, at $u = 1$, all cores are required running at maximum frequency to achieve the necessary throughput. At $u = 0.5$, an operating frequency of at least 945 MHz (half of $f_{\max}$) is required. The dashed lines show the range of valid operating points, corresponding to one or four online cores.

From Figure 10 we observe several interesting characteristics of the optimal frequency. Within the valid operating points, the characteristic IPC of the running workload does not significantly affect the optimal OP. Thus, for the remainder of the paper, we assume a fixed IPC at the midpoint 1.5. At low utilisations, the optimal response to increasing load is to increase frequency. At higher u, the optimal frequency grows slower than utilisation demands, requiring onlining of cores to provide the necessary capacity.

We propose a simple algorithm, characterised by a *frequency threshold*, f_{thresh}, to track the optimal operating point. Figure 11 shows graphically how the proposed algorithm

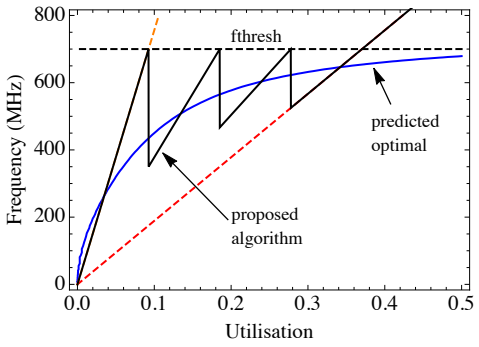

Fig. 11. Proposed algorithm, shown as the target frequency as a function of utilisation. The spikes in frequency correspond to the onlining/offlining of cores.

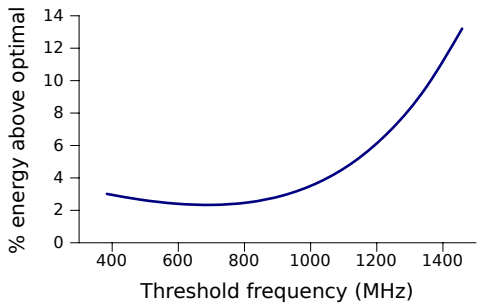

Fig. 12. Predicted energy consumption of proposed algorithm for varying threshold frequency (f_{thresh}), as a percentage above predicted optimal.

works, for a hypothetical f_{thresh} value of 700 MHz. As utilisation increases from 0, the frequency is increased on a single core. When the threshold frequency is reached, rather than exceed it, a second core is onlined. At this point, the frequency can be halved to achieve the same throughput. As utilisation increases from this point, we again scale frequency (now on two cores) until the threshold is reached, where the third core is onlined. Once all cores have been onlined in this manner, we have no choice but to increase frequency.

This policy has the advantage of simplicity: it has a single, well-defined tunable parameter f_{thresh}, it requires no workload-specific knowledge, and as we will show in the following section, it is reasonably simple to implement. However, we require a methodology to determine an appropriate threshold frequency.

First we use our power model to predict, for each possible f_{thresh}, the average energy consumption under the policy. This is done by integrating the sawtooth curve shown in Figure 11 over utilisation.[2] Then we compute the average energy consumption of the predicted optimal (the smooth curve of Figure 11), averaging over utilisation. Finally, we plot the difference between this average and the proposed algorithm, shown in Figure 12 as a percentage increase in predicted energy consumption.

From Figure 12 we see that the optimal f_{thresh} selection lies in the 600–800 MHz range. Moreover, we see that threshold frequency selection is not particularly sensitive: across the 400–1000 MHz range, energy varies by only about 1% of optimal, but above 1000 MHz the energy loss is significant. For the remainder of the paper we will use $f_{\text{thresh}} = 810$ MHz, the nearest supported frequency on the *S4-S* at the upper end of the predicted optimal frequency range.

[2]We only average over the utilisation range 0–0.6. Above that the algorithm diverges quickly from the predicted optimal due to the minimum frequency requirements imposed by high utilisation.

E. Summary

We have shown that running a workload on more cores can reduce energy consumption by two mechanisms:

1) allowing access to lower, more energy-efficient core frequencies for equivalent throughput; and
2) reducing execution time and thus minimising the energy contribution of the per-core static power and the chip-wide uncore.

This result is enabled by low per-core static power, which we have shown to be true for the *S4-E*. From this it follows that the optimal policy for such devices is to run all cores at the minimum frequency where utilisation is $\leq 100\%$, offlining cores only when they idle. On *S4-S* however, we observed comparatively high per-core static power, so the same policy is not energy-optimal on that platform. However, we can predict the optimal OP from a processor power model. We proposed an algorithm utilising this, which we now implement and evaluate.

IV. Medusa: An Offline-Aware Frequency Governor

Based on our insights, we have implemented an energy management policy, *medusa*, in the Linux kernel running on both Galaxy S4 platforms. The goal of this policy is to control the CPU frequency and number of online cores to minimise energy consumption by managing CPU slack time, but without adversely affecting performance. The implementation is a Linux "cpufreq" DVFS governor which controls frequency in the standard way, but also configures the number of online cores with the `cpu_up()` and `cpu_down()` primitives.

The policy selects a new operating point based on recent history of the number of runnable threads and load on each core. When the number of online cores does not need to change, the frequency selection algorithm is very similar to Linux's *ondemand* or *conservative* governors; specifically, attempt to choose the minimum frequency that keeps utilisation $\leq 100\%$. To select the number of cores to run, medusa uses as input the platform-specific f_{thresh} to attempt to maintain the following invariants:

1) allow $f > f_{\text{thresh}}$ only if all cores are online; and
2) if $f < f_{\text{thresh}}$, then only one core should be online.

Setting $f_{\text{thresh}} = 0$ corresponds to a policy of onlining all cores before increasing frequency at all—such a policy applies for systems with very low P_{idle}, such as the *S4-E*. On the other hand, setting $f_{\text{thresh}} = \infty$ corresponds to maximising frequency before onlining any additional cores, useful only on (probably fictional) systems where P_{dynamic} is sub-linear in f.

In more detail, medusa runs every 100 ms, and executes the following algorithm:

1) Select a candidate frequency f_{new}.
2) If $f_{\text{new}} > f_{\text{thresh}}$, and the number of runnable threads exceeds the number of online cores, online an additional core.
3) If $f_{\text{new}} \leq f_{\text{thresh}}$, predict the frequency f'_{new} that would be required to run the current load with $n-1$

online cores. If $f'_{\text{new}} \leq f_{\text{thresh}}$, then offline one core and switch to f'_{new}. Otherwise, switch to f_{new}.

4) If any core is below 5% utilisation, offline a core.

The candidate frequency f_{new} is that which is predicted to maximise utilisation $< 100\%$. f'_{new} is chosen by:

$$f'_{\text{new}} = \frac{n}{n-1} \times f_{\text{current}} \times u_{\text{avg}} \ ,$$

where u_{avg} is the current average utilisation across all online cores, and $n/(n-1)$ is the decrease in compute capacity when switching from n to $n-1$ cores. We selected 100 ms on that basis that other implementations use similar values: we observed sampling periods of 50–500 ms, depending on the device.

In practice, we apply averaging and hysteresis to most calculations to improve OP stability. We also attempt to increase frequency and online cores more aggressively after load increases to improve responsiveness. The implementation, including extensive configuration and debug support, is 1500 lines of C code.[3]

For *S4-E*, we observed that P_{static} is very low and thus where possible, onlining cores before increasing frequency is preferred. This is achieved by setting $f_{\text{thresh}} = 0$. On *S4-S*, we use $f_{\text{thresh}} = 810\,\text{MHz}$, which we selected using the results in Figure 12.

V. Evaluation

We now evaluate the energy consumption under medusa, comparing it against some existing policies. We also investigate the selection of f_{thresh}, including a sensitivity analysis.

Two factors affect energy consumption: selection of the optimal OP for a workload, and adapting to workloads changing over time. In the present work we focus entirely on the first problem, however from an implementation perspective, these same mechanisms address both issues. Furthermore, quality of service is a significant aspect for policies deployed in real devices. For example, it may be desirable from a user interface perspective to increase the OP above energy optimal to provide a certain degree to performance headroom so that sudden changes in load appear smooth. Again, such trade-offs are outside the scope of this paper.

A. Benchmarks

We evaluate medusa with a series of benchmarks of three types. In the first case, we use the *loadcpu* workload at the 10, 25, 50 and 75% load levels. For each, we compare the performance of medusa with the *static-optimal* setting, where the OP is fixed to a particular frequency and number of cores for the full run, corresponding to the setting that minimised energy consumption from our results of Section II. Since this synthetic workload is homogeneous, static-optimal is optimal in this case. We also compare with the performance of the policies that ship with the devices, which we call *default*. On *S4-S*, this consists of the *ondemand* frequency governor, plus a (closed source) userspace daemon called *mpdecision* which controls the number of online cores. On *S4-E*, OP is, like medusa, controlled entirely in the kernel with a modified version of *ondemand*.

[3]The code is available for download at http://ssrg.nicta.com.au/projects/energy-management.

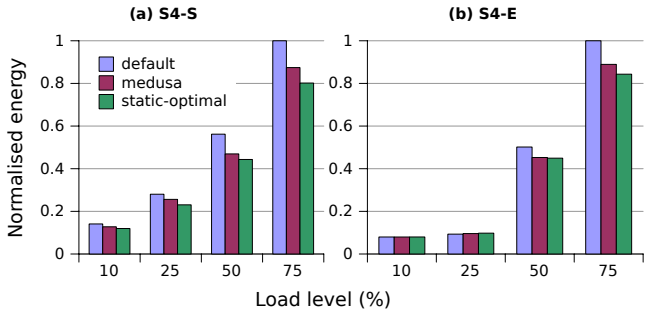

Fig. 13. Energy consumption of *loadcpu* on both devices under three energy-management policies. For static-optimal the frequency and number of cores is fixed.

Secondly, we use a video playback application configured to use a software decoder, and play an H.264-encoded 60 second video. We compare medusa with default as above, but also compare with two of the standard Linux DVFS governors, *ondemand* and *conservative*. Since these control only frequency, we repeated each four times, statically setting the number of online cores to 1, 2, 3 and 4. We report the minimum energy across the four scenarios.

Finally, we use two benchmarking applications, *AnTuTu 3DRating* and *Vellamo HTML5*. For medusa and default we measure the energy consumption for each benchmark and also report the result "score" as an attempt to quantify the impact on performance which, as discussed above, is largely due to different approaches in responding to workload variation. In the case of Vellamo, since we are running in airplane mode, the parts of the benchmark that require network connectivity are not used.

All of these benchmarks are multi-threaded. For *loadcpu*, as earlier, the total work is split over four processes. For video, AnTuTu and Vellamo, the degree of parallelism is controlled by the applications themselves. The related configuration options are left to their default values. For Vellamo, AnTuTu and video, the screen is on at minimum brightness.

B. Results

Figure 13 shows the performance of medusa with the *loadcpu* workload. On average, medusa achieves within 8% (11% worse case) of static-optimal on *S4-S*, and within 1% (5% worst-case) on *S4-E*. The default policy is on average 23% above optimal on *S4-S* (27% worst case) and 6% on *S4-E* (with 19% worst-case).

Figure 14 shows the results of the video playback benchmark. On both platforms, medusa has the lowest energy consumption: 88% of default on average. For ondemand, the minimal energy is at 1 core on *S4-S*, and 4 cores on *S4-E*, and for conservative, 2 cores on *S4-S*. We have no data on *S4-E* for the conservative governor, which appears to significantly over-estimate frequency on this platform. The optimal number of cores is consistent with our expectations: on *S4-E*, onlining all cores yields minimum energy, whereas for *S4-S*, that is not necessarily true.

Figure 15 shows the results of AnTuTu 3DMark and Vellamo HTML5 of medusa relative to default, for both energy and benchmark score. In all cases, medusa has both a lower energy consumption, and lower score, than default. On *S4-S*, medusa reduces energy by an average of 26%, while the

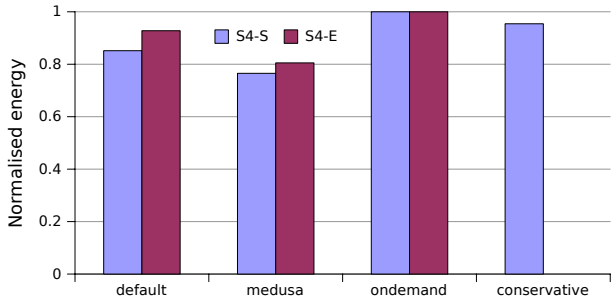

Fig. 14. Video playback energy consumption on both devices under medusa and default, as well as Linux standard ondemand and conservative governors with the number of cores set statically to that which minimises energy.

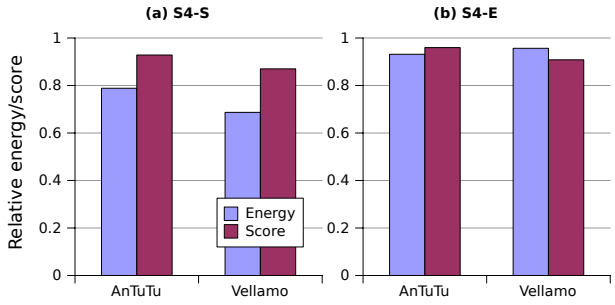

Fig. 15. AnTuTu 3DMark and Vellamo HTML5 energy consumption and score results of medusa, relative to default.

score decreases by only 10%. For *S4-E*, the energy and score decrease by 6% and 7% respectively.

We have already shown that medusa performs well compared with other policies under reasonably static workloads such as *loadgen* and video playback. By contrast, Vellamo and AnTuTu are dynamic workloads and as such, they are sensitive to the algorithm used to react to changing load. However, by using the benchmark score generated by these programs, we can attempt a quantitative evaluation of medusa's performance under such workloads. Using $\frac{\text{score}}{\text{energy}}$ as an overall figure of merit, default and medusa perform within 1% of each other on *S4-E*, but on *S4-S*, medusa outperforms default by 22%.

C. f_{thresh} *sensitivity analysis*

In our implementation, we determined f_{thresh} for *S4-S* from a model of both the algorithm implemented by medusa, and the predicted optimal OP. To determine the effect of this choice on energy consumption, we performed a sensitivity analysis. Using the *loadcpu* program, we ran five benchmarks at 10, 15, 20, 30 and 40% total system load, and measured the energy consumption using the medusa policy, with f_{fthresh} set to 486, 594, 702, 810, 918, 1026 and 1134 MHz. Figure 16 shows, for each threshold frequency, the average percentage by which each benchmark exceeds its minimum observed energy.

We can see that our selection of 810 MHz based directly from the theory produces the best results, consuming within 1.5% energy of the minimum on average. Furthermore, it appears that the cost of incorrectly setting f_{thresh} is higher if it is overestimated. This is consistent with expectation, since a higher threshold means the core spends more time at higher frequencies, and the power curve grows rapidly in this area due to the quadratic dependence on voltage. This behaviour

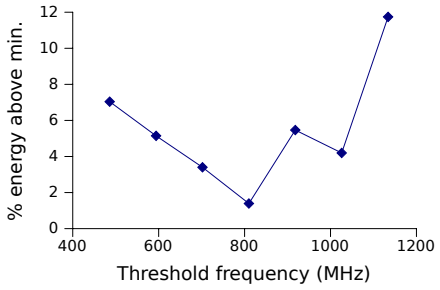

Fig. 16. Energy consumption vs. f_{thresh} selection, as a percentage above minimum.

largely mirrors the predictions made in Figure 12. Specifically, energy is not particularly sensitive to threshold selection within a window of $\approx$ 300 MHz. While the theoretical approach to determining f_{thresh} appears reasonably accurate and provides a good basis for understanding, we concede that the experimental approach is simpler in practice.

VI. Related work

Xu et al. [19] consider the related problem of clusters of DVFS-capable uni-core nodes, and prove that maximising the number of nodes is power-optimal when dynamic power dominates static. For periodic workloads, we show this to be true when static power is low, regardless of dynamic power.

Ghasemazar et al. [6] takes a theoretical approach to the problem of combining offlining and DVFS, developing a control-theoretic feedback algorithm to select an operating point. They demonstrate that increasing the number of online cores always decreases the optimum frequency, and claim that offlining unused cores is important. We show that for some processors, the per-core static power contribution is so low that offlining them saves negligible power, idle or otherwise. Maggio et al. [13] also use control theory to optimise operating point, specifically for the problem of video decoding.

Anderson and Baruah [1] consider the problem of choosing frequency and core count in the context of real-time task execution guarantees. From a dynamic-only power model, they observe that power can be made arbitrarily small by increasing the number of cores and reducing frequency, a similar result to our low-P_{static} observations. For non-trivial per-core static power, our results in Figure 10 indicate that as $n \to \infty$, the optimal frequency remains asymptotic, but tending towards a non-zero value.

Li and Martinez [12] 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 claim that the optimal operating point depends heavily on the power-performance curve of the particular processor. We show this not to be the case where per-core static power is low.

Chen at al. [4] develops algorithms with proven complexity bounds to solve the problem on a theoretical basis, considering also the energy and time cost in switching between operating points. Several authors [5], [10], [16] have considered the problem of co-control of online cores and frequency scaling to maximise performance under thermal or peak-power limitations. These generally used desktop and server-class systems, or simulations thereof.

Le Sueur and Heiser [11] investigate the relationship between DVFS and idle sleep states on several x86 and ARM platforms. They show that on x86, the use of both DVFS and C-states can improve energy efficiency. On various x86 platforms, they show that for periodic workloads, energy tends to be minimised at minimum frequency (combined with the use of C-states). This suggests that an approach similar to medusa may be feasible on some x86 systems. Conversely, Bircher and John [2] show that the power reduction capacity of C-states exceeds that of DVFS. On embedded multi-cores, idle states save little power, so the decision reduces to whether a core should be online or offline. Consequently, minimum frequency is always optimal for periodic workloads since there is no race-to-idle benefit.

Gupta et al. [8] 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 show that this is highly platform dependent, with uncore consuming close to 100% of idle power on *S4-E* and 20–50% on *S4-S*.

VII. Limitations and Future Work

The main limitation of our work is in the assumptions made in the processor power model. While we believe these to be reasonable for a certain class of CPUs (namely, mobile multi-cores), it is unlikely to apply to a wider range of devices. The complexity of idle states in a typical x86 processor means that race-to-idle may be optimal. Implementing a policy on such devices therefore requires a full dynamic power model, which is currently an unsolved problem for multi-core processors.

In the future we plan to extend medusa to incorporate active power management, allowing the trading of performance for a reduction in energy consumption. The most promising approach appears to be to extend Koala [17] to multi-core. It uses parameterised time and power models to make online predictions of a system's performance and energy response to changes in the operating point, based on sampling performance counters.

VIII. Conclusions

We have shown that for mobile multi-cores with low static core power, offlining cores generally leads to increased energy consumption. This occurs for two reasons: one, onlining additional cores allows running at lower, more energy efficient frequencies; and two, that the additional throughput allows the faster completion of a workload, and thus reduces the accumulation of static uncore power. When per-core static power is high, we have shown that up to a certain frequency (which we call the threshold frequency), DVFS is preferable to onlining cores. However, once this frequency is reached, onlining cores is preferred. We showed how to determine the threshold frequency, and that this yields good results.

Based on these observations, we implemented *medusa*, an offline-aware frequency scaling governor, in the Linux kernel running on two Galaxy S4 smartphones; one with an Exynos 5410 SoC, and another with the Snapdragon 600. We showed that medusa is capable of achieving close to optimal energy consumption for static workloads, and compared it with several other policies, including the default policy shipping on these devices, with favourable results.

Acknowledgements

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

[1] James H. Anderson and Sanjoy K. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *24th ICDCS*, Tokyo, Japan, Mar 2004.

[2] W. Lloyd Bircher and Lizy K. John. Analysis of dynamic power management on multi-core processors. In *SC'08*, Kos, Greece, Jun 2008.

[3] Aaron Carroll and Gernot Heiser. Mobile multicores: Use them or waste them. In *HotPower'13*, page 12, Farmington, PA, USA, Nov 2013.

[4] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *12th RTAS*, pages 408–417. IEEE, 2006.

[5] Ryan Cochran, Can Hankendi, Ayse K Coskun, and Sherief Reda. Pack & cap: adaptive DVFS and thread packing under power caps. In *MICRO*, pages 175–185. ACM, 2011.

[6] 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.

[7] Peter Greenhalgh. big.LITTLE processing with ARM Cortex-A15 & Cortex-A7. *ARM White Paper*, 2011.

[8] 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.

[9] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Int. Symp. Low Power Electron. & Design*, pages 38–43, Portland, OR, USA, Aug 2007.

[10] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *39th MICRO*, pages 347–358, Orlando, FL, USA, Dec 2006.

[11] Etienne Le Sueur and Gernot Heiser. Slow down or sleep, that is the question. In *2011 USENIX ATC*, Portland, OR, USA, Jun 2011.

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

[13] Martina Maggio, Henry Hoffmann, Marco D Santambrogio, Anant Agarwal, and Alberto Leva. Power optimization in embedded systems via feedback control of resource allocation. *Trans. Control Syst. Technology*, 21(1), 2013.

[14] 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.

[15] National Instruments Corporation. *NI 622x Specifications*, Jun 2007. 371290G-01.

[16] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: fine-grained power management for multi-core systems. In *36th ISCA*, pages 302–313, Austin, TX, USA, Jun 2009.

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

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

[19] Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. In *LCTES*, Chicago, IL, USA, Jun 2005.

[20] Dakai Zhu, Rami Melhem, and Daniel Mossé. The effects of energy management on reliability in real-time embedded systems. In *ICCAD*, pages 35–40, San Jose, CA, USA, 2004.