IO bound or CPU bound?

2014.10.31.

Dongjun Shin
Samsung Electronics

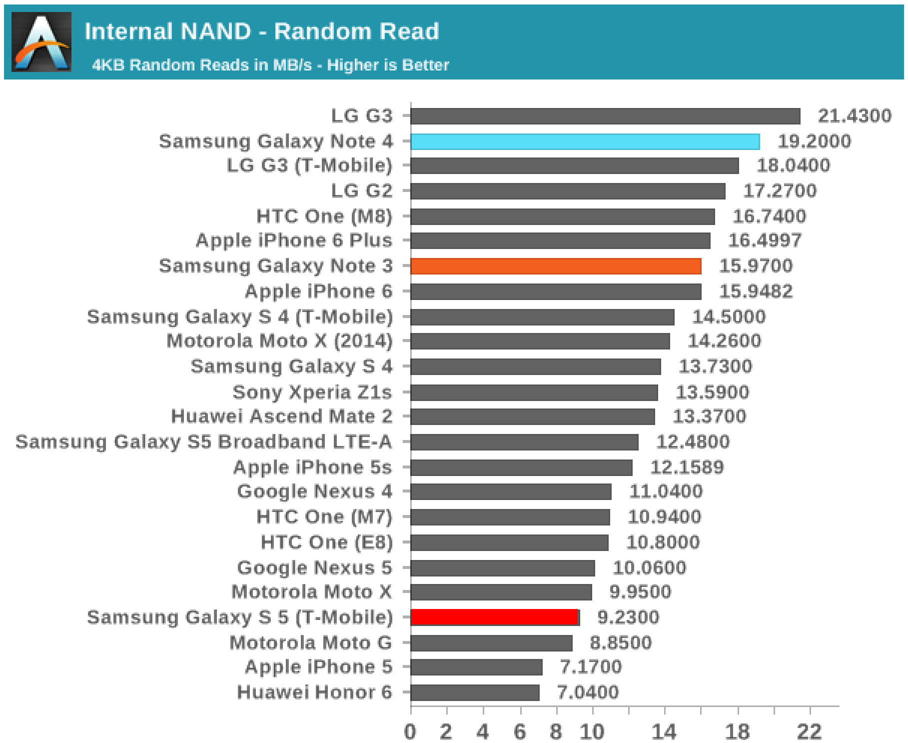
Contents

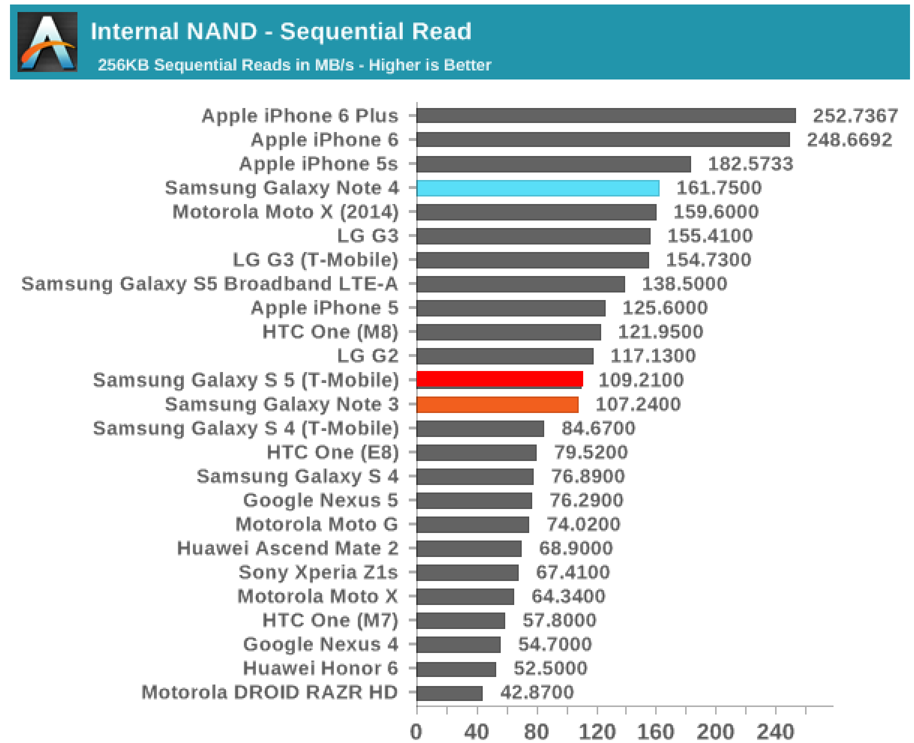
Background – Definition

- **CPU bound**
  - A computer is CPU-bound (or compute-bound) when the time for it to complete a task is determined principally by the speed of the central processor: processor utilization is high, perhaps at 100% usage for many seconds or minutes (wikipedia)
- **I/O bound**
  - I/O bound refers to a condition in which the time it takes to complete a computation is determined principally by the period spent waiting for input/output operations to be completed (wikipedia)
- **In reality**
  - I have a lagging application. Who is to be blamed?
  - I have a lagging application **and it seems that there are lots of I/O. It must be I/O bound.**
  - I have a lagging application and it seems that there are lots of I/O. **I can't believe it because it's running on ultra fast SSD!**

Background – Mobile Benchmark

- **AnandTech: Samsung Galaxy S5 vs. Galaxy Note 4**
  - What's the cause of read I/O regression?





	AP	eMMC
Galaxy S5	S801 2.5GHz x 4	5.0
Galaxy Note4	S805 2.7GHz x 4	5.0

http://www.anandtech.com/show/7903/samsung-galaxy-s-5-review
http://www.anandtech.com/show/8613/the-samsung-galaxy-note-4-review

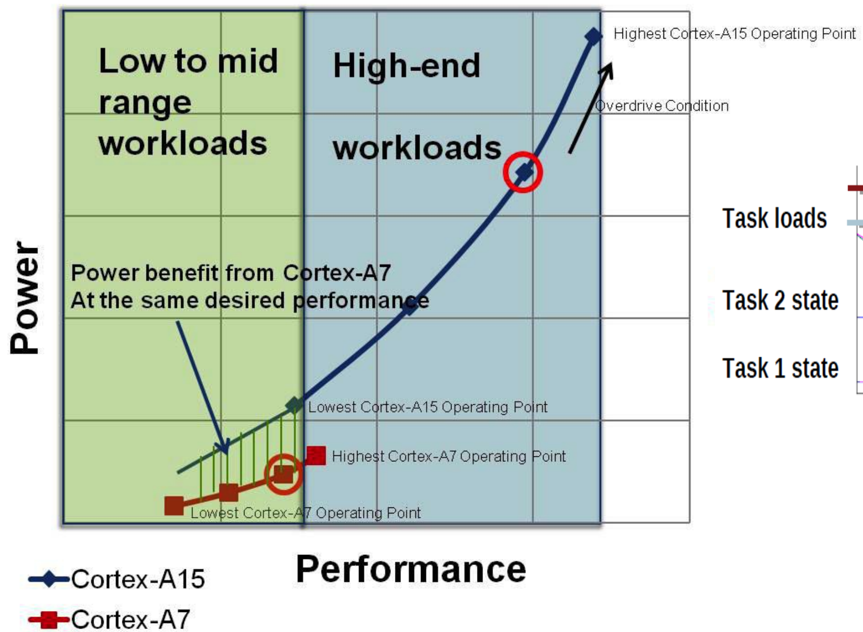
Understanding CPU Behavior (1/2)

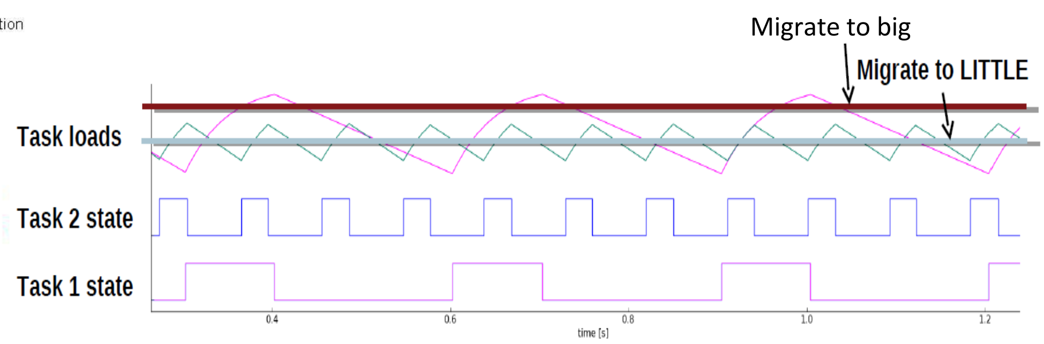
- **CPUFreq governor**
  - Performance
    - This locks the phone's CPU at maximum frequency
  - Powersave
    - This locks the CPU frequency at the lowest frequency
  - Ondemand
    - Boost clock speed to maximum on demand and step down if CPU load is low
  - Interactive
    - Similar to ondemand, but this governor dynamically scales CPU clock speed in response to workload
    - Interactive is significantly more responsive than ondemand, because it's faster at scaling to maximum
- **io_is_busy**
  - Flag that determines if waiting for IO should increase CPU utilization in bump up CPU frequency (for ondemand and interactive)
  - Tradeoff: performance vs. power

https://android.googlesource.com/kernel/common/+/android-3.4/Documentation/cpu-freq/governors.txt

Understanding CPU Behavior (2/2)

- **Characteristics of ARM big.LITTLE scheduling**
  - All interrupts are handled by CPU0
    - Load-balancing of interrupts across cores is not always the best solution*
  - Designed for power efficiency
    - Only use big cores when it is necessary**
- **What's the impact of this scheduling on I/O intensive app?**

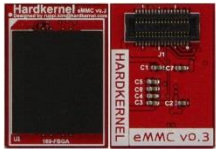




* Migrating software to multicore SMP systems (by Satyaki Mukherjee, ARM)
** Update on big.LITTLE scheduling experiments (by Morten Rasmussen, ARM)

Experiments

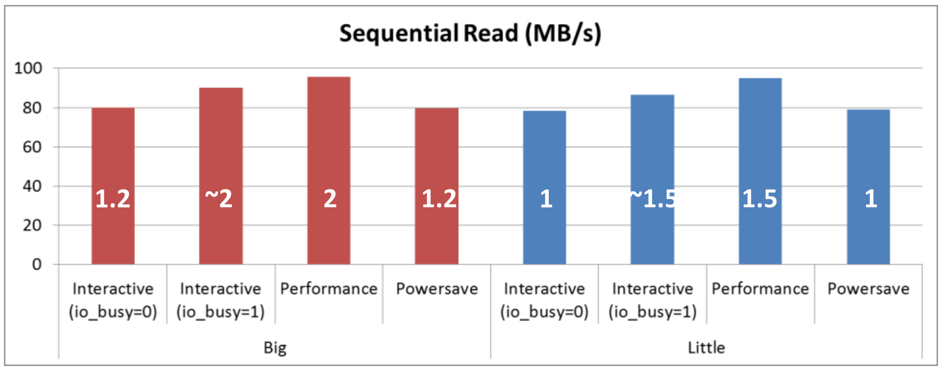
- **Hardware: ODROID XU3**
  - Exynos5422 (4x A15 1.2-2GHz, 4x A7 1-1.5GHz)
    - Little(A7): CPU0-3, Big(A15): CPU4-7
  - 2GB LPDDR3 DRAM
  - eMMC 5.0 HS400 64GB

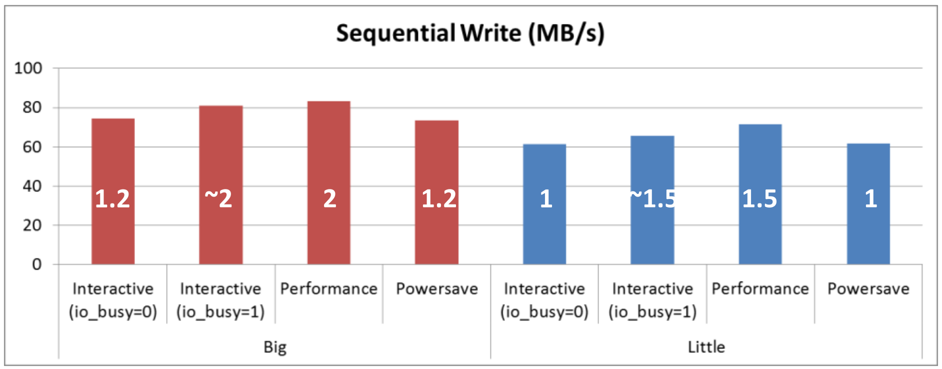


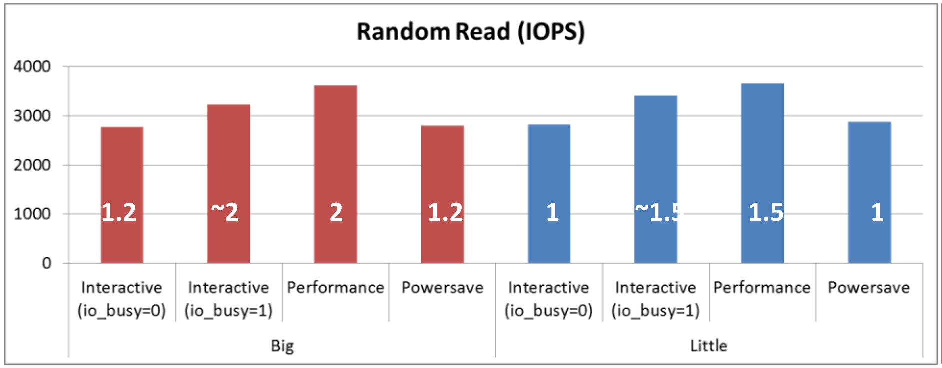
- **Software: Android 4.4.4**
  - Linux 3.10.9
- **Benchmark: fio**
  - Single thread: SW→RW→SR→RR (3 loops for each)
  - File size: 100MB (direct I/O), 1GB (buffered I/O)
  - I/O chunk: 256KB for sequential, 4KB for random
- **Parameters**
  - Governor: interactive (default), powersave (min), performance (max)
    - io_is_busy: toggle for interactive
  - Affinity: big vs. little

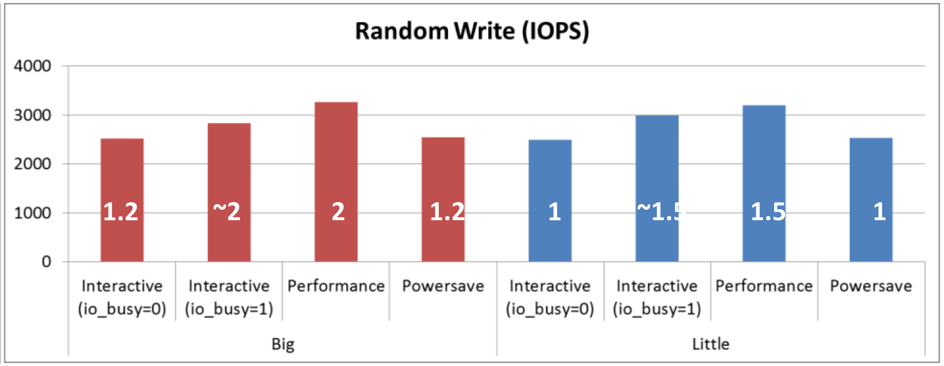
Experimental Results – 100MB Direct I/O

- **I/O throughput scales with CPU clock**
  - Performance vs. powersave: **+30% for RR & RW**, +20% for SR, +15% for SW
  - Interactive & io_is_busy=0: almost same with powersave
- **Effects of big.LITTLE**
  - +15% for SW





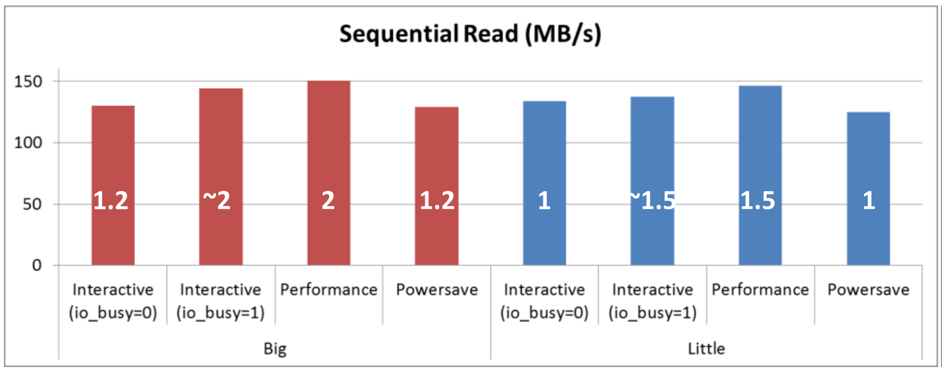


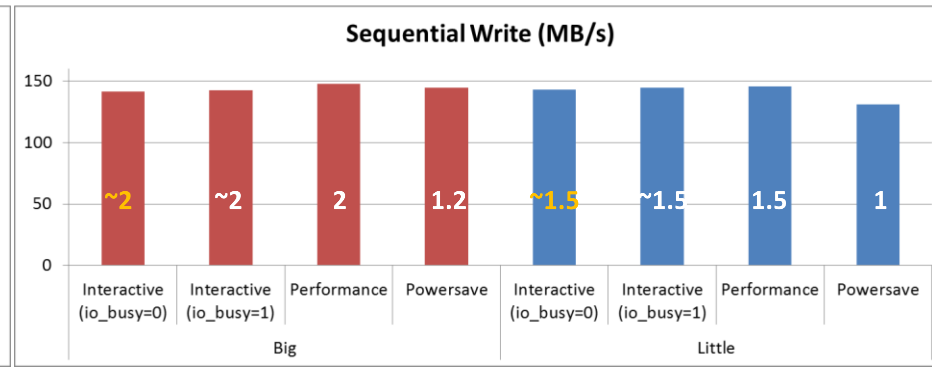


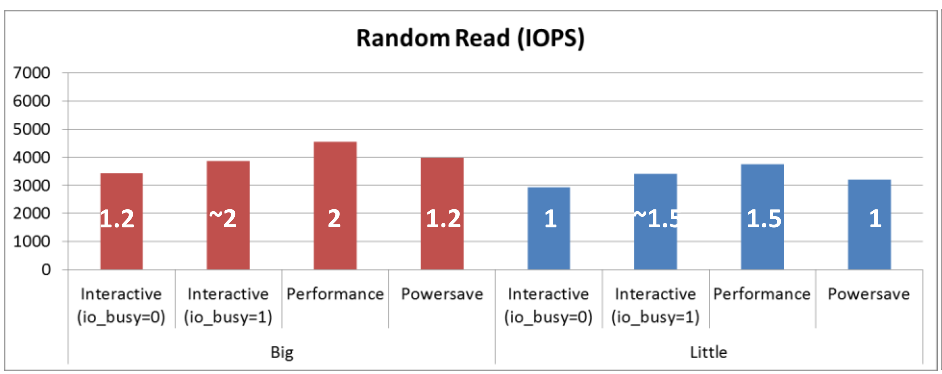
(numbers in the box means CPU clock frequency in GHz)

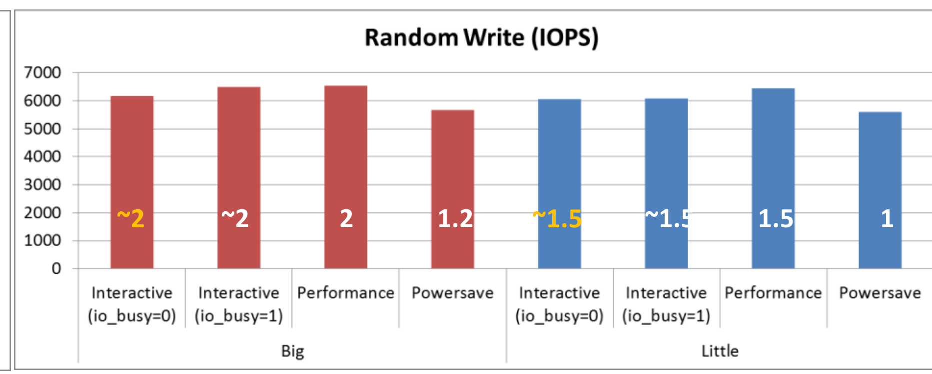
Experimental Results – 1GB Buffered I/O

- **Benchmark results are higher and less variable than direct I/O**
  - Buffered vs. direct: **+100% for SW & RW**, +50% for SR
- **RR is still CPU bound**
  - Performance vs. powersave: +30%
  - big vs. LITTLE: +20%





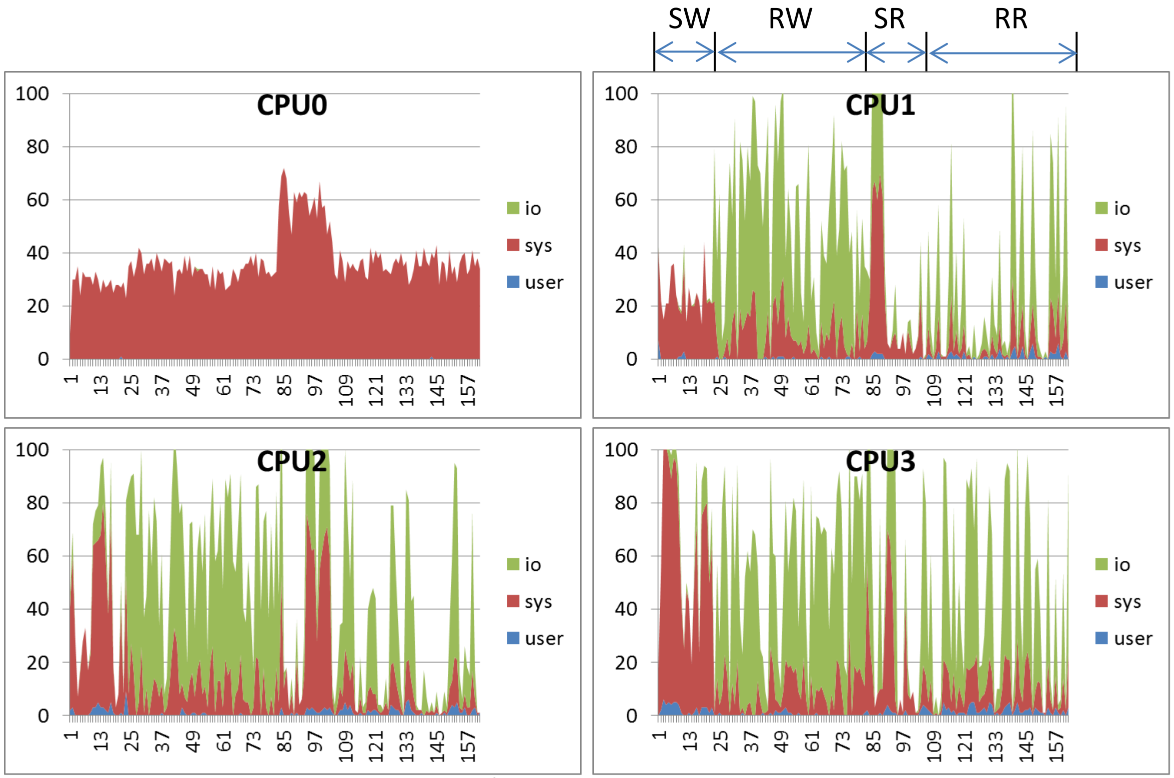




(numbers in the box means CPU clock frequency in GHz)

CPU Load & Scheduling Analysis

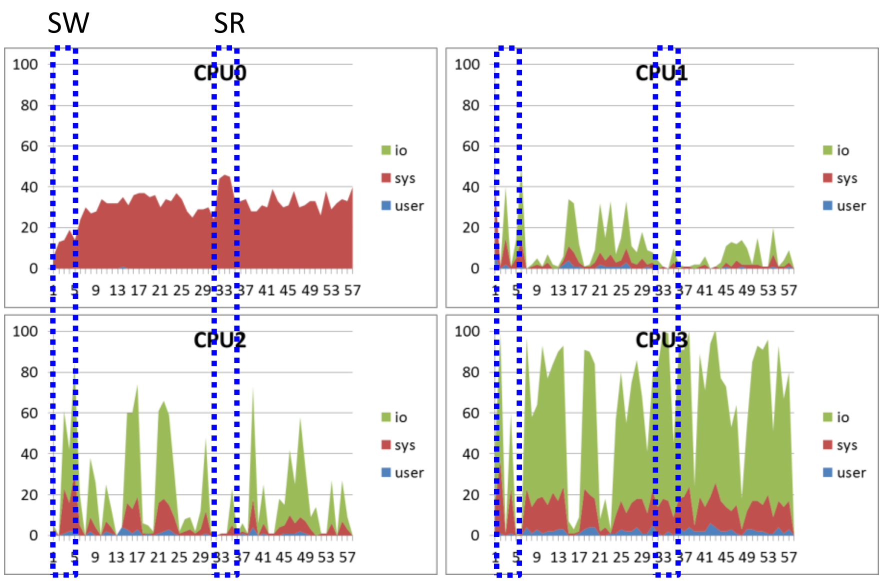
- **fio runs on 3x A7 only although all 8 cores are available**
  - fio process migrates among A7 cores
- **Issues**
  - CPU migration may be harmful for I/O intensive workload (D-cache efficiency)
  - A15 is faster at I/O handling



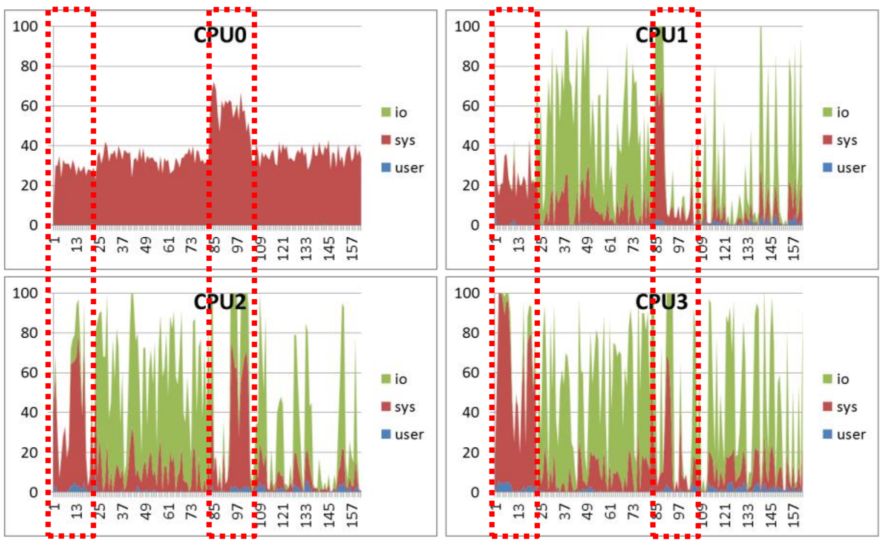
<Buffered I/O, interactive governor>

Direct I/O vs. Buffered I/O

- **Overall CPU utilization of direct I/O is lower by imbalanced %sys vs. %io**
  - Balanced means “well-pipelined”
- **Buffered sequential I/O is much faster when %sys is higher**
  - End-to-end pipeline: readahead, delayed write
- **Buffered RW is faster mainly due to eMMC cache (not CPU dependent)**



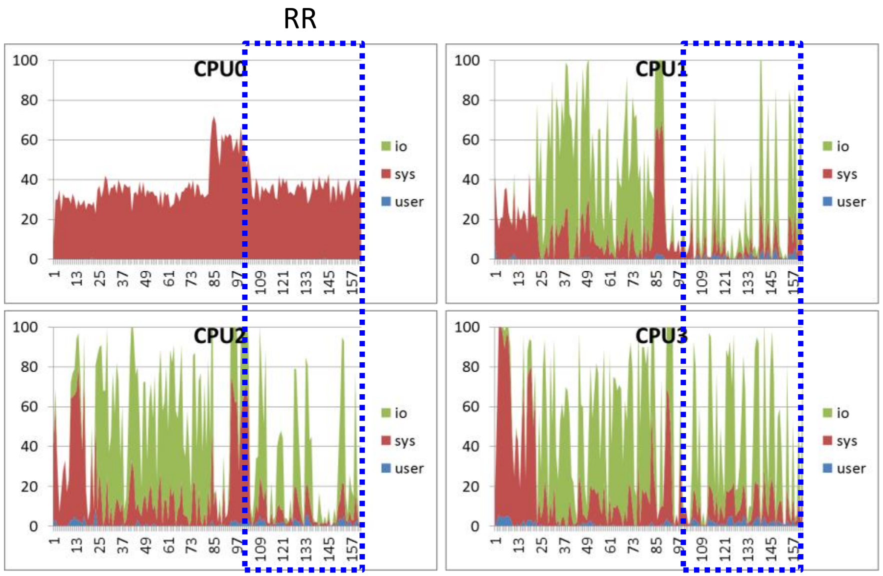
<fio in direct I/O>



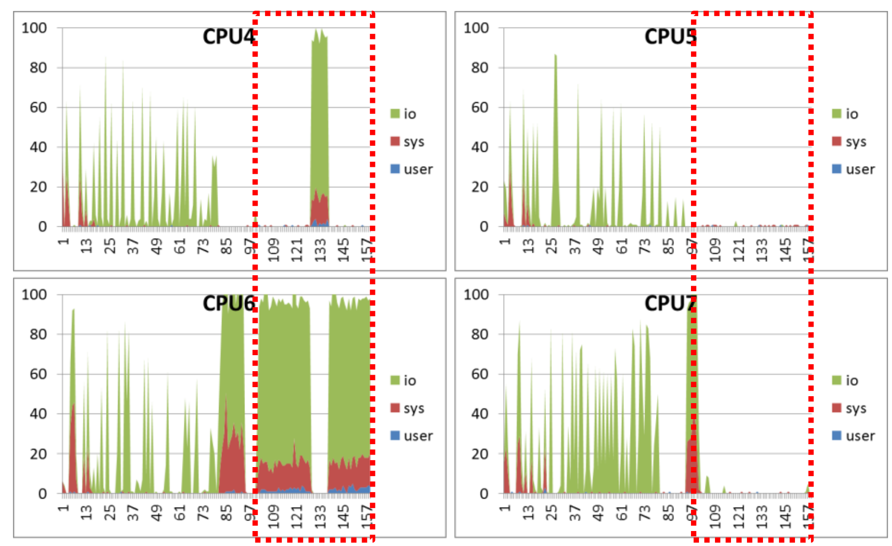
<fio in buffered I/O>

Little vs. Big

- **Buffered I/O performance is almost the same except RR**
  - CPU load is different: big has higher %io
  - **Big has potential room for improvement** if %io is balanced with %sys (more pipeline)
- **RR throughput has some relationship with CPU migration policy**
  - CPU migration: big << little



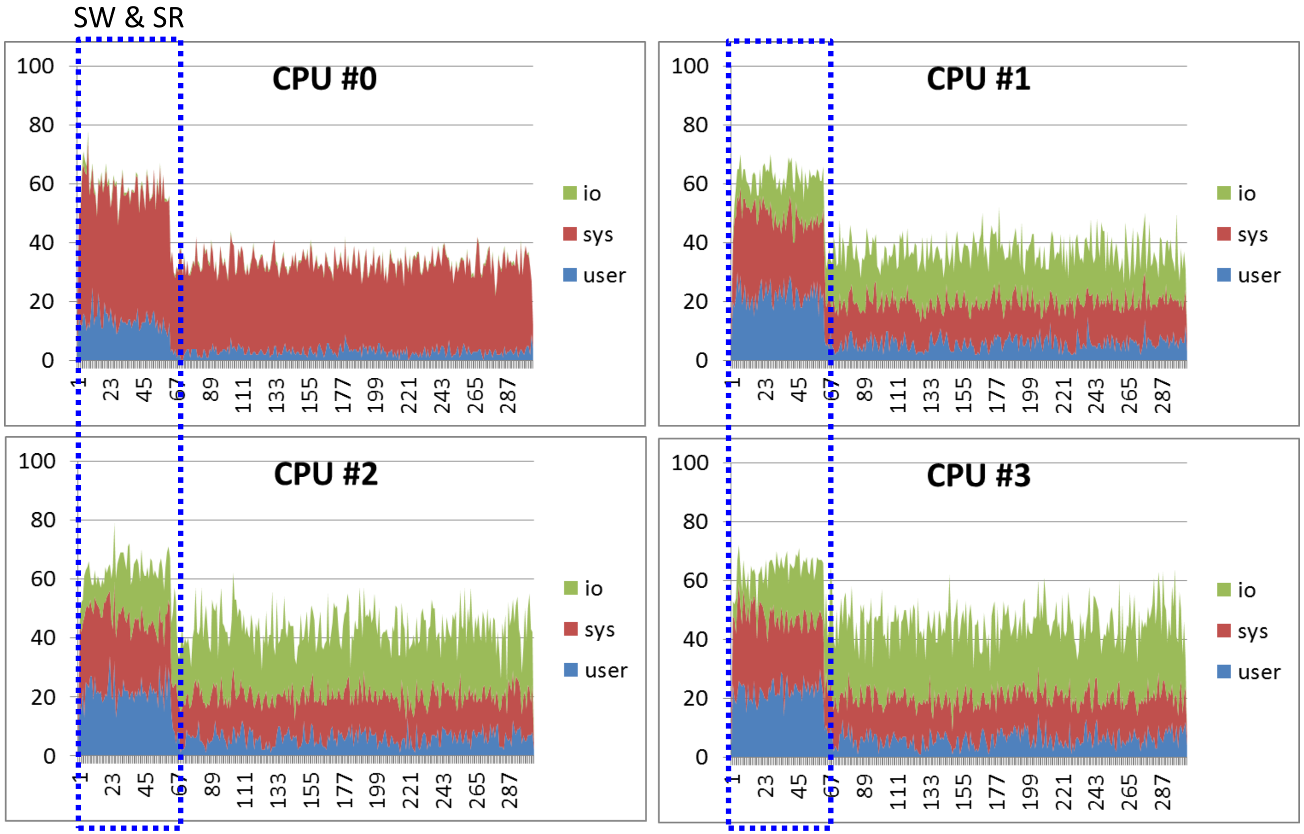
<fio running on LITTLE>



<fio running on big>

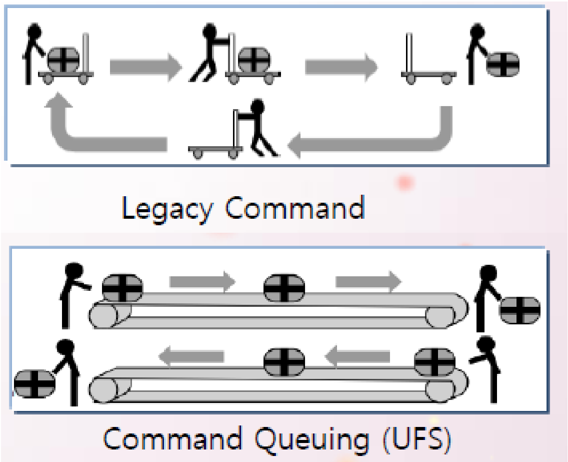
Cf. AndroBench (AnandTech)

- **I/O performance is lower than fio (direct I/O)**
  - App keeps migrating among little cores
  - CPU utilization is balanced, but is underutilized → **app is slow**

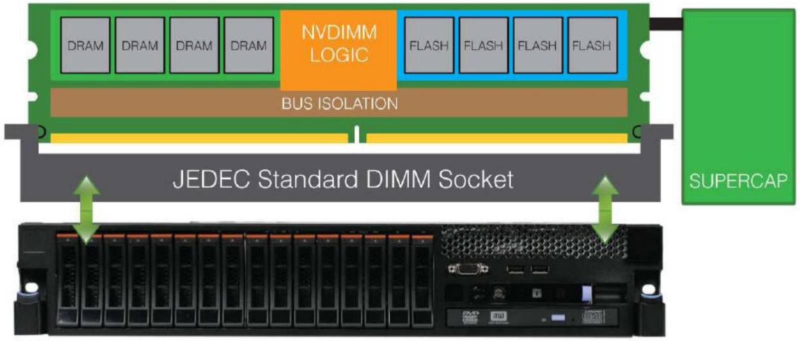


Improvement Idea

- **I/O friendly CPU scheduling**
  - ARM big.LITTLE scheduling is still in work-in-progress
- **Command queueing**
  - End-to-end parallelism by multiple I/O threads or async I/O
  - Good for benchmark vs. real user benefit
- **NVDIMM**
  - Move NVM from I/O bus to memory bus (no DMA!)
    - SNIA NVDIMM SIG (http://www.snia.org/forums/sssi/NVDIMM)
  - **OS & BIOS support is necessary**
    - Linux persistent memory API (https://github.com/pmem/linux-examples)



<UFS vs. eMMC @ NVRAMOS 2013>



<How it works @ SNIA NVDIMM Tutorial>

Revisiting Linux I/O Stack

Design of Linux I/O

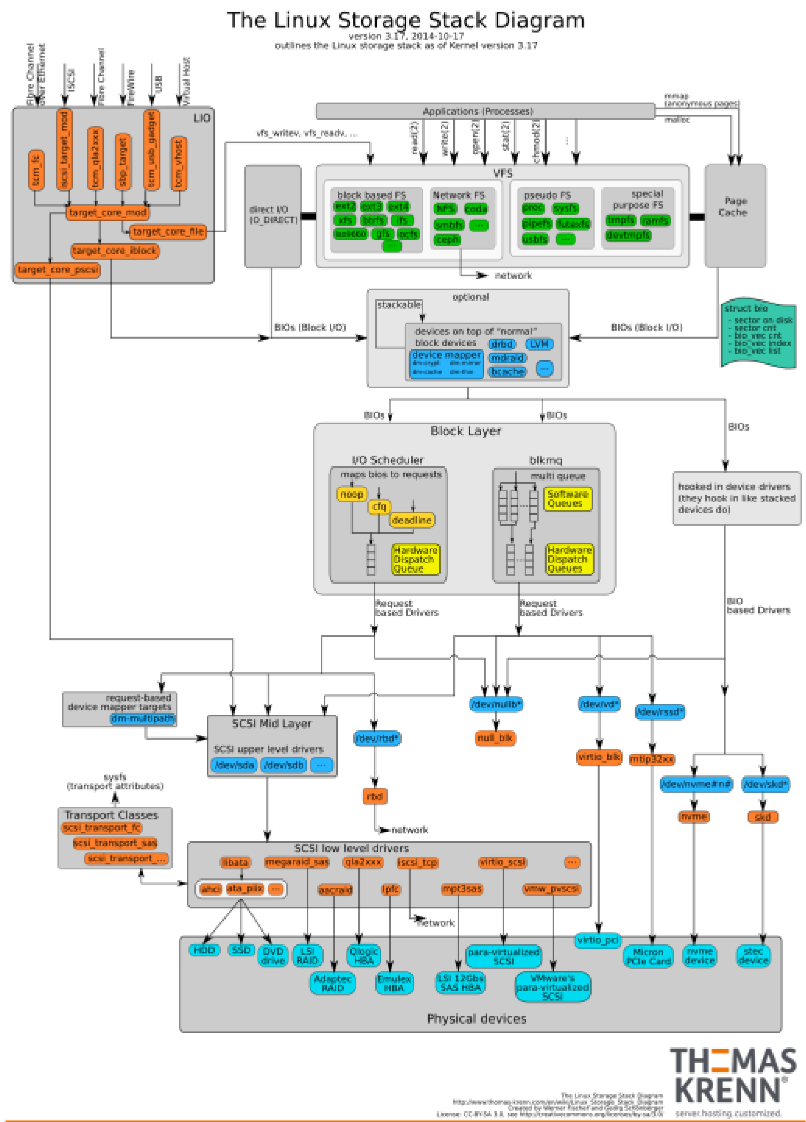
- Designed when CPU >> DRAM >> I/O
- POSIX I/O results in memory operation
- Buffered I/O, unified VM, DMA, …

CPU technology

- Clock speed race has been stopped
- Mobile computing trend puts more emphasis on **power-efficiency**

Storage

- Flash is much faster than HDD, but still **trying to mimic HDD** (FTL, position in I/O stack)
- SATA/SCSI → NVMHCI → NVDIMM(?)



http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram

Conclusion

- **What is the bottleneck if flash storage is fast enough?**
  - I/O bound: total I/O latency by software barrier – sync(), journaling by FS & DB
  - CPU bound: when CPU utilization is not balanced
- **Which I/O methods to use for benchmark?**

	Pros	Cons
Buffered I/O	Closer to device-level number, less CPU-bound	Need large file for benchmark to get consistent results
Direct I/O	Get consistent result in short time	Gap between benchmark and device number, more CPU-bound

- **Research trend keeps changing**
  - New storage → optimizing SW stack → **new SW & HW architecture**

	Improvements	Issues
CPU bound	Multi-core (homo vs. hetero)	No more CPU clock speed scaling Trends toward power-efficiency
I/O bound	Flash memory, I/O stack optimization, clustering	I/O is getting faster Deciding scale-up or scale-out