Linux
Plumbers
Conference 2022

>> Dublin, Ireland / September 12-14, 2022

VM & sched/DVFS interaction

Saravana Kannan

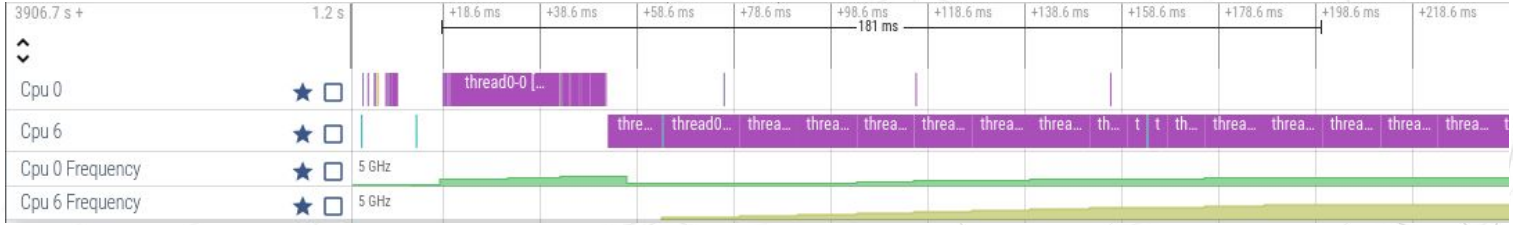
Goal

Running a use case in the host vs VM should not have any performance/power impact outside of the virtualization overhead.

On an idle host, running the use case in the host vs VM, should result in close to identical DVFS behavior of the physical CPUs and CPU selection for the threads.

Set capacity for guest thread to migrate

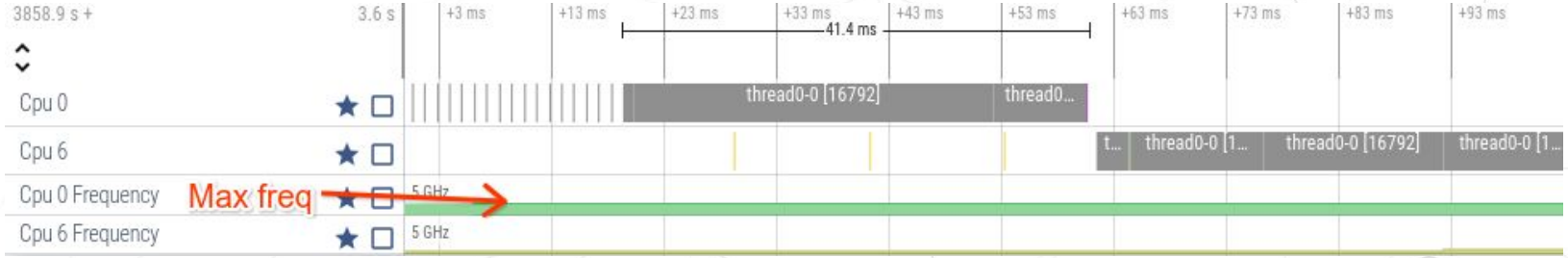
Host - 181ms to Fmax on big CPU.



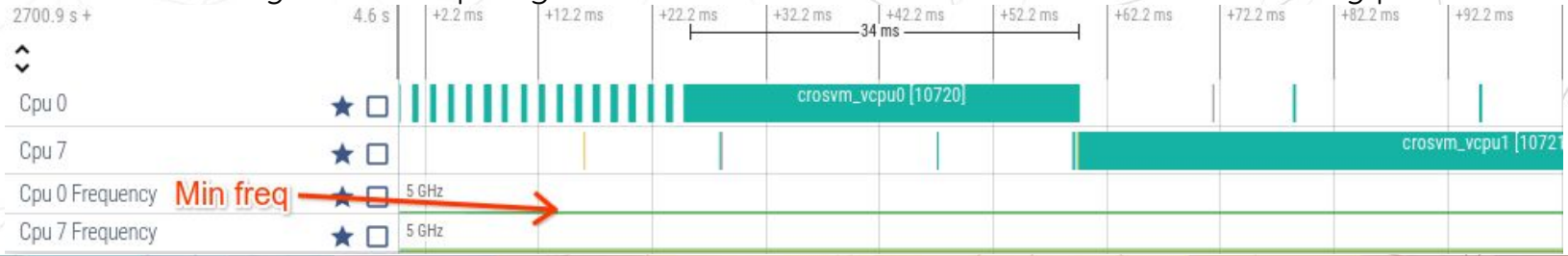
VM - 140ms to Fmax on little CPU. Guest thread never migrates to vCPU1 pinned to big CPU.

But, now things migrate too quickly

Host - Takes 41ms to up migrate when little CPU is at Fmax (shorter time to accumulate util).

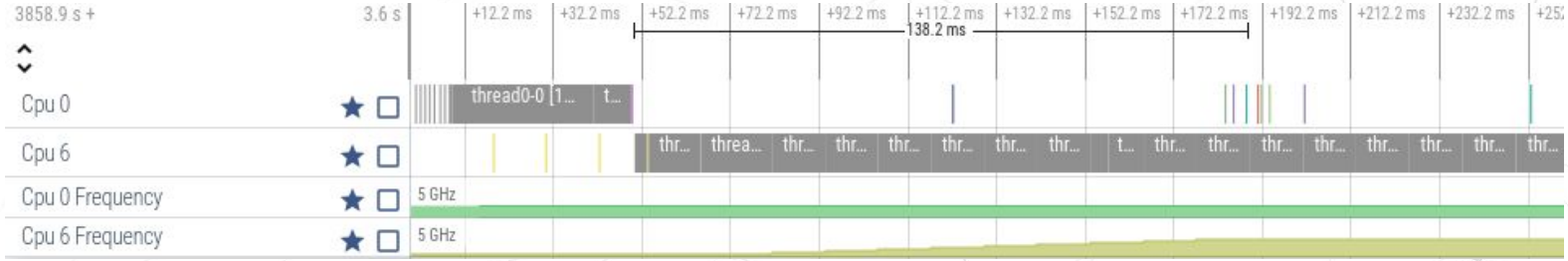


VM - Takes only 34ms to up migrate even when little CPU is at Fmin. Unnecessary power increase.

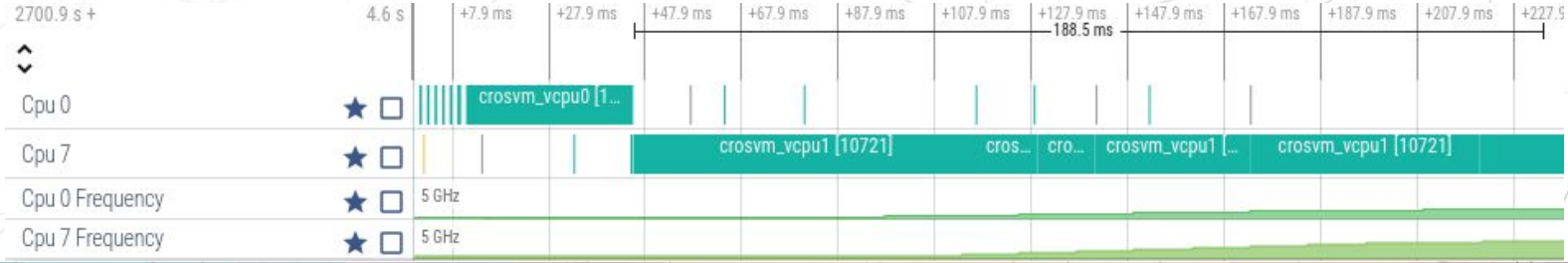


But VM performance will still be bad

Host - 138ms to get to Fmax on big CPU. Util migrated to CPU6 rq and starts at ~30% of max capacity



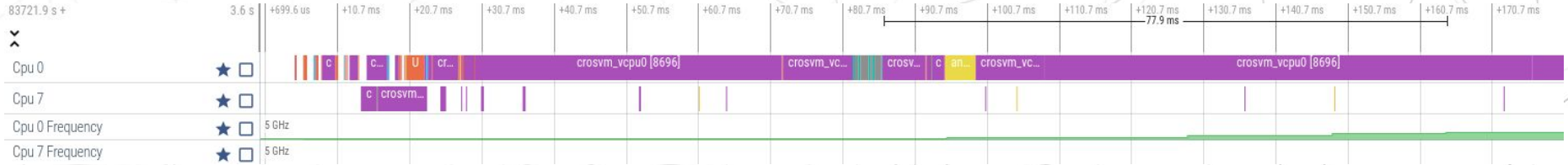
VM - 188ms to get to Fmax on big CPU. Util not migrated to vCPU1. Real big threads get worse performance



Also, new thread boost doesn't work

Host - 26ms to get to Fmax on little CPU. New thread util boosted to approx CPU0 Fmax/2 capacity.

VM - at least 77ms to get to Fmax on little CPU. New guest thread's boost is not reflected on vCPU0 util.



Other known issues

- Util clamp/util_est of guest threads is not reflected on vCPU thread.
- Source vCPU/CPU util doesn't go down when guest threads migrates away.

Game running on host container

- Android game running inside a container on a Chrome OS host.
- Threads spread across little and big CPUs.
- CPUs' frequency actually goes up/down as necessary.
- Util clamp set on background/foreground threads to manage power/performance actually makes a difference.
- Median FPS: 27

Game running on VM

- Same Android game running inside a VM on a Chrome OS host.
- vCPU thread placement is a lot more chaotic. Threads spread over all the CPUs.
- CPUs' frequency doesn't change much.
- Median FPS: 10

Proposal

- Current proposal:
  - Pin vCPUs to CPUs of same capacity/freq.
  - Set capacity of vCPU at boot to match physical CPUs it's pinned to.
  - Implement VM cpufreq driver that:
    - Fakes same set of frequencies as physical CPU.
    - Calls to its target() op ends up setting util_est/util of vCPU thread on the host side.
  - Use AMU or implement periodic physical CPU freq query from VM to set vCPU's freq.
- Other options?
  - We don't want to have vCPU migrate between big/Little CPUs. Changing capacity after boot up seems too messy/unnecessary.
  - We also needs VMs to be able to force some guest threads to little CPUs and unpinned vCPUs would prevent that.
  - Any other ideas?

Side bar

- We just need CPU cycle counter and don't need fixed reference clock cycle counter to do freq normalization of util.
- Today:
  - Every scheduler tick:
    - Fcur = CPU cycles / (Ref clock cycles * Ref clock period)
  - Every util update:
    - normalized_time = real time * (Fcur/Fmin) * (CPU_cap/1024)
- Proposal:
  - One time:
    - CPU_cycle_period (nanosec) = (10^9 / Fmax) * (CPU_cap/1024)
  - Every util update:
    - normalized_time = CPU cycles * CPU_cycle_period
- Removes need to query physical CPU frequency even without AMU.
- For VM/Host, more accurate util tracking when frequency changes between scheduler ticks.

Linux
Plumbers
Conference 2022

>> Dublin, Ireland / September 12-14, 2022

Thank you!