Energy Discounted Computing on Multicore Smartphones

Meng Zhu and Kai Shen

UNIVERSITY of ROCHESTER

Power-Ful & Hungry Smartphones

Multi-core Popularity and Low Battery Anxiety

Tri-cluster
Deca-core

Multicore Energy Disproportionality

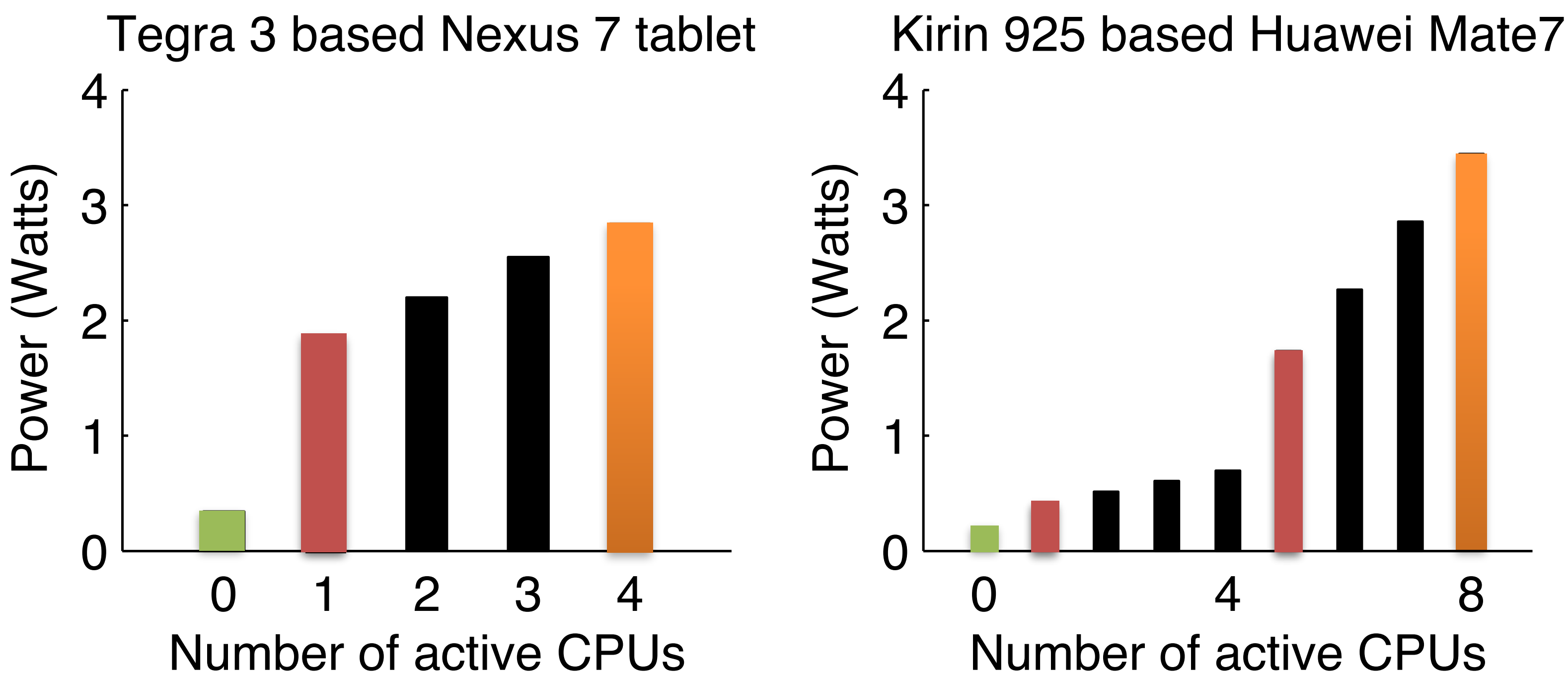
- Multicore processors are not energy proportional, despite all the efforts
- Aggressive hardware sharing
  - Shared clocking circuitry forces cores to operate at the same frequency
  - Power-gating enables low power idle state, but deep idle states can only be entered during simultaneous sleep

Multicore Energy Disproportionality

State	Name	Power	Description
C0	Wait for interrupt (WFI)	403 mW	Individual core clock gated.
C1	Individual powerdown	365 mW	- Individual core power gated. - L1 cache content lost
C2	Cluster powerdown	214 mW	- Enter during simultaneous sleeps - All state (e.g. L2$) lost

Device: Kirin 925 SoC on Huawei Mate7 Smartphone, cluster has four cores

Multicore Energy Disproportionality



- Very energy efficient during peak utilization
- Consume minimum power when all cores are quiescent
- Inefficient when only one core is utilized

Mobile Apps Lack Parallelism

- Typical smartphone applications are built on event-driven, UI-centric framework and serve a single user
- Do not have sufficient parallelism to utilize multiple CPU cores simultaneously
- On a quad-core system, of all the non-idle time:
  - All four cores are utilized: less than 1%
  - Only one core is utilized: 68%
  - Test conducted with a variety of popular mobile applications (Gao et al. ISPASS'15)

Opportunity

- Hardware energy disproportionality and Lack of thread level parallelism —> *Computing resources at additional cores are available at a deep energy discount*

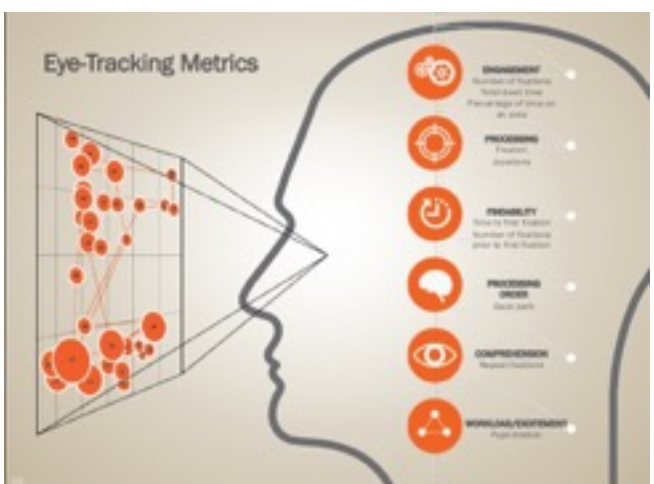
- Utilize these resources to run best-effort tasks: useful tasks on a smartphone but do not involve direct user interaction (thus its time of execution is flexible)

Best-effort Tasks

Upload and download

System maintenance work

Background sensing



Proactive Tasks

Energy Discounted Computing

- Bundling tasks to save energy on smartphones is not new
  - Lane et al. [Sensys 2013]: Piggybacking sensing activities
  - Nikzad et al. [ICSE 2014]: Annotation language for developers to delay certain work

- Our contribution: *Maximum energy discount is only realized when the co-run best-effort task execution does not elevate the overall system power state.*

Best-effort task execution must NOT

- Disrupt the multicore CPU idle state [“**C**” state]
  - Follow the step of interactive task execution
  - Non-work-conserving-scheduling
- Increase the core frequency [“**P**” state]
  - Invisible to the system frequency adjustment
  - Do not affect frequency scaling for interactive tasks
- Affect the smartphone’s suspension period [“**S**” state]

Hide behind the CPU power profile of interactive tasks

Implementation (Huawei Mate7, Android 4.4, Linux 3.10.30)

- Best-effort tasks are put into a control group
- Idle state preservation:
  - Each core maintains a status bit: BUSY, IDLE, BEST-EFFORT
  - Regular tasks have absolute priority over best-effort ones
  - If a best-effort task is picked, check sibling cores to see if anyone is BUSY. If no one is BUSY, enter idle state directly (non-work-conserving scheduling)

Implementation

- CPU frequency preservation:
  - DVFS is controlled by *cpufreq* governor, adjust frequency based on load
  - Best-effort tasks are ignored during load calculation
  - Performance of regular tasks are not affected
- Suspension state preservation:
  - Best-effort tasks are not allowed to hold wakelocks

Contention Mitigation

- Co-run applications leads to contention on multicore resources, cause performance degradation to interactive applications

- Scheduler priority modification does not remove contention on shared resources, i.e. cache and memory bandwidth

- Monitoring last-level-cache miss rate using performance counters and throttle best-effort tasks accordingly

Implementation

- Contention mitigation:
  - Monitor last-level-cache miss rate
  - ARMV7_A15_PERFCTR_L2_CACHE_REFILL_READ/WRITE
  - Sample every 20 ms
  - Stop scheduling best-effort tasks when the miss rate reaches certain threshold

- Overhead: less than 1% for all of our benchmarks

Evaluation: Setup

- Device: Huawei Mate7 (late 2014)
  - 1.8 GHz ARM Cortex-A15 Quad Core
  - 2MB L2 cache, 2GB RAM
  - Power measurement using Monsoon power meter with smartphone battery detached

Evaluation: Benchmark

- Interactive application:
  - Bbench: load locally cached websites
  - Angry bird: casual game

- Best-effort tasks: Spin, Compression, Encryption, AppOpt, FaceAnalysis

Evaluation: Test flow

- Use automate UI testing tools (RERAN[1]) to minimize variations

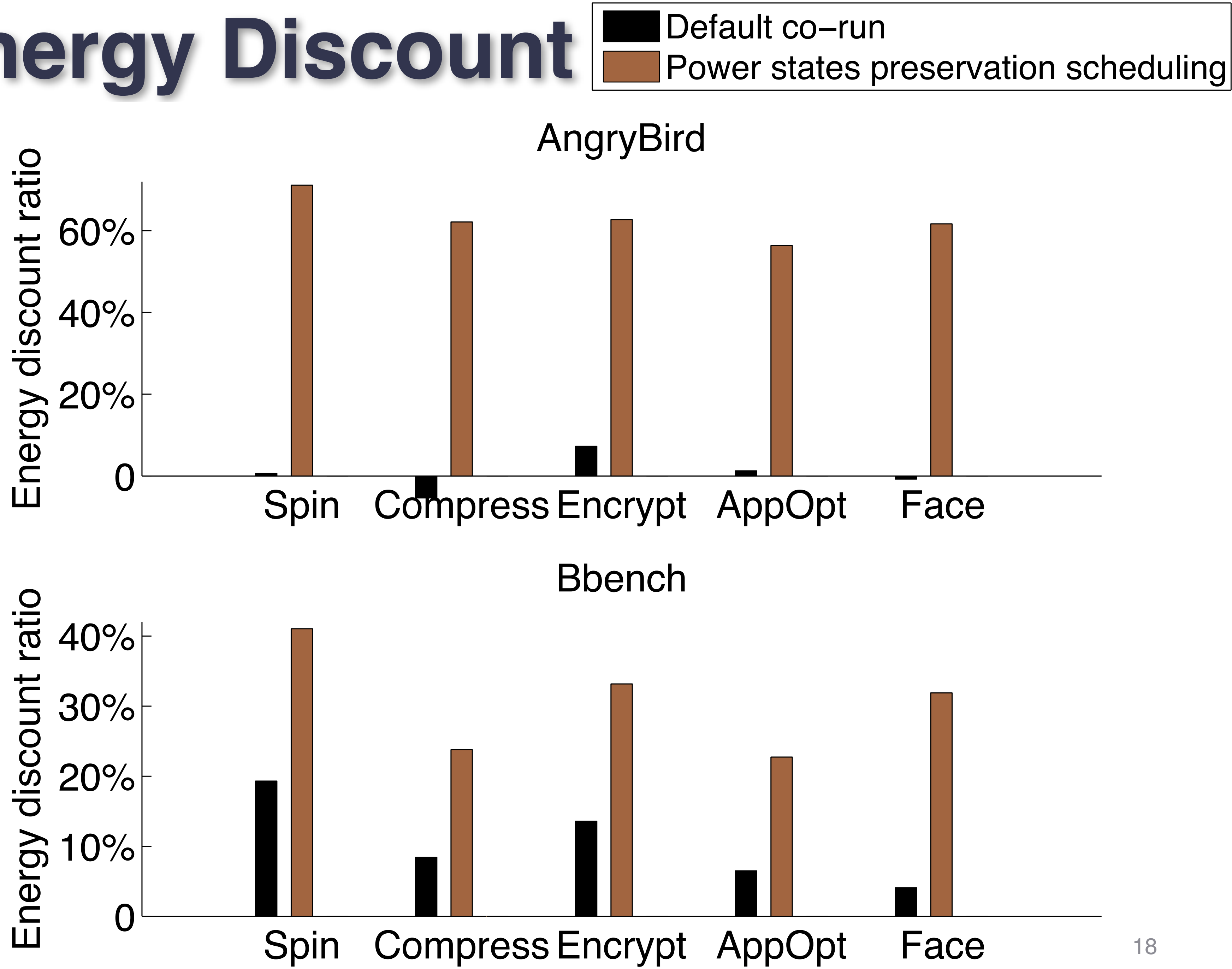
- Launch two applications roughly at the same time

- Configure the workload such that application executions mostly overlap

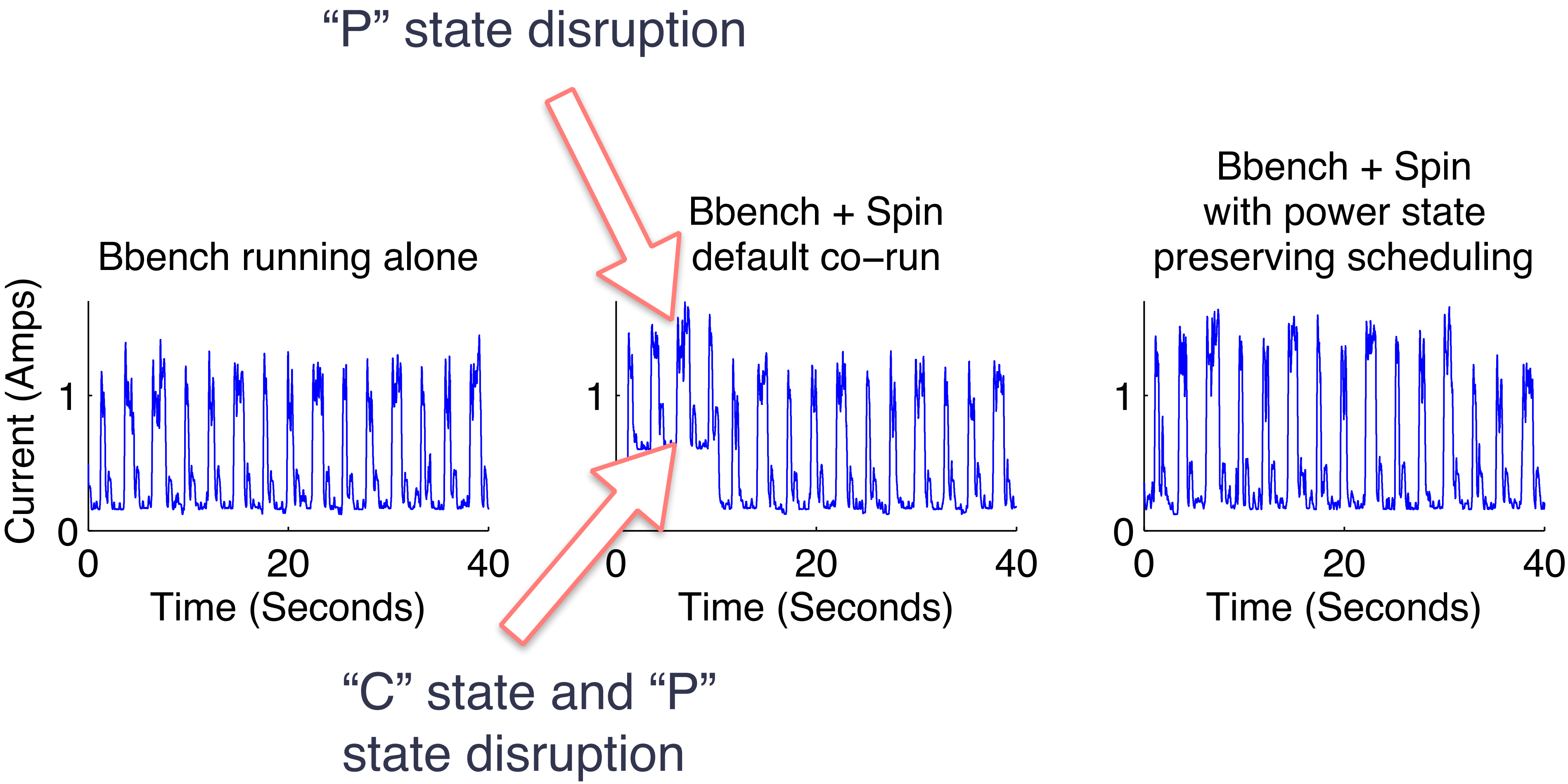
Discount $\sigma = 1 - \frac{E_{\text{co-run}} - E_{\text{interactive\_alone}}}{E_{\text{best-effort\_alone}}}$

[1] Gomez et al. ICSE'13

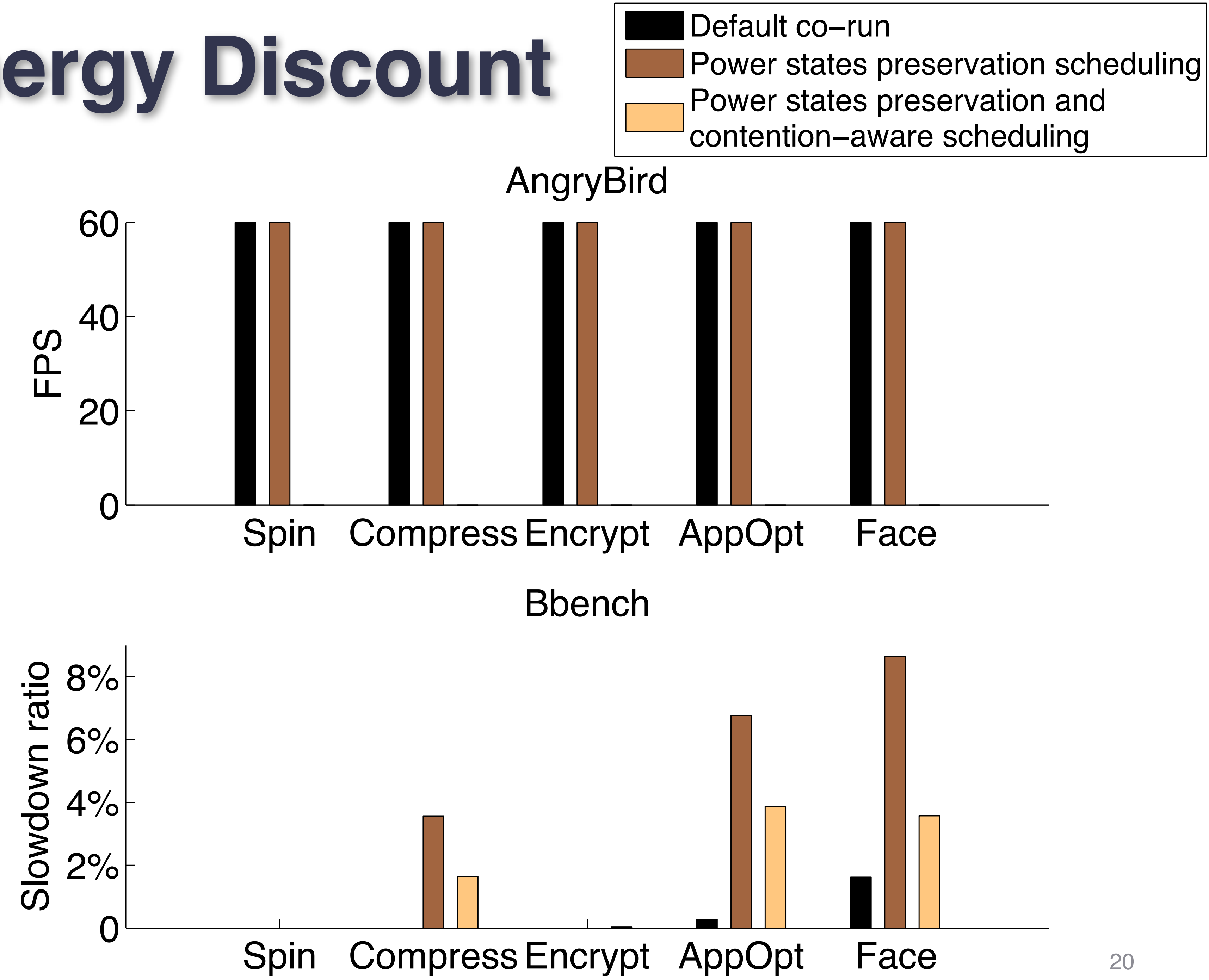
Energy Discount



Current Wave (Bbench + Spin)



Energy Discount



Abundance of Discounted CPU Cycles

Category	Abundance	Frames of face analyzed	Minutes of video encrypted
Web browsing	1.63	30	21
Video streaming	2.41	4	3
Gaming	1.61	21	15
Navigation	2.42	13	9
Messaging	2.88	3	2
Social network	1.88	12	9

* Abundance of discounted CPU cycles is the ratio of energy discounted CPU cycles to the active CPU cycles used by the corresponding interactive application.

Summary

- Energy disproportionality of multicore CPUs and lack of parallelism of smartphone applications provide abundant opportunities to run useful best-effort tasks at deep energy discount

- Maximum energy discount can only be realized when overall system power states are preserved

- Contention-aware scheduling based on monitoring hardware performance counters is effective in mitigating interactivity slowdown

- Experiments show significant energy savings (up to 63%) and little performance impact (less than 4% in the worst case) to the smartphone interactivity

Thank you
Questions?

Energy Discounted Computing on Multicore Smartphones

Meng Zhu and Kai Shen

UNIVERSITY of ROCHESTER