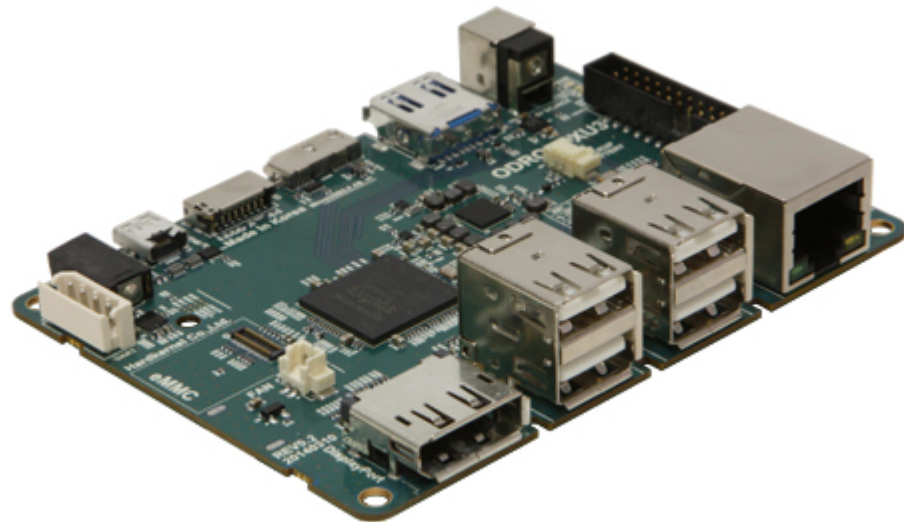


Benchmarking ARM big.LITTLE

CPE-534
Operating Systems
Spring 2020
William Pannell

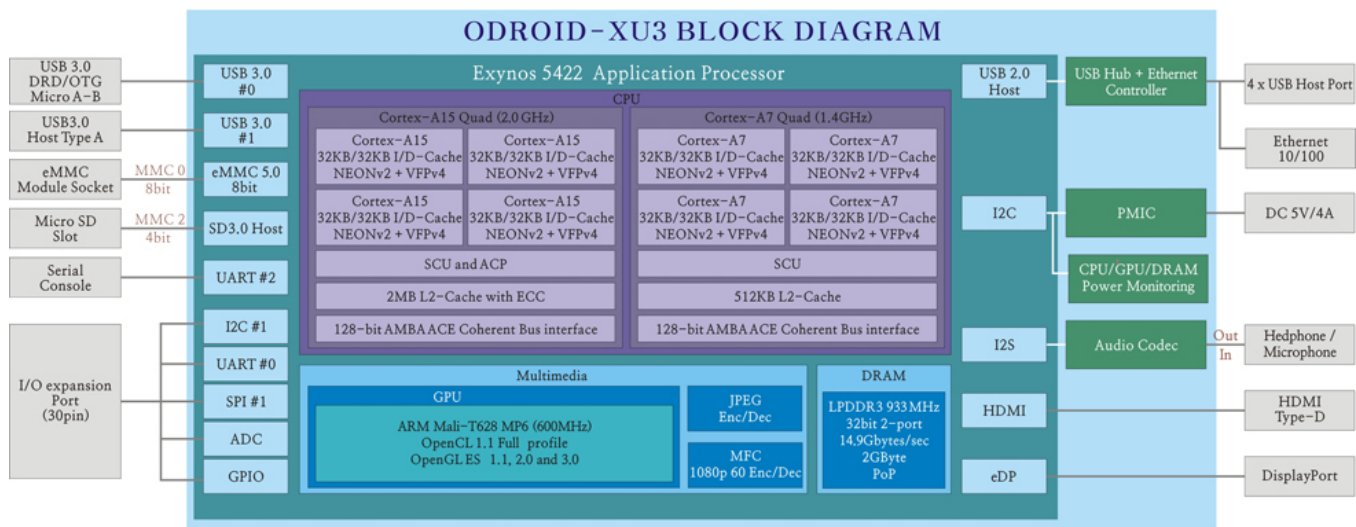
Unit Under Test

- The Odroid-XU3
 - Samsung Exynos 5422
 - 8 core big.LITTLE System on Chip
 - 4x ARM Cortex-A15 (2GHz)
 - 4x ARM Cortex-A7 (1.3GHz)
 - 2GB LPDDR3 (933MHz)
 - micro-HDMI (Mali T-628 MP6 iGPU)
 - 10/100 Mbps Ethernet
 - 4X USB 2.0, 1X USB 3.0
 - 30pin GPIO



Specs and images sourced from wiki.odroid.com

ARM big.LITTLE Architecture



Block Diagram sourced from wiki.odroid.com

Highlights:

- “big” A15 Cores:
 - 32KB I/D cache per-core
 - Cores share 2MB L2 cache
- “LITTLE” A7 Cores
 - 32KB I/D cache per-core
 - Cores Share 512KB L2 Cache
- DRAM, GPU, and IO peripherals connected via 128-bit bus.

Identifying big.LITTLE cores

- Use /proc/cpuinfo to get information about the CPU
 - “processor” gives core enumeration
 - “CPU part” Identifies the core type
 - 0xC07 → Arm A7 → LITTLE Core
 - 0xC0F → Arm A15 → big Core
 - Cores 0-3 are LITTLE, 4-7 big



```
odroid@odroid: ~  
odroid@odroid:~$ cat /proc/cpuinfo | grep -e ^processor -e ^CPU part  
processor      : 0  
CPU part      : 0xc07  
processor      : 1  
CPU part      : 0xc07  
processor      : 2  
CPU part      : 0xc07  
processor      : 3  
CPU part      : 0xc07  
processor      : 4  
CPU part      : 0xc0f  
processor      : 5  
CPU part      : 0xc0f  
processor      : 6  
CPU part      : 0xc0f  
processor      : 7  
CPU part      : 0xc0f  
odroid@odroid:~$
```

Benchmarking with



- 7Zip has builtin Benchmarking
- `7z b [N] [-mmt{N}] [-md{N}]`
 - N runs
 - `-mmt{N}` Benchmark using N threads
 - `-md{N}` Benchmark with 2^N byte sample dict
 - `-md22` yields 4MB test file
 - Tests start at 4MB and climb to `-md{N}` setting
 - 64MB test file crashes on this machine
- Each run consists of 1 Compression and 1 Decompression

Summarized from 7zip manual

<https://sevenzip.osdn.jp/chm/cmdline/commands/bench.htm>

Benchmarking with



- Compression

- Speed Depends upon:
 - RAM Latency
 - Data Cache Size/Speed (As opposed to Instruction Cache)
 - TLB hit/miss
- Large Number of Random Accesses to RAM/Cache Memory

- Decompression

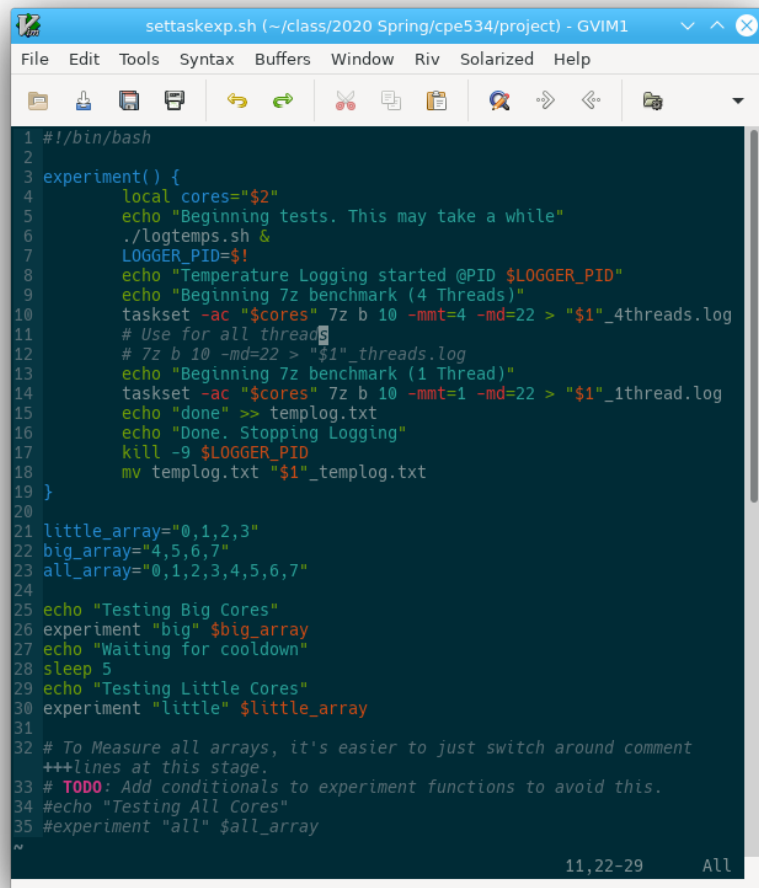
- Speed Depends Upon:
 - 32bit integer operation latency (shift/mult/add)
 - Branch (Mis)Prediction Penalty
 - Conditional Execution
- No Special Extension Instructions (SSE, NEON, etc.)

Summarized from 7zip manual

<https://sevenzip.osdn.jp/chm/cmdline/commands/bench.htm>

Methodology

- Use taskset to pin 7z to cores
- Perform 10 runs each using:
 - 4 big cores
 - 1 big core
 - 4 LITTLE cores
 - 1 LITTLE core
 - All cores
- Log temperature during tests

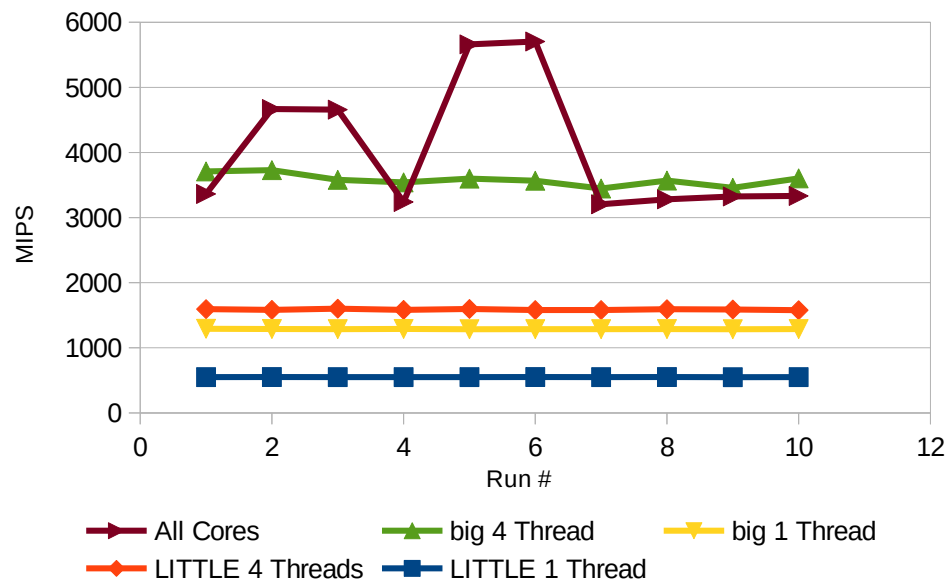


```
1 #!/bin/bash
2
3 experiment() {
4     local cores="$2"
5     echo "Beginning tests. This may take a while"
6     ./logtemps.sh &
7     LOGGER_PID=$!
8     echo "Temperature Logging started @PID $LOGGER_PID"
9     echo "Beginning 7z benchmark (4 Threads)"
10    taskset -ac "$cores" 7z b 10 -mmt=4 -md=22 > "$1"_4threads.log
11    # Use for all threads
12    # 7z b 10 -md=22 > "$1"_threads.log
13    echo "Beginning 7z benchmark (1 Thread)"
14    taskset -ac "$cores" 7z b 10 -mmt=1 -md=22 > "$1"_1thread.log
15    echo "done" >> templog.txt
16    echo "Done. Stopping Logging"
17    kill -9 $LOGGER_PID
18    mv templog.txt "$1"_templog.txt
19 }
20
21 little_array="0,1,2,3"
22 big_array="4,5,6,7"
23 all_array="0,1,2,3,4,5,6,7"
24
25 echo "Testing Big Cores"
26 experiment "big" $big_array
27 echo "Waiting for cooldown"
28 sleep 5
29 echo "Testing Little Cores"
30 experiment "little" $little_array
31
32 # To Measure all arrays, it's easier to just switch around comment
33 # +++lines at this stage.
34 # TODO: Add conditionals to experiment functions to avoid this.
35 #echo "Testing All Cores"
36 #experiment "all" $all_array
37 ~
```

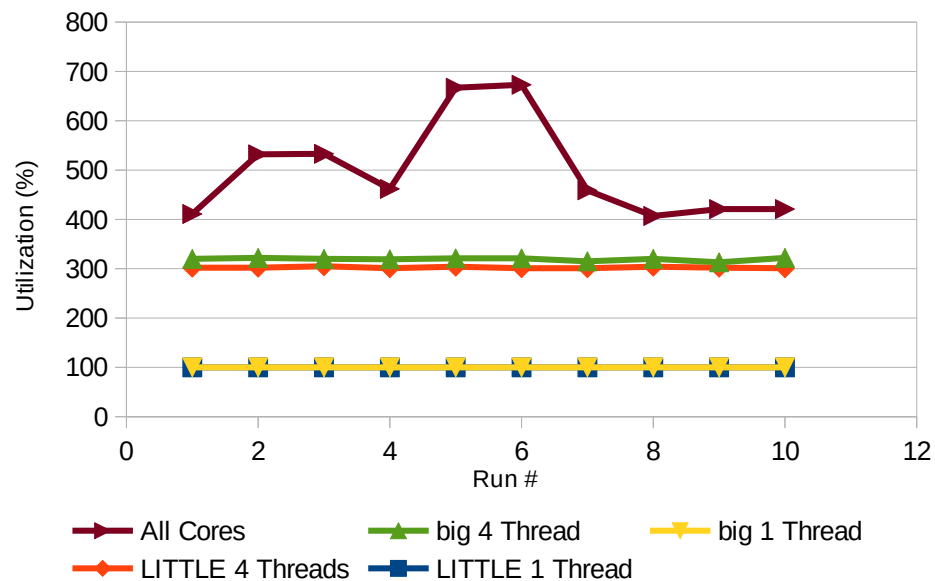
11,22-29 All

Compression Performance

Compression MIPS



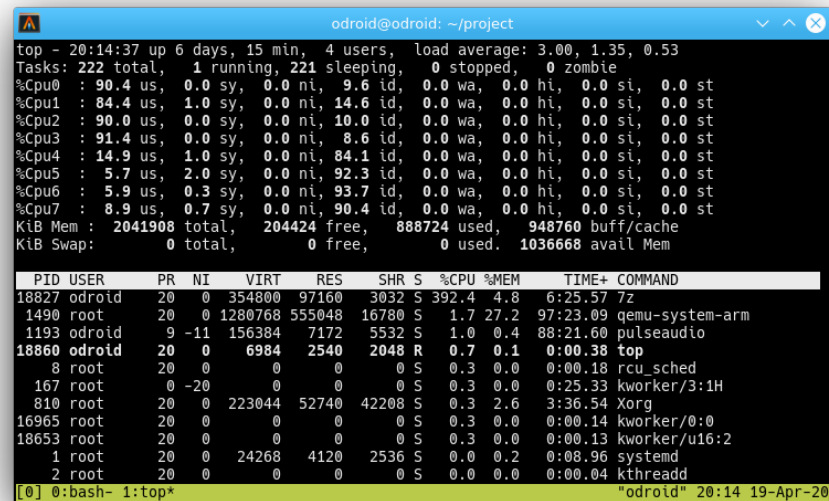
Core Utilization - Compression



Compression – Core Utilization

- Screen shot of Top Running during All Cores test
- High Utilization on LITTLE cores, Low utilization of big cores
- High idle time on big cores
- Background processes not to blame

```


    android@oidroid: ~/project
    top - 20:14:37 up 6 days, 15 min, 4 users, load average: 3.00, 1.35, 0.53
    Tasks: 222 total, 1 running, 221 sleeping, 0 stopped, 0 zombie
    %Cpu0 : 90.4 us, 0.0 sy, 0.0 ni, 9.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu1 : 84.4 us, 1.0 sy, 0.0 ni, 14.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu2 : 90.0 us, 0.0 sy, 0.0 ni, 10.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu3 : 91.4 us, 0.0 sy, 0.0 ni, 8.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu4 : 14.9 us, 1.0 sy, 0.0 ni, 84.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu5 : 5.7 us, 2.0 sy, 0.0 ni, 92.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu6 : 5.9 us, 0.3 sy, 0.0 ni, 93.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    %Cpu7 : 8.9 us, 0.7 sy, 0.0 ni, 90.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
    KiB Mem : 2041908 total, 204424 free, 888724 used, 948760 buff/cache
    KiB Swap: 0 total, 0 free, 0 used. 1367668 avail Mem

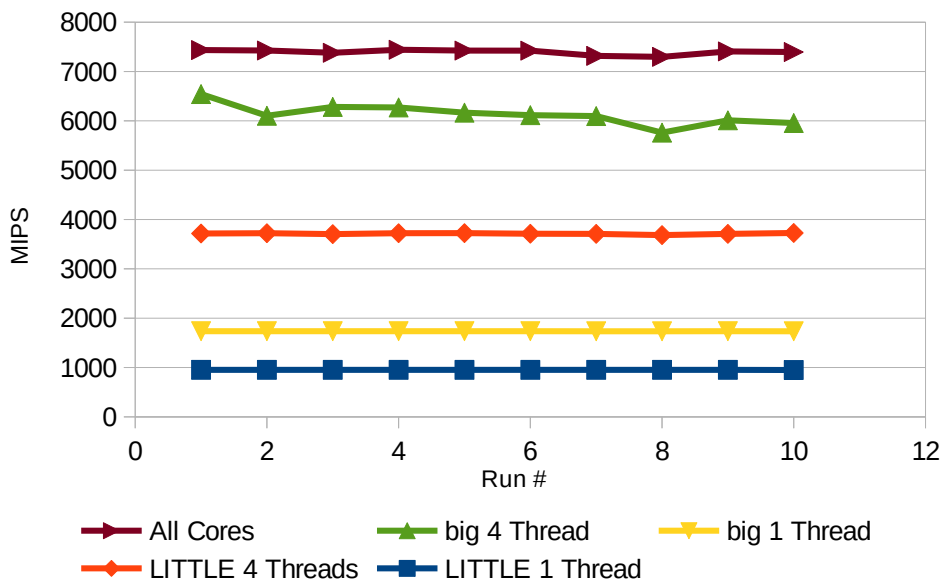
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  18827 odroid   20   0 354800  97160  3032 S 392.4 4.8   6:25.57 7z
  1490 root       20   0 1280768 555048 16780 S  1.7 27.2 97:23.09 qemu-system-arm
  1193 odroid   9 -11 156384   7172 5532 S  1.0 0.4   88:21.60 pulseaudio
  18860 odroid  20   0 6984    2540 2048 R  0.7 0.1   0:00.38 top
    8 root      20   0      0      0 0 S  0.3 0.0   0:00.18 rcu_sched
  167 root      20  -20      0      0 0 S  0.3 0.0   0:25.33 kworker/3:1H
  810 root      20   0 223044 52740 42208 S  0.3 2.6   3:36.54 Xorg
 16965 root      20   0      0      0 0 S  0.3 0.0   0:00.14 kworker/0:0
 18653 root      20   0      0      0 0 S  0.0 0.0   0:00.13 kworker/u16:2
    1 root      20   0 24268   4120 2536 S  0.0 0.2   0:08.96 systemd
    2 root      20   0      0      0 0 S  0.0 0.0   0:00.04 kthreadd

    [0] 0: bash - 1: top*
    "odroid" 20:14 19-Apr-20

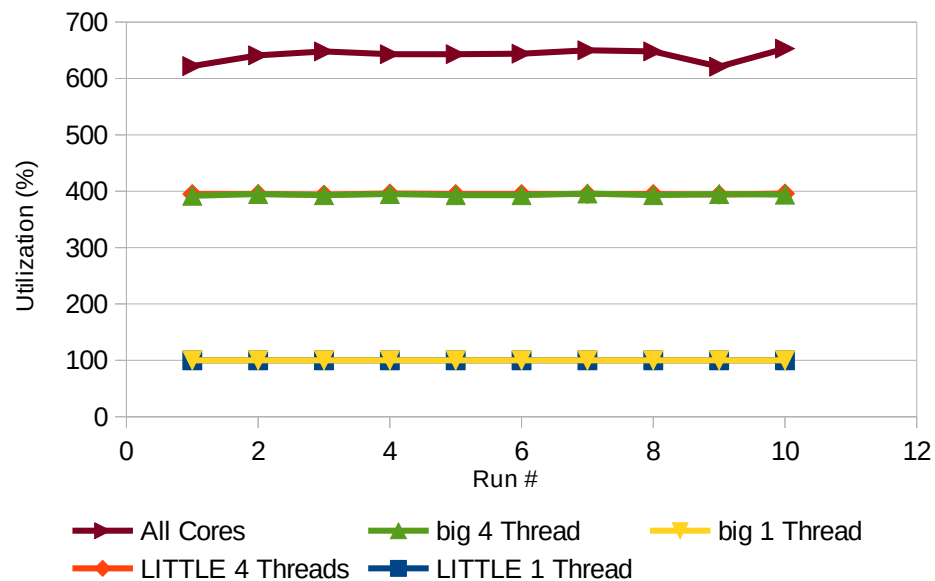
```

Decompression Performance

Decompression MIPS



























































































































































Core Utilization - Decompression



Decompression - Core Utilization

- Screen shot of Top Running during All Cores test
- Near 100% Utilization on all LITTLE cores
- 85% utilization of big cores
 - Averaging 13.5% idle time on each big core
- Background processes not to blame

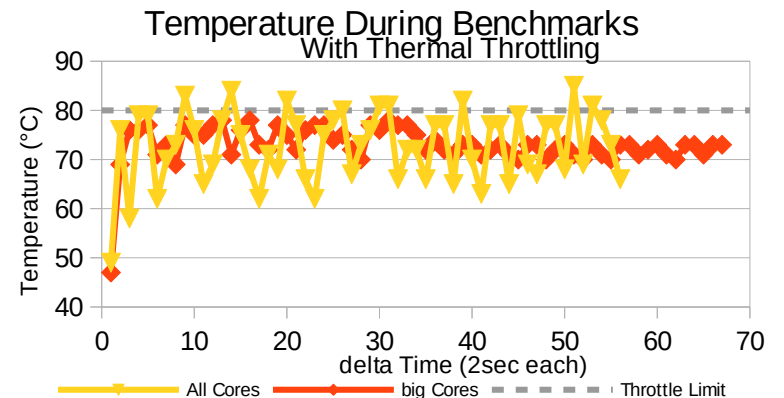
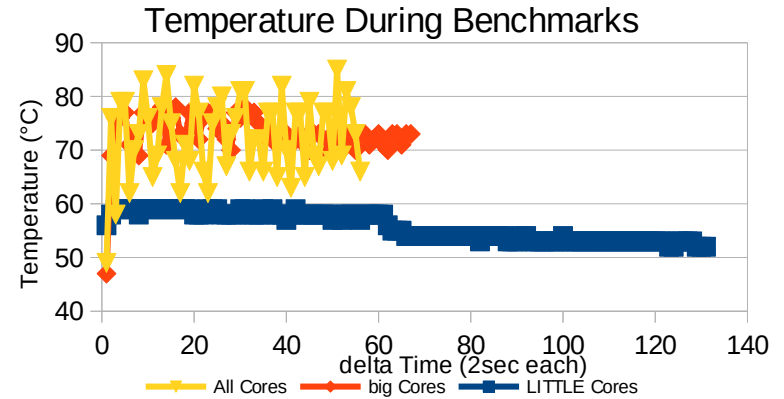
```

```

Thermal Performance

- Temperature logged during both 1 and 4 threaded tests
 - 4 Threads First
 - 1 Thread test begins at Dip
- Evidence of Throttling During “All Threads” and “big cores 4 threads” runs
- No evidence of Throttling During LITTLE cores runs.



Performance Summary

- big ~2X faster than LITTLE
- Cache constraints hurts LITTLE Compression performance
- 4 Thread Compression Experiences Bottlenecks
- LITTLE punches above weight in integer-math-heavy workloads (decompression)
- 8 Core performance falls short of expectations due to thermal constraints.

Relative MIPS	Compression	Decompression
Big 4 Thread	100.0%	100.0%
Big 1 thread	35.9%	28.3%
LITTLE 4 thread	44.3%	60.6%
LITTLE 1 Thread	15.4%	15.5%
All cores	112.9%	120.6%

Further Experimentation

- Improve Datalogging
 - Monitor core frequencies to detect throttling
 - Instrument DRAM/Cache usage (perf in linux-tools)
 - Instrument Power Consumption
 - Log each run individually
 - Allow quantifying performance differences run-to-run
- Split Compression/Decompression phases into discrete tests
 - Provides better granularity of logging.
 - No built-in support, but 7z is open source
- Reduce Variables
 - Kill other processes (X11, PulseAudio, Virtualized Guests, etc.)
 - Migrate background tasks to other cores before starting tests
- Goals
 - Verify assumed bottlenecks
 - Identify cause of low utilization during 4/8 threaded compression tests.
 - Empower reasonable decisions (when to use big vs Little), in common workloads
 - Optimize All-Core Performance vs Power consumption

Acknowledgments

- UAH Faculty and Staff
 - Providing Odroid cluster for experiments
- Dr. Kulick
 - Providing Topic/Encouragement

Raw data, Formatted Data, Test Scripts, and Presentation can be found at:
<https://github.com/wcpannell/armbiglittle>