Extended Abstract: Decelerating Suspend and Resume in Operating Systems

(A workshop paper using the same title will appear at HotMobile'17)

Shuang Zhai, Liwei Guo, Xiangyu Li, and Felix Xiaozhu Lin
Purdue ECE

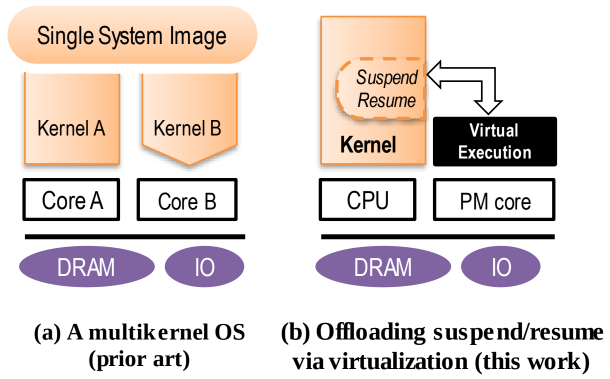

Figure 1: A comparison of two alternative OS structures for harnessing heterogeneous, incoherent cores

1. MOTIVATIONS

Today's mobile and wearable computers see a large number of intermittent, short-lived tasks such as push notification, email sync, and "always-on" UI. The short-lived tasks, as shown in recent work, drain a large portion of battery, e.g. 29% on smartphones [1]. To execute a short-lived task, the whole system exits from a deep-sleep state, runs user code, and re-enters the deep-sleep state. The power state transitions are performed by suspend/resume, a core power management (PM) function in OS.

Ironically, despite critical to system energy efficiency, the suspend/resume procedure itself is expensive. In running a short-lived task, suspend/resume often dominates the energy consumption, sometimes incurring 10× higher energy than the task's user code execution [2] .

2. FINDINGS

To pinpoint the bottlenecks, we profile the Linux suspend/resume on a variety of mobile SoCs. Our findings indicate that suspend/resume is fundamentally slowed down by IO: the CPU spends much of its time waiting (being not only idle but also busy) for hundreds of IO devices to complete their power state transitions. Unfortunately, shortening such transitions is difficult: the transition delay is often bound by physical factors or interface standards; the slow IO devices are diverse and platform-dependent; transitions of multiple IO devices can hardly be parallelized.

The profiling suggests that suspend/resume poorly fits today's high-frequency, complex processors that are intended for rich mobile applications. Instead, the OS suspend/resume should be offloaded to miniature, low-power processors which we dubbed "PM cores". This goal, however, is challenged by the high complexity of the OS suspend/resume code, the PM core's instruction set architecture (ISA), and the demand for remaining compatible with commodity OSes.

3. SOLUTIONS AND CONTRIBUTIONS

We present a novel virtual executor, as shown in Figure 1: running on a PM core, it completely takes charge of suspend/resume by directly executing the main CPU's unmodified kernel binary. This greatly reduces energy cost by keeping the main CPU powered off for OS suspend/resume.

To fit a weak PM core, our virtual executor is purely software-based, an approach previously believed too expensive. To make it practical, we *specialize* the virtual executor for the kernel suspend/resume path, rather than supporting generic kernel code. The virtual executor consists of two key components: a translator that dynamically translates the main CPU's kernel binary and executes it; a function pruner that replaces generic, expensive OS functions with specialized, simplified versions.

Of the two components, the binary translator's overhead is critical to the entire system's efficiency – its software-based cross-ISA translation, according to conventional wisdom, incurs prohibitive overhead. To tackle this problem, our insight is to exploit the *similarity* between heterogeneous ISAs, which are commonly seen among co-located modern processors, as exemplified by the ARM A-series and M-series cores. Through a set of key optimizations, we reduce the translation overhead by 5× as compared to the state of the art. Based on the measured power and overhead, we estimate to reduce the total energy cost of suspend/resume by up to 70% and extend the battery life by 30%.

We have made the following contributions:

- We quantify the Linux suspend/resume procedure on a variety of SoCs and examine the causes of high energy consumption and long delay;
- We offload the OS suspend/resume to an extremely low-power core, which runs a novel virtual executor for executing the unmodified kernel binary with low overhead;
- We describe a first-of-its-kind working prototype of the virtual executor; at the time of writing, the prototype contains 50.5K SLoC, of which 4.5K are new[1]. The preliminary results show great promise.

4. REFERENCES

[1] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015.

[2] M. Lentz, J. Litton, and B. Bhattacharjee. Drowsy power management. In *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015.

[1]Generated using David A. Wheeler's 'SLOCCount'.