Reconciling Application Power Control and Operating Systems for Optimal Power and Performance

Dominic Hillenbrand, Yuuki Furuyama, Akihiro Hayashi, Hiroki Mikami, Keiji Kimura, Hironori Kasahara, Waseda University - Tokyo, Japan - Tel./Fax. 81-3-3203-4485/4523
Email: {dominic, furuyama, ahayashi, hiroki, kimura}@kasahara.cs.waseda.ac.jp , kasahara@waseda.jp

Abstract—In the age of dark silicon on-chip power control is a necessity. Upcoming and state of the art embedded- and cloud computer system-on-chips (SoCs) already provide interfaces for fine grained power control. Sometimes both: core- and interconnect-voltage and frequency can be scaled for example. To further reduce power consumption SoCs often have specialized accelerators. Due to the rising specialization of hard- and software general purpose operating systems require changes to exploit the power saving opportunities provided by the hardware. However, they lack detailed hardware- and application-level-information. Application-level power control in turn is still very uncommon and difficult to realize. Now a days vendors of mobile devices are forced to tweak and patch system-level software to enhance the power efficiency of each individual product. This manual process is time consuming and must be re-iterated for each new product. In this paper we explore the opportunities and challenges of automatic application- level power control using compilers.

I. INTRODUCTION AND RELATED WORK

In the domain of mobile devices the market is dominated by multi-core SoCs such as Texas Instrument’s OMAP, Qualcomm’s Snapdragon, NVIDIA’s Tegra [1, 2] and Samsung’s Exynos. These SoCs have accelerator- and peripheral-cores for video and audio applications. SoCs for base stations Freescale QorIQ Qonverge and car navigation Renesas SH-Navi3 - for example - are conceptually similar but deploy different domain specific accelerators.

Recent SoCs support various methods for reducing power consumption, such as: DVFS [3] (Dynamic-Voltage-Frequency-Scaling), adaptive body bias [4–6], big-little [7] as well as power- and clock-gating. These power saving mechanisms can often not be independently applied to cores due to resource sharing at the hardware-level. Thus otherwise independent device drivers must be aware of shared clocks and voltage controllers - for example - when they exert power control. Excessive resource sharing may severely reduce the design space of power control.

In the reference [8], the authors projected that in a relatively short time span a significant amount of chip area will remain "dark" - due to power- and parallelism-constrains. New approaches such as near-threshold computing [9] achieve up to 10 times better power efficiency and may help to reduce "dark silicon“. Intel [10] has recently designed a prototype processor that is able to operate from 280mV up to 1.2V (3-915MHz) - thus covering the range from sub-, near- up to super-threshold. In the sub-threshold region leakage dominates and in the super-threshold region dynamic power. The lowest energy per instruction is achieved in the near-threshold region.

Should future SoCs provide scaling from sub- to super-threshold then power will vary more than 10x depending on voltages. Thus DVFS-control - for example - has a large window of opportunities for power reductions in such chips.

In this paper we focus on power control in the open-source Linux- and Android operating system. Both support DVFS through the “cpufreq” [11] device driver. The cpufreq device driver calls user-selectable "governors" to determine new voltages and frequencies. Afterwards the cpufreq device driver invokes low-level device drivers to actually set voltages and frequencies.

Linux has several governors: user-space, ondemand, conservative, powersave, performance and interactive. For our considerations the user-space-governor is most important as it enables DVFS for user-space applications. The ondemand, conservative and interactive-governor provide automatic DVFS-control based on monitoring application activity. The last two governors powersave and performance merely configure the lowest- or highest-performance operating points.

Similar to cpufreq driver, Linux has a cpuidle driver [12] which has two governors ladder and menu. The cpuidle device driver calls the active governor to determine sleep modes for idling. The ladder-governor selects sleep modes in a step-wise fashion. The menu-governor exploits scheduling information which are available when the kernel supports "tickless" mode [13]. In the next section we present a motivational case study to highlight the challenges of user-mode power control [14] in the Linux kernel.

II. MOTIVATIONAL CASE STUDY

The Renesas RPX processor - see Figures 1 and 2 - provides low latency DVFS and clock-gating. Changing the voltage- and frequency registers takes a few microseconds and clock gating merely nano-seconds. In Figure 1 we can see that the
The computing world is moving away from standard computers to more specialized devices. Tablet PCs, smart phones and server processors utilize highly specialized SoCs. Accordingly, power management becomes more specialized as well. Operating systems usually have DVFS- and idle-device drivers for new SoCs - but they are not able to schedule applications and power control efficiently together - simply because the scheduler is unaware of higher-level behaviors.

To escape the dilemma partially one possibility - we propose in this paper - is to use auto-parallelizing compilers such as our OSCAR-compiler [17–21] - for suitable applications. Our OSCAR-compiler generates static task- and data-transfer-schedules - see Figure 3 - as well as power control code with nano-second resolution [22]. OSCAR requires a SoC-specification file to compute the schedules. Thus porting the peripheral cores for DDR, SATA, PCIe, DMA, GPIOs and UART. The chip consumes ca. 3 watts at 648 MHz and 1.15V. In our board configuration the voltage can be scaled in three steps from 1.1 - 1.3V. The frequency is adjustable in four steps: 81 MHz, 162 MHz, 324 MHz and 648 Mhz. The RPX-SoC is supported by two operating systems: Linux 2.6.27 and LWOS. LWOS is a light-weight operating system written by Renesas for internal usage. Linux can only utilize the first processor cluster (4 cores) since cache coherency is not maintained between clusters. LWOS and its applications do not have this limitation and can utilize all 8 cores. The following section we propose compiler assisted power control for some user-space applications and introduce our OSCAR compiler tool-chain.

### IV. Compiler generated power control to the rescue

The following section we propose compiler assisted power control for some user-space applications and introduce our OSCAR compiler tool-chain.

#### OSCAR - Task Schedule

The figures illustrate a static schedule generated by our OSCAR compiler for a heterogeneous SoC with processors and accelerators. The left figure visualizes task dependencies. The boxes represent macro-tasks (MT) which are coarse grained tasks with loops, function calls and basic blocks. The blue colored boxes can be mapped to processors; the green colored boxes can additionally be mapped to accelerators (ACC). The figure on the right shows the schedule for three processors CPU0, CPU1, CPU2 and an accelerator (ACCa). CPU2 offloads tasks to the accelerator and performs necessary data transfers. First, our compiler assigns the ready macro-task MT1 to CPU0. Then MT2 and MT3 are mapped to CPU0 and CPU1. After MT1 finishes, MT2 and MT3 become ready and so forth until all tasks have been executed. Occasionally, “green” accelerator tasks are mapped to processors if the accelerator is unavailable.

### III. Experimental Hardware Setup

For our experiments we have used the Renesas RPX-SoC [15, 16]. This 45nm research SoC - see Figure 2(a) - has eight SH4A processors, reconfigurable ALU arrays, two MX-2 matrix processors, a video processing unit, and various peripheral cores for DDR, SATA, PCIe, DMA, GPIOs and UART. The chip consumes ca. 3 watts at 648 MHz and 1.15V. In our board configuration the voltage can be scaled in three steps from 1.1 - 1.3V. The frequency is adjustable in four steps: 81 MHz, 162 MHz, 324 MHz and 648 Mhz. The RPX-SoC is supported by two operating systems: Linux 2.6.27 and LWOS. LWOS is a light-weight operating system written by Renesas for internal usage. Linux can only utilize the first processor cluster (4 cores) since cache coherency is not maintained between clusters. LWOS and its applications do not have this limitation and can utilize all 8 cores. The following section we propose compiler assisted power control for some user-space applications and introduce our OSCAR compiler tool-chain.

#### OSCAR - Task Schedule

The figures illustrate a static schedule generated by our OSCAR compiler for a heterogeneous SoC with processors and accelerators. The left figure visualizes task dependencies. The boxes represent macro-tasks (MT) which are coarse grained tasks with loops, function calls and basic blocks. The blue colored boxes can be mapped to processors; the green colored boxes can additionally be mapped to accelerators (ACC). The figure on the right shows the schedule for three processors CPU0, CPU1, CPU2 and an accelerator (ACCa). CPU2 offloads tasks to the accelerator and performs necessary data transfers. First, our compiler assigns the ready macro-task MT1 to CPU0. Then MT2 and MT3 are mapped to CPU0 and CPU1. After MT1 finishes, MT2 and MT3 become ready and so forth until all tasks have been executed. Occasionally, “green” accelerator tasks are mapped to processors if the accelerator is unavailable.

#### Linux user-space-governor interface is not able to exploit the hardware capabilities. Where do these overheads occur?

For user-space DVFS it is necessary to understand how the user-space-governor functions:

First, applications open a pseudo-file “scaling_setspeed” in the sysfs-file system. Secondly, they write a text string with the new frequency into the pseudo-file. Thirdly, they close the file.

Thus three system calls are required. However, this still does not explain the manifold overhead. Our analysis indicates that the sysfs-kernel layer which passes pseudo-file operations to the cpufreq-device driver is to blame. In Section VI-A we will present improved interfaces for user-space DVFS control.

#### Our novel contributions are:

- Case Study: Analysis of three methods for user-space DVFS
- Efficient clock- and power-gating for user-space applications via “autoidle”-threads and new system calls
- A power-adaptive in-kernel barrier for user-space applications
- Discussion of opportunities and challenges of user-space power control

Our contributions and insights apply to embedded systems as well as large data centers since both are power constrained and frequently utilize Linux. In the following section we introduce our experimental hardware setup.

#### III. Experimental Hardware Setup

For our experiments we have used the Renesas RPX-SoC [15, 16]. This 45nm research SoC - see Figure 2(a) - has eight SH4A processors, reconfigurable ALU arrays, two MX-2 matrix processors, a video processing unit, and various peripheral cores for DDR, SATA, PCIe, DMA, GPIOs and UART. The chip consumes ca. 3 watts at 648 MHz and 1.15V. In our board configuration the voltage can be scaled in three steps from 1.1 - 1.3V. The frequency is adjustable in four steps: 81 MHz, 162 MHz, 324 MHz and 648 Mhz. The RPX-SoC is supported by two operating systems: Linux 2.6.27 and LWOS. LWOS is a light-weight operating system written by Renesas for internal usage. Linux can only utilize the first processor cluster (4 cores) since cache coherency is not maintained between clusters. LWOS and its applications do not have this limitation and can utilize all 8 cores. The following section we propose compiler assisted power control for some user-space applications and introduce our OSCAR compiler tool-chain.

#### OSCAR - Task Schedule

The figures illustrate a static schedule generated by our OSCAR compiler for a heterogeneous SoC with processors and accelerators. The left figure visualizes task dependencies. The boxes represent macro-tasks (MT) which are coarse grained tasks with loops, function calls and basic blocks. The blue colored boxes can be mapped to processors; the green colored boxes can additionally be mapped to accelerators (ACC). The figure on the right shows the schedule for three processors CPU0, CPU1, CPU2 and an accelerator (ACCa). CPU2 offloads tasks to the accelerator and performs necessary data transfers. First, our compiler assigns the ready macro-task MT1 to CPU0. Then MT2 and MT3 are mapped to CPU0 and CPU1. After MT1 finishes, MT2 and MT3 become ready and so forth until all tasks have been executed. Occasionally, “green” accelerator tasks are mapped to processors if the accelerator is unavailable.

#### Linux user-space-governor interface is not able to exploit the hardware capabilities. Where do these overheads occur?

For user-space DVFS it is necessary to understand how the user-space-governor functions:

First, applications open a pseudo-file “scaling_setspeed” in the sysfs-file system. Secondly, they write a text string with the new frequency into the pseudo-file. Thirdly, they close the file.

Thus three system calls are required. However, this still does not explain the manifold overhead. Our analysis indicates that the sysfs-kernel layer which passes pseudo-file operations to the cpufreq-device driver is to blame. In Section VI-A we will present improved interfaces for user-space DVFS control.

#### Our novel contributions are:

- Case Study: Analysis of three methods for user-space DVFS
- Efficient clock- and power-gating for user-space applications via “autoidle”-threads and new system calls
- A power-adaptive in-kernel barrier for user-space applications
- Discussion of opportunities and challenges of user-space power control

Our contributions and insights apply to embedded systems as well as large data centers since both are power constrained and frequently utilize Linux. In the following section we introduce our experimental hardware setup.

#### III. Experimental Hardware Setup

For our experiments we have used the Renesas RPX-SoC [15, 16]. This 45nm research SoC - see Figure 2(a) - has eight SH4A processors, reconfigurable ALU arrays, two MX-2 matrix processors, a video processing unit, and various
software stack involves creating a new SoC-specification and re-compiling the code.

OSCAR is implemented as a source-to-source -compiler for C/FORTRAN. From input sources - OSCAR generates sources for each processor which is compiled by standard C/FORTRAN-compilers such as gcc for example. In the following section we introduce four HW/SW architectures and explain where our contributions fit in.

V. ARCHITECTURAL OVERVIEW

Developing and maintaining highly customized system software- and hardware can be time consuming and error prone. Thus many integrators of embedded- and server-systems try to minimize soft- and hardware changes. Figure 4 illustrates four architectures. The first two (a) and (b) require few or no changes to OS-kernels, whereas (c),(d) are more complex in terms of hardware- and software [33,34]. They are outside of the scope of this paper. In the following section we explain how applications can efficiently control DVFS based on the first two (a-b) architectures.

VI. CASE STUDY: USER-SPACE DVFS-CONTROL

The motivational case study in Section II revealed that user-space power control can be inefficient. In this section we will introduce two alternative methods of user-space DVFS control.

A. New system call for DVFS

To avoid the pseudo-file system overheads of the Linux user-space governor, we implemented a new system call that directly invokes the cpufreq-device driver. Our initial version resembled this code fragment:

```c
asmlinkage long sys_freq(int core, int freq) {
    struct cpufreq_policy policy;
    cpufreq_get_policy( &policy, core);
    policy.cpu = core;
    policy.governor->store_setspeed((&policy, freq);
}
```

The above code first fills the cpufreq_policy data structure with the core number and calls the governor’s store_setspeed function. Our new system call avoids textual parameter parsing, the pseudo-file system layer and reduces the number of systems calls - 1 instead of 3.

B. User-space device driver

After reducing the overhead of the kernel system call we were asking ourselves how we could further minimize overheads. On our hardware platform frequency- and voltage-registers are memory-mapped registers. Via remapping
memory-pages it is possible to access these registers from user-space.

On Linux memory mapping can be performed by custom device drivers or more generally by using the /dev/mem device driver and the mmap-system call. The following code fragment illustrates the procedures:

```c
fd = open("/dev/mem", O_RDWR|O_SYNC);
...
mapped_addr = (unsigned int) mmap(NULL,
    num_of_map, {PROT_READ | PROT_WRITE},
    MAP_SHARED, fd, CnIFC_ADRS(0));
...
```

\( CnIFC_ADRS(0) \) stands for the frequency control register address of core 0 on RPX. The frequency registers of the remaining cores follow on the subsequent memory pages. Care must be taken that these mappings are not cached. On Linux the /dev/mem device must be opened with the O_SYNC flag set. In our case even this did not work till we patched the kernel /dev/mem-device driver.

After mapping the necessary registers changing frequencies becomes a memory store operation:

```c
*(unsigned volatile int*) freq_ctrl_addr = ifc;
```

The ifc value is a platform specific and is used to configure the on-chip frequency divider. Changing the voltage is done in a similar fashion.

Once we could remap frequency- and voltage-registers successfully into user-space, we ported the kernel device driver to user-space and tested it successfully. In the following section we compare the performance of our two new power control interfaces and original one.

C. User-Space DVFS-Performance

Figure 5 shows that both our DVFS methods have a much lower latency than the original Linux DVFS interface via the user-space governor. Our user-space device driver performs...
best. Our new system call takes longer than can be explained by system call overhead which accounts only for 3\(\mu\)s. The additional 46\(\mu\)s are spend in kernel for "extra" activities.

Closer investigation within the Linux kernel reveals that the cpufreq driver calls a cpufreq_notify_transition function that is invoked before and after every frequency change. The function notifies kernel sub-systems about processor frequency changes. The call chain must be synchronized across processors and may therefore be costly. The adjust_jiffies function - for example - is called before and after frequency changes to adjust time keeping in the Linux kernel. Besides this function there are no other sub modules that need notification on RPX.

However, for more complex SoCs such as those mentioned in the introduction the situation is often more complex. Frequency- and voltage changes may affect multiple on-chip components and kernel drivers. Through the notification call chain otherwise independent Linux device drivers can act upon changes in shared infrastructure - such as clocks or voltage regulators. The cost of this flexiblity is however, that changes must be synchronized across multiple processors. Thus power saving capabilities of modern chips are potentially diminished for "short" time periods. In the following section we try to make clock- and power gating accessible to user-space applications.

VII. CASE STUDY: CLOCK- AND POWER-GATING

In addition to DVFS we wanted to make clock- and power-gating accessible to our OSCAR-compiled applications. On our prototype platform RPX clock- and power-gating can be initiated by issuing the privileged sleep instruction with different flags. Unfortunately, the instruction is only accessible if the processor is in privileged mode. This is especially annoying since clock gating requires just a few nanoseconds but system calls at least 3\(\mu\)s. Executing applications in privileged mode would allow instructions such as sleep to be accessible by applications.

The Linux-kernel supports clock- and power gating indirectly through the idle threads. Idle threads are invoked whenever (per-processor) scheduler’s run-queues are empty. Eventually, idle threads will cause processors to transition to certain sleep-modes which deploy clock- or power-gating.

For user-space applications we have implemented a new pair of system calls which (1) invoke the kernel idle functionality directly, or (2) wake idling threads up. The following code fragment was taken from the Linux idle-function and integrated into our new idle system call:

```
if (cmd == SYSFREQ_IDLE) {
  ...
  tick_nohz_stop_sched_tick(1);
  while (!need_resched())
    idle();
  tick_nohz_restart_sched_tick();
  ...
}
```

(a) Conservative governor
The Linux conservative-governor is slower in its response to load changes than the two previous approaches. Furthermore, it oscillates if the system is unloaded.

(b) On-demand governor
The Linux on-demand-governor adapts DVFS to system load. Like the auto-idle enabled Linux-kernel it quickly ramps up DVFS but does not immediately reduce power if load drops.

(c) Our Auto-Idle
Our new auto-idle enabled Linux-kernel exploits the low-latency clock- and power-gating capabilities of the RPX-SoC. When idle-threads are activated then we immediately power-gate processors. If an application thread is activated we ramp-up DVFS to pre-specified values. For latency sensitive applications that perform their activities in bursts - for example in sensor networks - this behavior may be better suited than the standard Linux governors.

Fig. 6. Operating-system Power Control - Auto-Idle, On-Demand, Conservative - The three graphs show power [W] over time as the system transitions from unloaded to loaded and back again to unloaded. We conducted the measurements using the Renesas RPX-prototype board which supports inductive power measurements of the SoC.
The code disables the periodic scheduler tick to avoid unnecessary wake-ups. As long as the scheduler does not require re-scheduling the idle task will invoke the platform specific idle function. The idle function calls low-level device drivers for clock- and power-gating.

Our new idle system call allows OSCAR compiler-generated power control code to directly call idle for clock- and power-gating - while keeping caches hot.

For applications that have not been compiled with OSCAR, we have developed an experimental autoidle- function that can be enabled at run-time. If the kernel autoidle-flag is set, a processor will immediately switch to lower frequencies and/or clock gate the processor - when the kernel idle task is scheduled. If the kernel idle task relinquishes control, then the previous frequency will be restored immediately. The initial base frequency is fixed but configurable.

On the processor scope auto-idle has a binary "on/off" pattern, whereas the Linux on-demand- or conservative-governors need more time to track application activity - see Figure 6.

We think that our experimental autoidle-mode may be useful to save power in event-triggered applications. In the following section we introduce a power-adaptive kernel interface for barrier synchronization.

VIII. ADAPTIVE AND POWER AWARE KERNEL BARRIER

We have implemented a barrier-system call within the Linux-kernel similar to the gcc-OpenMP barrier\(^1\):

Threads that arrive at our kernel barrier spin for some time before blocking in idle. As described earlier, in idle-mode processors are either clock- or power gated. The last thread to arrive at the barrier will wake-up all waiting threads and reset the barrier.

Our adaptive power optimizations adaptively set the frequencies of the first threads to arrive at the barrier to reduced values while they spin. Reducing the frequency also helps other threads on RPX for example - since the voltage controller is shared - thus voltages can only be dropped if all threads fall below certain frequency levels.

If all but one threads have arrived at the barrier, then we boost the frequency of the last thread in order to finish the barrier quicker. This behavior may be beneficial if static power is high and excessive wakeups burn power.

Since the barrier is implemented within the kernel it is possible for processes to synchronize and not only for threads. Ideally, OSCAR applications will not need the power-adaptive features of our kernel barrier - since the static task schedule will automatically issue near-optimal power control commands. However, on some hardware architectures with complex memory architectures and interference from other unrelated tasks it may be possible that the static schedule is disturbed. Our adaptive barrier can help to dynamically fix such situations until the threads synchronize again. In the next section we discuss how we try to keep interference from unrelated Linux applications to a minimum.

IX. TASK-PROCESSOR BINDING

OSCAR applications assume processors to be under their full control in respect to scheduling and power control. On LWOS - see Section III - only one application is running at the time and this assumption holds. On Linux - however - the situation may be very different. It is up to the Linux scheduler to decide when and what tasks to execute and migrate among available processors.

Therefore, we have devised a kernel modification which keeps all Linux background tasks on processor zero. Thus the remaining processors are "free" for OSCAR-applications.

In Linux each process has a task_struct. We extended this task_struct with an OSCAR-flag and patched all places where the Linux scheduler may migrate threads. Thus at run-time we can ensure that Linux application will never be spawned or migrated to processors under OSCARs control.

The following source code fragment shows how our new system call binds OSCAR processes via our SF_BIND command - before executing the sched_setaffinity call.

```c
int rv = sched_setaffinity(
    getpid(), sizeof(cpu_set_t), &set);
```

To test our approach we have written a small test application that binds to processors other than processor zero and calls our new idle system call - see Section VII. Thanks to our kernel modifications we were able to stay in idle for up to 30 seconds without any interruptions. On processor zero where all background tasks and daemons are located this would be impossible. On processors 1-3 our modified RPX- Linux faces few disturbances and therefore provides a suitable environment for statically scheduled OSCAR applications. In the next section we introduce our new kernel system calls for taking processors completely offline.

X. PROCESSOR HOT-PLUGGING FROM USER-SPACE

The Linux kernel supports processor hot-plugging from user-space via an "online" pseudo-file. Applications can open this file and read- and write to it similar to the default DVFS user-space pseudo file mentioned earlier. The standard kernel includes many unnecessary wait-statements that we could remove safely for the RPX-SoC. We were able to reduce the transition times from 2 seconds down to a few milliseconds.

On RPX Linux however - the processor hot-plug device driver is not yet able to exploit power- or clock gating if processors

\(^1\)See libgomp source from http://gcc.gnu.org/ for the barrier implementation. Currently two targets are supported Linux and POSIX. The Linux target uses the FUTEX-system call for fast synchronization.
SoCs have to be carefully analyzed and possibly changes discussed in the previous sections. Currently, existing control ranging from hardware issues to security which we deeply that we have been faced with in the previous sections - more section we reflect upon some user-space power control issues still missing on those operating systems. In the following middle-ware that automatically switches among governors is always require the Android applications. Many applications may be suitable for execution OSCAR-applications. To exploit low-latency compiler-controlled power control in parallel applications. The next section we discuss security issues of user-space power control.

XI. Security

The Linux kernel requires root status to let user-space applications write to pseudo-files that provide interfaces to device drivers. Our system call has currently no security checks which is fine for prototyping, testing and closed embedded systems. In the future we may include checks based on group permissions. OSCAR compiled applications could - for example - belong to an OSCAR group to automatically gain access to user-space power control. On the hardware side security is rather coarse grained. Privileged instructions for clock-gating - for example - can usually not be made accessible to selected applications but only to the kernel. For user-space device drivers it will be necessary to define fine-grained security models in order to provide safe access to hardware settings. In the next section we discuss the challenge of synchronizing state between the kernel- and user-space device drivers.

XII. Synchronizing State Between Kernels and User-space device Drivers

All kernel based interfaces for power control drivers - such as our new system call for DVFS maintain a correct view of hardware states within the kernel. User-space device drivers - however - may cause inconsistencies between user-space- and kernel-device drivers. During testing we avoided inconsistencies by configuring the user-space governor of Linux. The user-space governor does not actively change frequencies- or voltages. Furthermore, our user-space device driver restores frequencies- and voltages - so before- and after executing OSCAR-applications.

For smart phones- and tablet PC- operating systems such as Android this approach may be to static. It may be necessary to switch between different governors depending on active applications. Many applications may be suitable for execution with the ondemand, conservative or interactive-governors that Android and Linux provide. OSCAR applications - however - always require the user-space governor. A power management middle-ware that automatically switches among governors is still missing on those operating systems. In the following section we reflect upon some user-space power control issues - that we have been faced with in the previous sections - more deeply.

XIII. Experiences from the User-space Power Control Front-line

There are several challenges surrounding user-space power control ranging from hardware issues to security which we have discussed in the previous sections. Currently, existing SoCs have to be carefully analyzed and possibly changes must be made to kernels in order to work around hardware limitations. Unfortunately, user-space power control is not even an after thought in architecture and operating systems.

RPX - our prototype processor - allowed us to re-map frequency- and voltage-registers into user-space. Other architectures may require privileged instructions to set register values. On RPX - for example - clock gating requires privileged instructions. To fully exploit clock gating on RPX we would need to run our applications in privileged-mode along side with the kernel.

Another, easily overseen aspect is if processors can configure DVFS only for themselves, or also for other processors^2. On some architectures certain processor specific registers can only be changed reliably if instructions execute on the target processors. For RPX under LWOS - for example - it is necessary to wait 1\(\mu\)s after writing to a frequency register of another processor. On Linux - however - the RPX processor is configured differently and only local processors can reliably change their own frequencies. The low-level RPX Linux device driver migrates itself to the target processor if necessary. However, task migration can be very costly - on RPX >100\(\mu\)s for example. The Linux eSPARC DVFS device driver - in comparison - must execute a minimum number of NOPs on the target processor after frequency changes. This can only be guaranteed if interrupts are disabled - something which is normally not possible from user-space. The next section concludes our paper.

XIV. Conclusion

In this paper we have proposed to use auto-parallelizing compilers to generate task- and power-control schedules. The generated schedules can be configured for very high time resolutions down to nanoseconds. Upcoming- and existing research processors already offer low latency DVFS, clock- and power-gating. However, current applications and operating systems cannot exploit these capabilities fully. Our DVFS-case study showed that existing applications can be reduced to negligible amounts - if hardware and operating systems are flexible enough. Furthermore, operating systems and hardware must ensure that statically scheduled applications are not disturbed by unrelated applications, or kernel-threads that can be migrated, postponed or deactivated. In this paper we have made contributions to this area. We want to raise awareness among processor architects and hope they will enable us to exploit low-latency compiler-controlled power control in parallel applications.

REFERENCES


^2For OSCAR compiled applications it is generally sufficient if underlying drivers respect specified hardware behaviors. The OSCAR power control functions are specified in the official OSCAR-API which can be downloaded from our website.


