Software Engagement with Sleeping CPUs

Qi Zhu, Meng Zhu, Bo Wu, Xipeng Shen, Kai Shen, and Zhiying Wang
North Carolina State University, USA
University of Rochester, USA
Colorado School of Mines, USA
National University of Defense Technology, China

Energy Proportionality and CPU Sleeps

- Energy proportionality [Barroso and Hölzle 2007]
  - Energy is consumed only when performing work.
- CPU hardware sleeps, idle states, or C states
  - An idle CPU can save power by halting cycles, shutting off clocks, flushing and powering down caches, and even removing core voltage.
  - On a dual-socket (24-cpu) Intel Haswell machine, active idle of all CPUs consume 91W (processor+DRAM) while C6 sleeps consume just 14W.
- The energy benefit is recognized [Le Sueur and Heiser 2011] and CPU sleeps are routinely utilized in existing systems
  - But profound system implications in today's context (emerging devices and workloads) require strong, principled software management.

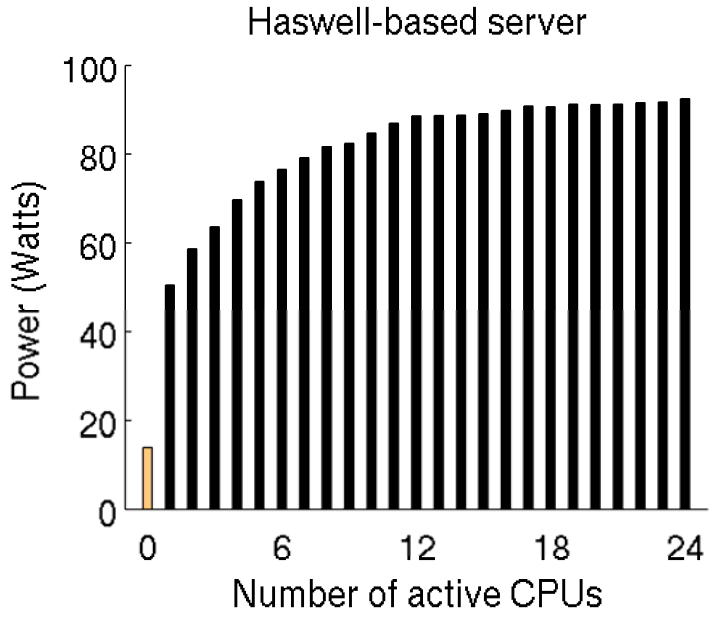
Sleep Exits Are Not Instantaneous

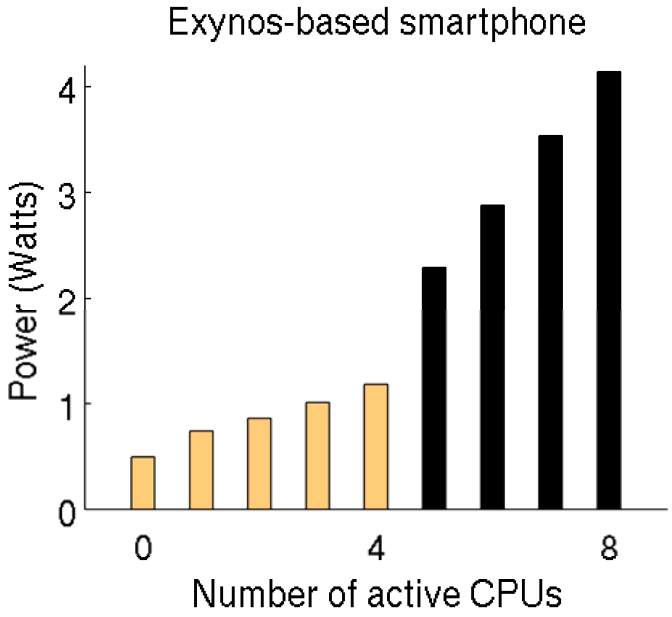
- Deep CPU sleeps incur delays to reactivate
  - Activating voltage/clocks, resuming cycles, loading flushed cachelines, …
  - 100usecs or more on modern multicore processors
- May add substantial (possibly multi-fold) delays to
  - operations on emerging fast devices like SSDs and integrated GPUs
  - fine-grained network services (e.g., in-memory hashtable) in a data center

⇨ On-demand (interrupt-driven) resource activation is inadequate for high responsiveness.

Energy Disproportionality on Multicores

- Due to multicore resource sharing, power is not proportional to the number of active (non-sleeping) CPUs.





⇨ Energy efficiency motivates new resource scheduling to shape desirable sleep patterns.

Anticipatory CPU Wakeups

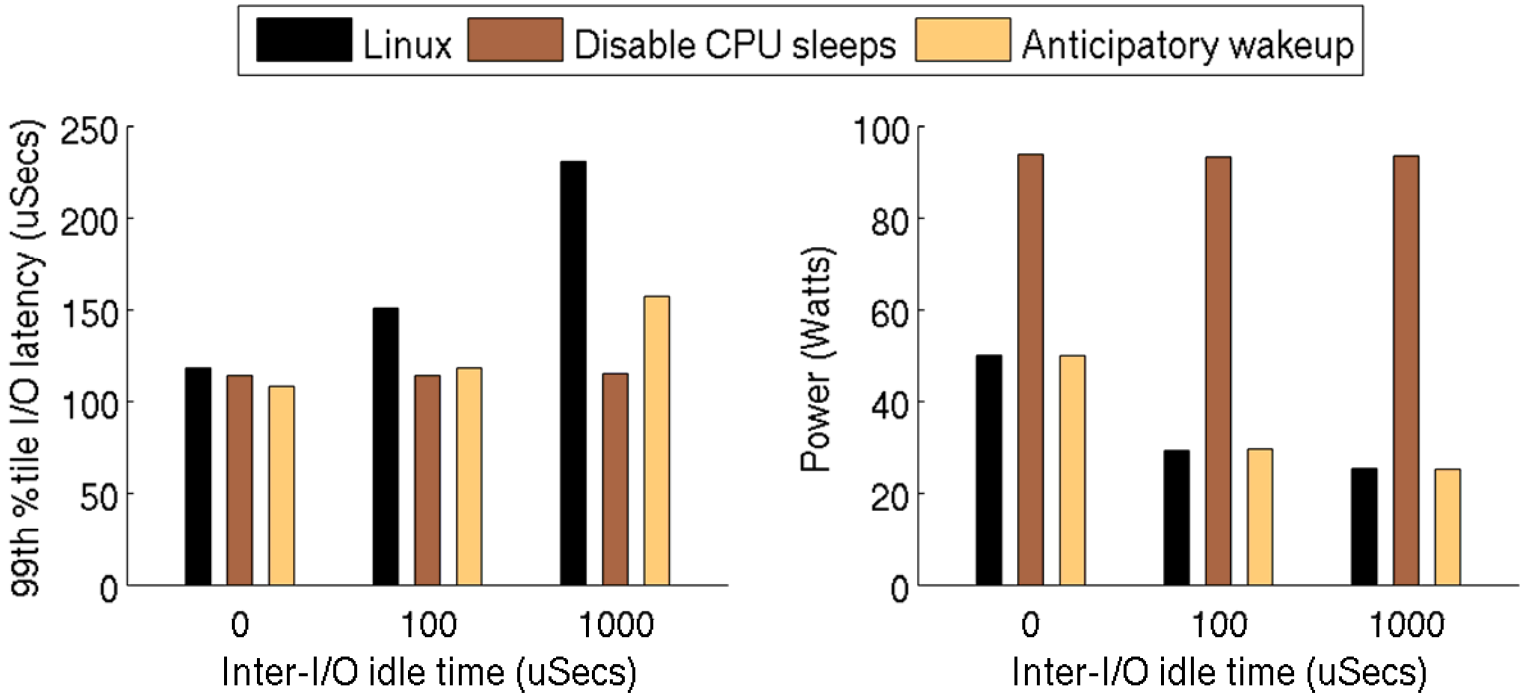
- For high responsiveness, a sleeping CPU should wake up in advance so that it is immediately ready for work when needed.
- Main challenge: anticipate the timing of future work.
  - When blocked on SSD I/O, future work is anticipated at the I/O completion time (modeled linearly on the I/O size) ⇨ Anticipation in system
  - Computation time on Turing-complete GPUs is hard to model, but many apps (iterative solvers, ML refinements) iterate over similar kernels many times and allow history-based prediction ⇨ Anticipation by application
  - On a network server, anticipation of future client requests may require client notification in advance ⇨ Anticipation over network
- Related to anticipatory I/O [Iyer and Druschel 2001]
  - Aiming for a binary decision, rather than anticipating the future event timing

A Simple Prototype

- We augment the block layer
  - predict SSD I/O time using a linear model (on I/O size)
  - request anticipatory CPU wakeup for synchronous I/O operation
  - fully transparent to applications
- Activate all CPUs necessary for work
  - CPU that'll run the blocked application process
  - CPU that'll handle the I/O interrupt

Preliminary Evaluation

- 24-CPU Intel Haswell machine, Samsung 850 PRO SSD



Energy-Conserving Sleep Shaping

- Saving most power on multicore by simultaneous CPU sleeps of
  - an entire multicore socket
  - the high-power cluster on a heterogamous smartphone SOC
- Motivate energy-conserving sleep shaping
  - Non-work-conserving CPU scheduling
- Exploit quality-of-service slacks for opportunities to delay work
  - Not all work in a smartphone system critically affects user interaction.
  - A server system may only be concerned about responses beyond a certain threshold.

Server Staged Bursts

- A server machine alternates between two phases–
  - a staging phase that buffers requests without running them, and
  - a burst phase that runs buffered requests in high parallelism.
- The staging proxy best runs on a low-power companion processor, or on a few designated proxies in a data center.
- A simple case evaluation:
  - Apache Solr search engine, searching Wikipedia pages, 100 reqs/sec workload
  - reduce power from 68 Watts to 53 Watts, while keeping peak responses below 500 msecs

Energy-Conserving Sleep Shaping

- Main challenge:
  - When can work be delayed or slowed without hurting quality-of-service?
- In a smartphone system
  - Quality-of-service is defined by responsiveness to a user interaction (from touch screen input to screen rendering of results).
  - Identify causal dependencies and critical path in a user interaction through sync/communication events (pipes, sockets, signals, Android binders, …).
  - During an I/O operation on the critical path, concurrent CPU work may be delayed or slowed without hurting user response.

Summary

- Energy proportionality has brought us aggressive CPU sleeps, but
  - sleep exit time is causing significant latency increase on emerging fast devices (SSDs, integrated accelerators) and fine-grained network services
  - on multicores, power is disproportionate to the number of active (non-sleeping) CPUs
- Advocate new CPU resource management approaches
  - anticipatory wakeups to minimize latency impact
  - non-work-conserving sleep shaping to maximize energy efficiency
- Concept (particularly anticipatory wakeups) is applicable to other dynamic-power resources (memory, storage, …)