A Model-Based Power Governor for Heterogeneous Platforms

Alexei Colin, Milda Zizyte
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA

***Abstract*—Applications on mobile devices have counter-intuitively high resource demands. In the drive to satisfy these demands CPUs with increasingly higher frequencies target such devices. However, the new application possibilities for the user opened by the gains in performance, are counter-balanced by decreased battery life. This situation has inspired hardware and software designs that aim to deliver the same performance at lower energy consumption. One of the sources exploited for energy savings is variability in the degree to which the workload is CPU-bound. Armed with an estimate of the resources required by the task, it is possible to scale down the frequency, hibernate or shutdown a processing unit, or migrate to a low-frequency and energy-saving core. We propose a model-based approach to make all of the above decisions and evaluate it on Nvidia Tegra3 platform as well as in a custom simulator. Our power manager achieves better energy savings than the default power management scheme, at the cost of increased lateness.**

I. Introduction

Computational demands placed on mobile devices exceed the demands that were placed on desktop systems in the recent past. A mobile device, such as a smart-phone or a tablet computer, is expected to act as a media acquisition device, an end-point for multimedia consumption, a gaming platform, and a secure communication device. Some of the above applications require a minimum CPU operating frequency to deliver an acceptable user experience and for many an increase in processing capacity results in a quality-of-service improvement. Thus, the market will continue to select for the most raw performance per dollar in the near future.

Increased energy consumption and heat generation that comes at the cost of higher operating frequency is severely constraining this race for performance. A power budget restricts the frequency, f by restricting V^2f, where the lower bound on supply voltage, V, grows with f. Mobile devices are influenced by these effects to the extreme. Their size requirements limit the battery capacity since energy density growth has not matched the growth in demand. The strictly passive cooling has a low dissipation rate and must operate at temperature levels tolerable by humans. Thus, designs of hardware and software platforms that offer the most performance per Watt are in high demand.

One potential source of energy savings is the variability in the CPU demand of the workload over time. Idle phases due to I/O, synchronization, or periodicity are present in most workloads. Furthermore, extra cycles do not always translate into an earlier task completion time due to stalls on memory access. Whenever their utility is sufficiently low, cycles can be spared by scaling down the voltage-frequency level, hibernating or disabling processing units, or switching to hardware that can perform less work per cycle in return for reduced energy cost. The challenge common to all of the above mechanisms is selecting the most efficient hardware configuration based on the workload.

Dynamic Voltage Scaling (DVS) and idle/sleep states are mechanisms that have been widely supported by hardware and a variety of policies have been investigated and deployed. In contrast, a recent addition to the repertoire of implemented mechanisms is offered by Nvidia Tegra 3 SoC: a heterogeneous platform that combines clusters of cores with identical ISAs but different operating frequency range and power characteristics. We propose a model-based power-manager that exploits such heterogeneity by matching the hardware configuration to the workload. The governor chooses the most power-efficient cluster, number of online cores, and frequency.

II. Related Work

The common challenge of saving energy by exploiting the workload variability has spawned a surprisingly diverse set of problem definitions even within policies for the same mechanism. This is due to the several possibilities for kind of information available about the workload and the details of the hardware implementation of the mechanism.

In real-time systems the timing properties of tasks are known at design time and the completion time of a hard real-time task is of equal value regardless as long as it meets its deadline. Multiple heuristics for this scheduling problem have been proposed for finding the schedule that trades off the slack for energy by executing at a lower frequency [1]. Hints from compiler or application have been used to determine relative thread execution speed and slow down the ones that would wait at synchronization points [2]. While the extra information could enable efficient policies, a policy that does not depend on any a priori information or application assistance is more widely applicable and is our focus. Proposed strategies in this category range from simple threshold logic for maintaining CPU load at a constant level employed by the Linux kernel [3] to heuristics based on learning [4] and control algorithms [5].

Long before the physical availability of heterogeneous platforms, heuristics for task placement and task migration have been proposed. Task placement is concerned with finding the task to core assignment with least energy-delay in a model where all cores execute concurrently [6]. Task migration, on the other hand, is concerned with finding one optimal core on which to execute in a model where only one core can be active at a time. The approach in [7] avoids explicit model that maps workload properties to energy consumption per core by sampling potential destination cores for sample periods much shorter than the scheduling quantum. The `auto_hotplug` policy implemented by default in the Tegra 3 driver switches clusters when a low enough VF level is set by the DVFS governor [8].

III. Approach

A common DVFS approach is to gauge the workload by the utilization of the task scheduling quanta in an interval and classify the most recent interval as I/O- or performance-bound. The classification of the next interval is estimated. Low-utilization I/O bound intervals are then executed at a lower frequency. However, on multi-core systems this strategy is insufficient, since not only the optimal frequency but also the optimal number of cores must chosen. The problem becomes yet more complex on heterogeneous multi-core systems, where the optimal subset of cores must be chosen based on their power-performance profiles.

We propose a model-based approach for choosing the most power-efficient hardware configuration given the system load. We present an energy model for the class of heterogeneous platforms where cores are grouped into clusters and each cluster corresponds to a voltage-frequency domain. We employ the model in the implementation of an in-kernel power-management governor. At each time-step, the governor determines the needed computational capacity from current system load and all possible hardware configurations of at least that capacity are compared based on the energy modeled for each for the following time-step.

A. Energy model

We model the power at the level of cores and clusters. Each core contributes static power whenever it is online and dynamic power whenever it is not idle. Each cluster contributes uncore static power and dynamic power (e.g. the clock power). Finally, an explicit term for the base system power is included since all are measurements contain it (see Section IV).

A more formal definition of the model follows along with a listing of inputs and parameters in Table 1.

$$E(z,f,w,u) = P_{\text{busy}}\Delta t + P_{\text{idle}}\Delta t$$

where Δt is the time-step of the governor and P_{busy} and P_{idle} are computed for corresponding values of z and w in the following power model:

$$P(z,f,w) = B + \sum_{i}^{N} z_i \left(S_i^L + D_i^L(f) + P_i^P(z,f,w)\right)$$

name	description
$z_i, z_{i,j}$	on/off for any or of core j in cluster i
f_i	operating frequency of cluster i
$w_i, w_{i,j}$	busy/idle for any or of core j in cluster i
u	total system load w.r.t. to total frequency
N	number of clusters
N_i	number of cores in cluster i
B	non-cpu power
S_i^L	static power of cluster i
S_j^P	static power of core j
$D_i^L(f)$	uncore dynamic power of cluster i
$D_{i,j}^P(f)$	dynamic power of core j from cluster i
K, α	sub-parameters of $D(f)$

Fig. 1. Inputs and parameters for the power model.

where the core-specific component is

$$P_i^P(z,f,w) = \sum_{j}^{N_i} z_{i,j}\left(S_j^P + w_{i,j} D_{i,j}^P(f)\right)$$

and dynamic power is estimated from

$$D(f) = \frac{1}{2} C_L V_{\text{DD}}^2 f \approx K f^{\alpha}$$

For efficient fitting, we approximate this model with a linear function by splitting $D_{i,j}(f)$ into a set of discrete values $D_{i,j,f}$ and by setting $\alpha = 3$, which is a typical value.

As defined, the energy model is applicable to the class of heterogeneous systems composed of cores grouped into clusters with a per-cluster clock domain. Tegra3 processor is one such system, albeit not the most general example due to the constraint of a single active cluster. Figure 2 shows the model predictions when fitted to the data measured on Tegra3 (while clock-gated and with clock-gating disabled, only the former set is shown in the figure). Note that the linear approximation degrades the fit, however, for the purpose of the comparison for the governor, only the relative values are of consequence.

B. Instruction-level heterogeneity

Our power model incorporates differing power profiles of different clusters, however it assumes that the power consumption does not depend on the task being executed. This is a simplification. In fact, in [7] Kumar et. al. argue that on some platforms this instruction-level heterogeneity can be be exploited to yield better energy-delay reduction than an unaware DVFS. Whether this holds depends on the type and degree of heterogeneity among the cores as well as on the variability in the workload.

Our model can be extended with awareness of the match between the task and the core type. To explore this direction, we have investigated how does execution time and power consumption vary for different tasks on Tegra 3 platform. Our results in this direction are presented in V-C.

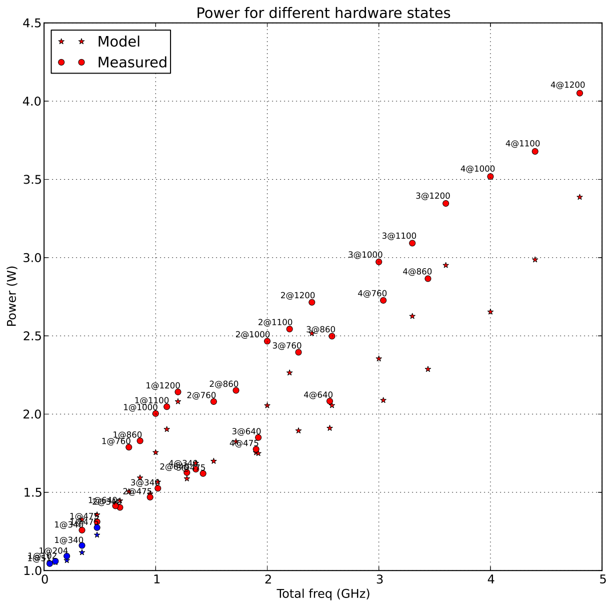

Fig. 2. Measured power values for different Tegra3 hardware configurations and corresponding modeled estimates. Blue configurations are for low-power cluster, red – for high-power cluster.

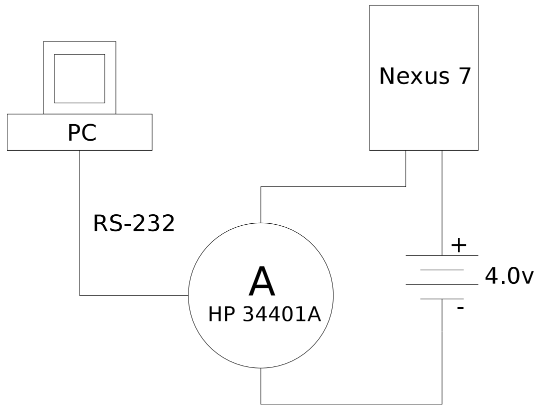

Fig. 3. Power measurement instrument configuration

IV. Experimental Setup

A. Platform

The hardware used is a Google Nexus 7 tablet handheld device with an nVidia Tegra 3 System on Chip. This system is of interest as it is one of the first widely-deployed heterogeneous platforms. Its architecture is composed of two clusters of cores: the high-power one with four cores and the low-power one with one core. All cores are CortexA9 cores, which means they export the same ARMv7 Instruction Set Architecture (ISA) and implement it in the same way. However, the low-power core is manufactured using a "special low-power silicon process" [8], enabling lower power consumption but limiting its maximum frequency to 475 MHz. Each cluster supports DVFS within its own clock and voltage domain shared by all cores in the cluster. The allowed frequency ranges for the two clock domains are $[340, 1200]$ [1] and $[51, 475]$ MHz, respectively.

The hardware imposes several constraints. In particular, only one cluster, either the high-power or the low-power, can be active at any given time. Furthermore, cluster switch can happen only when the high-power Core 0 is the only active core. While our implementation of the governor abides by these constraints, our overall strategy does not depend on this restrictions to hold.

The Nexus 7 device runs a Linux kernel within the Android operating system. The platform driver exports four logical CPUs to the rest of the operating system. Hence, the existence and usage of the low-power is transparent to the architecture-independent layer of the operating system and to user-space applications.

The manufacturer's architecture-specific code implements part of the functionality that traditionally belongs to the kernel, including power policy. In particular, the `auto_hotplug` component defines and implements the policy for the number of online cores. It balances the load and consolidates tasks across cores striving to keep the least number of active cores. The same component decides on when to switch to and from the low-power cluster based on the voltage/frequency level chosen by the DVFS governor

For efficient data collection, we have implemented a `cpusetup` script that implements a set of commands for manipulating and monitoring the state of the CPUs (available on our project website). Any core can be brought online or offline, a switch to the low-power core can be forced, frequency can be scaled to any supported level. This functionality is implemented as sequences of reads and writes from the relevant `sysfs` and `debugfs` interfaces to kernel and the platform driver in particular.

B. Power measurement

To measure power consumption we replaced the device battery with a variable voltage power supply fixed at 4.0V and measured total system current as shown in Figure 3. The power in Watts was calculated as $P = IV = 4.0 * I$. Note that the total system voltage is a constant, despite any voltage scaling, because it is measured on the input side of the regulators that power the CPU. We are restricted to this approach since access to the CPU regulator output is not exposed on the production device.

An HP34401A multimeter was configured to sample the current with an accuracy of 0.01 mA. The sampling period determined by this accuracy level was 36ms. The data was streamed over the RS-232 interface of the multimeter. The C-Kermit scripts for configuration and acquisition as well as the schematic for the customized null-modem cable are available on our project website. We were unable to locate any vendor

[1]The upper limit is 1300 MHz if only one core in the cluster is running, however, we do not make use of this.

software for this purpose.

The above setup implies that our values include the power consumed by main memory and by all system peripherals, including the display, wireless controller, and GPS receiver. Our analysis is sound only for relative comparisons and only when the power consumed by these devices is constant. We ensured the latter condition by configuring a barebones kernel with almost all device drivers excluded (incl. the networking stack) with the assumption that an uninitialized device will consume a constant amount of power or none at all. We verified that the constant-power condition holds by observing the baseline power at a fixed CPU state for several minutes.

For each run, the current data acquisition was launched slightly after the benchmark process and ended slightly before the benchmark process exited. This eliminated the process preambles from the measurement interval since the power consumption is likely to be different during these phases due to e.g. file system access to service page faults or load core dynamic libraries. We observed a standard deviation below 0.03W for all our experiments.

C. Micro-benchmarks

To study the performance at different hardware operating points, we created two micro-benchmarks: a CPU-bound and a memory-bound task. The `perf` tool (packaged with the kernel source) was used to record the number of retired instructions, I, and execution time, T, in seconds, of each run of the benchmark process. The tool uses the kernel performance counter interface that abstracts hardware and software counters.

Performance metric was then calculated as the normalized completion time $\hat{T}$. The process creation overhead was included in the performance metric, however, we treat its contribution as negligible.

Listings 1 and 2 show the respective source: a loop with a sequence of arithmetic operations and a loop with a sequence of loads and stores to far-apart addresses on the heap. The allocated heap segment size and the access addresses were chosen such that the target memory could not be all cached at the same time. The number of iterations was chosen such that the completions times are under two minutes at the lowest frequency.

The CPU-bound task was written in assembly to avoid confounding compiler optimizations and process preambles, the latter of which cause an unwanted variation in the total number of instructions from run to run. The same could be done for the memory-bound task, but is not as crucial, since what the overhead library code would add is more memory accesses.

D. Synthetic workload

To evaluate the in-kernel implementation of the governor (see Section V-A), we defined a simplified synthetic workload. The primary property of the workload is the degree to which it can be rebalaced when core count changes. Since in our current implementation, we model the load as perfectly divisible (to limit the scope), we designed the workload to approximate this assumption.

```
.text
.globl _start
_start:
    mov %r6, $10
.LabelStart1:
    mov %r5, $0x10000000
.LabelStart2:
    mov     %r0, $255
    mov     %r1, $232
    mul     %r3, %r0, %r1
    add     %r3, $214
    rsb     %r3, %r1
    add     %r4, %r3, %r0
    subs    %r5, %r5, $1
    bne .LabelStart2
    subs    %r6, %r6, $1
    bne .LabelStart1
```

Listing 1. CPU-bound microbenchmark

```
#define MEM_SIZE 4096*1024

int main(int argc, char *argv[])
{
    volatile long long x = atol(argv[1]);
    volatile char c;
    unsigned long i = 0;

    char *m = (char *)malloc(MEM_SIZE);
    while (x) {
        c = m[i % MEM_SIZE];
        m[(i+7) % MEM_SIZE] = c;
        c = m[(i+13) % MEM_SIZE];
        m[(i+27) % MEM_SIZE] = c;
        i += 313;
        x--;
    }

    return 0;
}
```

Listing 2. Memory-bound microbenchmark (abbreviated)

The workload consists of multiple threads each consuming from a shared queue of work requests. Work is placed onto the queue by a master thread that reads from a pre-generated work trace file. Before adding a work request to the queue, it is subdivided into a sequence of minimal chucks. Hence, the rebalancing resolution is the size of the chunk, and is independent of the work items in the work trace.

To generate the work trace, we draw inter-arrival times from a geometric distribution ($r = 0.5$) and a work item sizes from a uniform distribution (full range, lower-half, upper-half). Maximum sized work item fully-utilizes the system for one time-step at maximum compute capacity. To measure response-time, the requests are time-stamped in the trace and the worker-threads time-stamp each chunk when it gets processed. Figure 4 shows an example of a work trace.

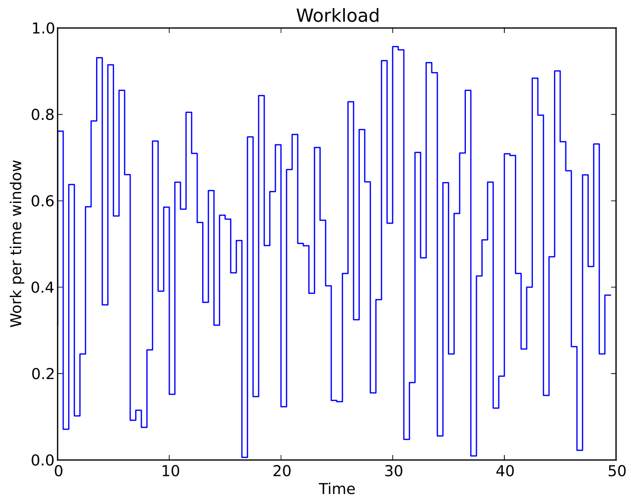

Fig. 4. Example work trace: inter-request times drawn from geometric distribution ($r = 0.5$), request sizes draw from a uniform distribution over full range.

E. Workload characterization

Given the earlier scope of the project, we explored several benchmarking options, including MiBench [9], Imbench, Spec2000, and Parsec. The first goal of Workload Characterization was to analyze what operation was being performed within a time slice. In particular, we wished to determine whether a certain slice of the program was doing a CPU or Memory operation. In order to determine this, we measured Instructions Per Cycle (IPC) over each 50ms interval. This technique relies on the assumption that memory-bound tasks have a smaller IPC than CPU-bound tasks. To measure Instructions Per Cycle, we utlized the unix perf_event_open() utility. We ran a loop which, at every 50ms interval, would poll the perf_event_open() utility for the current number of instructions used and the current number of clock cycles consumed. This was given a specific PID to run with. Thus, to measure workload, a child process running the benchmark may be forked off and the script is then run. Currently, we have only run this utility on an x86 architecture. However, we expect similar results for the Tegra architecture. We ran our perf_event_open() wrapper code agains the MiBench benchmarks, which is useful as it deals speficially with embedded applications, running such tasks as compression and encryption.
While this initial exploration of workload was important to establish future procedures and explore the current direction of work, we recognize the need for further, more refined, usage-specific measurements. In particular, to better describe user interactions with a mobile device, we must create benchmarks which must depend on user input. Tablet applications inherently rely on user input: browsing relies on scrolling and navigating, games rely on interaction, e-mail and calendars rely on navigation, and so on. Moreover, we can exploit the inherent partitioning in these tasks. This partitioning results from waiting for user input. In this stage, the program or application is more likely to be idle, and we may utilize the ability to migrate to a smaller core. Consider the following examples:

- **Browsing**: Without user input, webpage is loaded. Some dynamic content might be displayed, but the CPU is mostly idle. When user input occurs, the application must process a query, make server requests, and render new content. Once the webpage loads, we go back into a near-idle state.
- **Games**: A background (which might be deterministically animating) plays when a user is not interacting with the device. User input requires computation - physics engine computations, loading from buffer, rendering images - depending on the game. Once user input is provided, CPU might do heavier calculation for a while before resuming normal operation.
- **Typing input**: While polling for user input into such applications as calendar or e-mail does not consume much CPU, the user activity is indicative of some active computation later, such as network communication or dat synchronization between apps.

In particular, except in the cases of heavy media playback, which is always loading from memory/network and refreshing screen or speaker output, the device is usually idle without user input. Most applications can then be assumed to be "background processes" until a user makes a specific request. We denote waiting for user input as "I/O" phase. Note that computations may be performed in I/O phase - in fact, this is what differentiates the I/O phase from a sleeping device. Our goal is to pinpoint these partitions of input and non-input in order to optimize our migration strategies. The challenge lies in measuring this usage. A possible approach is to base phase recognition on CPU slice use percentage, like the technique used in the current scheduler for priority assignment. We may assume that if the CPU did not use the entire slice, then it is waiting for user input - that is, in I/O phase. The workload characterization problem thus becomes to assess the variance of this CPU slice usage variable. For maximum impact with respect to our approach, we anticipate that the optimal variance is medium-size lengths of I/O versus CPU-heavy phases. That is, these phases are long enough to justify the overhead of switching cores. We would also like to characterize the variance in order to remove potential spikes that may incorrectly partition the phases. If such spikes occur, future work would be to explore smoothing algorithms to make the data more reliable for our purposes.

V. Results

To evaluate our model-based approach, we fitted the model to Tegra3 power measurements and implemented a governor in the Linux kernel for the Nexus 7 device. In addition, to study the behavior of our scheme without the any confounding effects of the real platform, we have built a custom simulator.

A. *In-kernel*

The governor was implemented in a kernel module and the ftrace subsystem was used for detailed instrumentation. Whenever the module is loaded and the governor is enabled, the CPU load is monitored and periodically the hardware configuration is re-evaluated. If the system load is below the top threashold (95%), then the system is deemed over-provisioned and a more power-efficient hardware state is searched based on the modeled energy. If the system load is above the top threshold (95%), then the system is deemed under-provisioned, and the capacity is increased as follows. First, frequency is scaled to the maximum without changing clusters or bringing cores online. Next, if still under-provisioned, cluster is switched. Next, if still under-provisioned, cores are brought up one by one. This is a conservative strategy minimizes extra switching at the cost of slower adaptability to the load.

To minimize the overhead of model evaluation, the power values for busy and idle case for each hardware configuration was precomputed stored in a radix tree keyed on the configuration value. Although this is not scalable indefinitely, for Tegra 3, the total number of configurations is modest: 1 clusters * 4 cores * 8 frequency levels + 1 cluster * 1 core * 5 frequency levels = 37 configurations.

Figure 5 shows the governor in action. As the CPU load changes, both due to new workload and due to underprovisioning (top), the governor switches to a different cluster, brings cores online/offline, and/or scales frequency (bottom), which results in a changed total compute capacity (middle).

Figure 6 shows a comparison of the proposed scheme with performance governor and ondemand governor coupled with auto-hotplug manager. The performance governor does not modify any hardware state, but keeps the maximum number of cores on at the maximum frequency. The ondemand governor keeps the system utilization at around 70% by scaling the frequency. The cluster switching and core count control is done by architecture-specific auto-hotplug which attempts to keep the least numbers of cores on. While our scheme yields savings in energy at the same computation time, it does so at the expense of timeliness. The large response times indicate that work items are being delayed. The compute time does not increase since there is sufficient slack in the workload (system is not completely utilized) to complete the work before it accumulates.

B. *Simulation*

In order to explore the effectiveness of the governor, a simulator was implemented in python. This program takes as input a core configuration file which describes the voltage and frequency levels at which the cores and clusters of a system operate, as well as a trace file of the amount of new work introduced at every timestep. Using the governor implementation below, the simulator chooses a configuration state and calculates the work done and power consumed based on this state. The power model used is as follows,

$$P_{\text{static}} = P_{\text{on}},$$

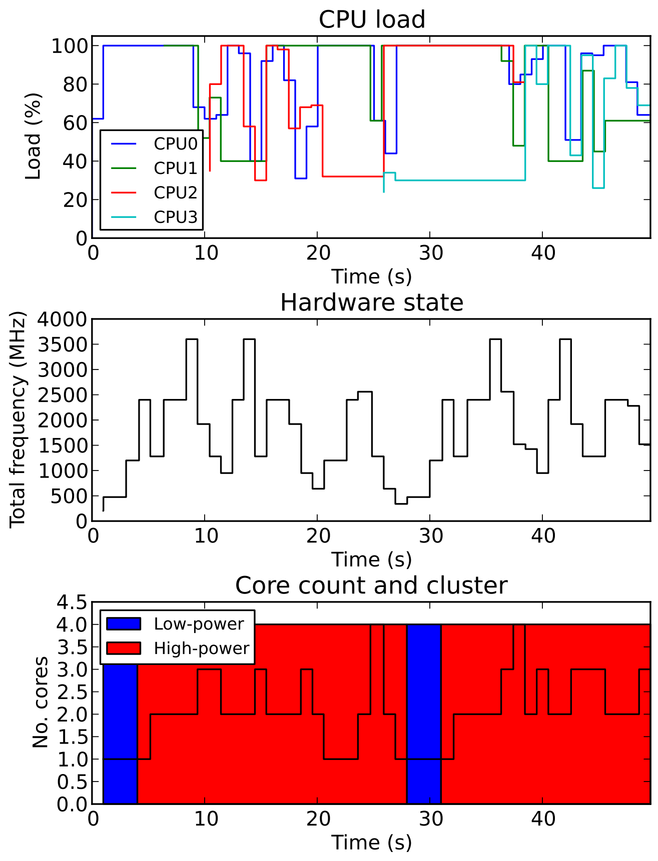

Fig. 5. Model-based governor in action: changes in hardware state through time.

determined by the configuration file,

$$P_{\text{cluster dynamic}} = P_{\text{dyn\_on}} * f_{\text{cluster}} * \alpha,$$

and

$$P_{\text{core } i \text{ dynamic}} = P_{\text{dyn\_on}} * f_{\text{core } i} * \alpha,$$

with total power being

$$P = P_{\text{base}} + \sum_{i=1}^{N_C} \left(P_{\text{static}} + P_{\text{cluster dynamic}} + \sum_{j=1}^{N_i} P_{\text{static}} + P_{\text{core } j \text{ dynamic}} \right),$$

where N_C is the number of clusters, and N_i is the number of cores in cluster i. All of the parameters are determined either from the configuration file or from the current state (number of cores on and their frequencies) of the system.
The remaining work is determined based on the capacity of the current system state, which is simply the frequency of the cores running. At every step, the work from the trace file is added to the remaingn work, while the simulated work completed is subtracted. The work remaining and power used at every time step is recorded and then plotted.

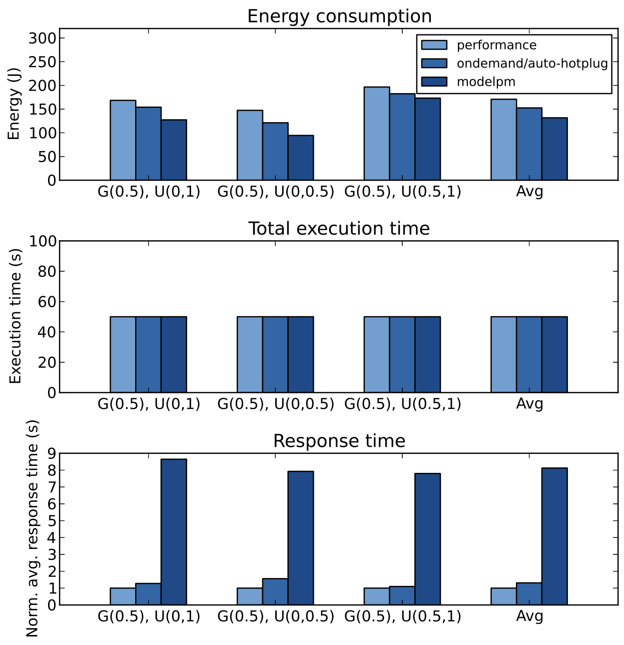

Fig. 6. Comparison of energy, execution time, and response time of the in-kernel implementation of the proposed governor with competitors.

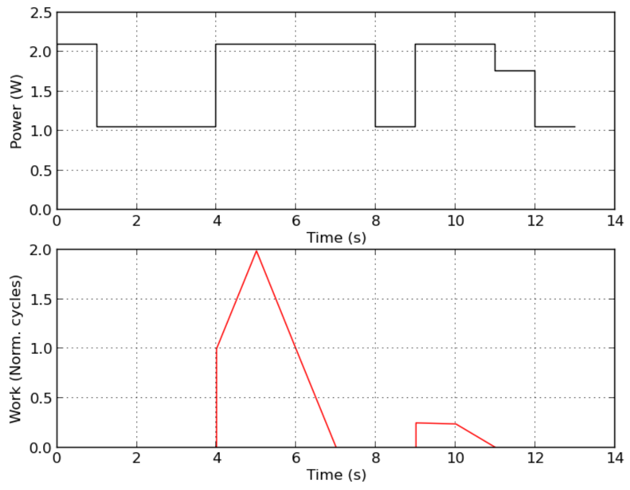

Fig. 7. Performance and power of the memory-bound microbenchmark for each platform configuration

1) Governor Algorithm: The simulated governor algorithm is as follows: At every timestep, we check the load. If the idle time is under 20%, we try to increase frequency. Every time we increase frequency, we increase it to the max level possible. Otherwise, at every time step, we attempt to keep idle time over 30%. We search for the lowest frequency that can sustain this workload. Additionally, these decreases in frequency must be in steps of at least 5% of current frequency. The load of each potential state is computed based on the capacity of the state and multiplied by the current system load.

Figure 7 shows the results of a simulation with trace file workload $\{0, 0, 0, 0, 1, 1, 0, 0, 0, .25, 0, 0, 0\}$. As expected, this graph shows power use increasing with workload. When we have no outstanding work, our power level is at the minimum possible. When our utilization is very high, power is at the maximum it can be (because we scaled to the maximum frequency levels) and stays there.

C. Instruction-level heterogeneity

Next we analyze the extent to which heterogeneity at instruction-level holds on Tegra3. That is, we measure the performance and power consumption of different tasks under the same hardware configuration. Left-hand side of Figure 8 shows that a memory-bound task is less sensitive to a decrease in frequency suggesting that it would be more *energy-delay*-efficient to run it a frequency below the maximum. On the right-hand side of Figure 8 the jumps for both cpu- and memory-bound task confirm that the high-power core always consumes more power than the low-power one. For the memory-bound task this effect is more pronounced, suggesting that memory-bound tasks might be more suitable for the low-power core.

However, we reject our hope that there exist tasks which are more *energy-delay*-efficient when run on the low-power core. The primary difference between the cores on Tegra 3 is the frequency range. The top frequency of the slower core (475 MHz) is lower than the frequency at which memory stalls become a bottleneck and cycles are wasted. Hence, even for the most memory-bound workloads the high-performance core yields the optimal energy-delay (at least at one of its operating frequencies if not at any of them). This fact is confirmed by our measurements for the energy-delay product for different operating points (where most optimal core was chosen whenever choice existed) as seen in Figure 9. The measured optimums are at 1200 MHz and 860 MHz respectively.

Given these results and since the architectures are otherwise identical CortexA9s, we don't expect that there exist applications which could have an energy-delay optimum at an operating point on the low-power core.

VI. Conclusion

Modern mobile devices must satisfy high resource demands without exceeding a strict power budget. However, peak performance is needed only occasionally, while increased battery life is preferred most of the time. Rich gaming and user interfaces require peak operating frequency, while phone calls or media playback do not load the processor. Manufacturers have addressed this trade-off with heterogeneous multi-core platforms, which attempt to match the workload with the most suitable hardware.

However, choosing the correct configuration is a challenge. We have addressed this challenge with a model-based governor, which selects the most efficient hardware configuration by comparing the projected energy consumption of the available

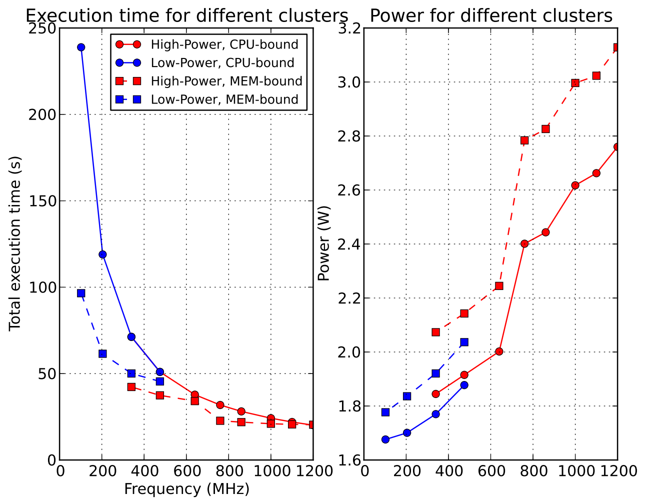

Fig. 8. Performance and power for different benchmarks and cores.

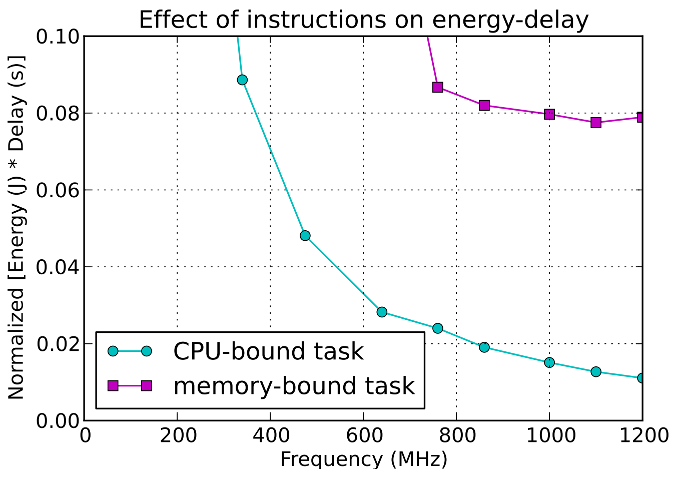

Fig. 9. Energy-delay for different operating points and microbenchmarks.

configurations. We fitted the model to Nvidia Tegra3 platform and implemented the governor in the Linux kernel. Our governor does save more power than the default power manager without increasing total computation time, however it does so at the cost of delayed work.

We have also explored making the power model aware of the task characteristics, however our measurements suggest that this is not exploitable on the architecturally identical clusters of Tegra3. Exploring this direction on a different, more heterogeneous platform is left as future work.

VII. Milestones, Deliverables, Attributions

1) **[Week 1-2][AC]** Learn and control the power management knobs on Nexus 7
2) **[Week 2-3][AC]** Setup power measurement configuration and instruments.
3) **[Weeks 4] [AC]** Measure power for each hardware configuration per cpu/mem-bound microbenchmark.
4) **[Week 5][MZ]** Research potential workloads for evaluation.
5) **[Weeks 6][AC]** Develop a power-model and fit it to Tegra3.
6) **[Weeks 7][AC]** Implement a custom simulator and model-based scheme proof-of-concept.
7) **[Weeks 8][MZ]** Implement default strategies in the simulator for comparison.
8) **[Weeks 9][AC]** Implement the model-based governor in the kernel.
9) **[Week 10][AC]** Measure, evaluate, and compare the in-kernel implementation.
10) **[Week 10][AC,MZ]** Compose report.

Primary code and data delivered are:

1) in-kernel implementation of the model-based governor
2) a custom simulator with strategies implemented in it
3) measurements and fitted model for Tegra 3

References

[1] S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority RT-Systems," in *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings.* IEEE, 2003, p. 106–114. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1203042http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1203042

[2] Q. Cai, J. González, G. Magklis, P. Chaparro, and A. González, "Thread shuffling: Combining DVFS and thread migration to reduce energy consumptions for multi-core systems," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, 2011, p. 379–384. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5993670

[3] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, 2006, p. 215–230. [Online]. Available: http://scourge.fr/mathdesc/documents/kernel/linuxsymposium_procv2.pdf#page=223

[4] G. Dhiman and T. S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," in *Proceedings of the 2007 international symposium on Low power electronics and design*, ser. ISLPED '07. New York, NY, USA: ACM, 2007, p. 207–212. [Online]. Available: http://doi.acm.org/10.1145/1283780.1283825

[5] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, 2007, p. 38–43. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5514266

[6] A. Schranzhofer, J. Chen, and L. Thiele, "Dynamic Power-Aware mapping of applications onto heterogeneous MPSoC platforms," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 692 –707, Nov. 2010.

[7] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, 2003, p. 81–92. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1253185

[8] Nvidia, "Variable SMP: a Multi-Core CPU architecture for low power and high performance," 2011.

[9] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: a free, commercially representative embedded benchmark suite," in *2001 IEEE International Workshop on Workload Characterization, 2001. WWC-4*, Dec. 2001, pp. 3 – 14.