Linux
Plumbers
Conference | Richmond, VA | Nov. 13-15, 2023

CPUfreq/sched & VM guest workloads problems

Saravana Kannan <saravanak@google.com>
David Dai <davidai@google.com>

Quick Overview

Continuing my LPC 2022 talk

Workloads running inside a VM have terrible power/performance when compared to running on the host even if you assume virtualization overhead is zero.

- Load tracking/task placement inside VM is broken because:
  - VM has no awareness of architecture differences between pCPUs (Eg: big/little, P/E)
  - VM has no awareness of pCPU min/max/current frequency
- Host side CPU frequency scaling is broken because:
  - Host has no idea of load on each vCPU.
  - Host has no idea of workload types inside vCPU (eg: io_boost)

How did we address it?

Architecture awareness and CPU min/max frequencies

Added crosvm support to insert the relevant DT properties:

- Add dmips-per-MHz to CPU nodes to make VM aware of CPU differences
- Add operating-points-v2 table to CPU nodes to make VM aware of CPU min/max frequencies

How did we address it?

Current CPU frequency and load awareness

CPU frequency MMIO device:

- Register to get vCPU's frequency - Traps to host and gets the pCPU's current frequency

When the VM doesn't support virtualized arch CPU performance counters, this register is used in a similar fashion and is registered with the scheduler.

- Register to set vCPU's "hardware" frequency - Traps to host and set vCPU thread's uclamp min

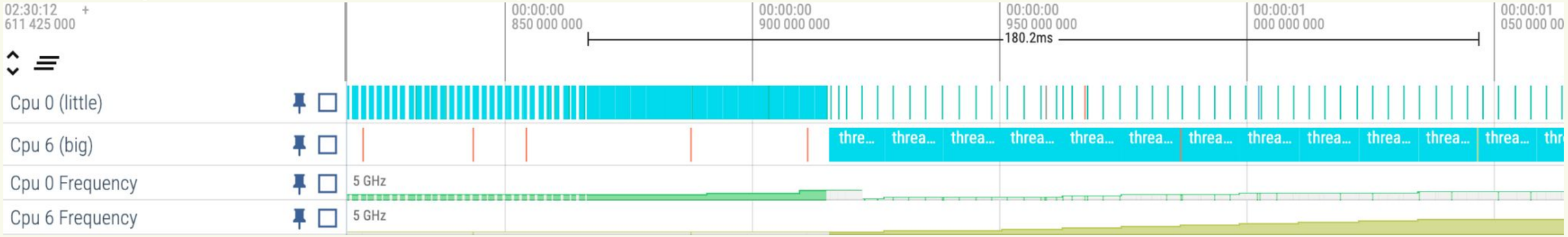
The host is indirectly made aware of the vCPU loads by the VM cpufreq governor setting this register.

Linux
Plumbers
Conference | Richmond, VA | Nov. 13-15, 2023

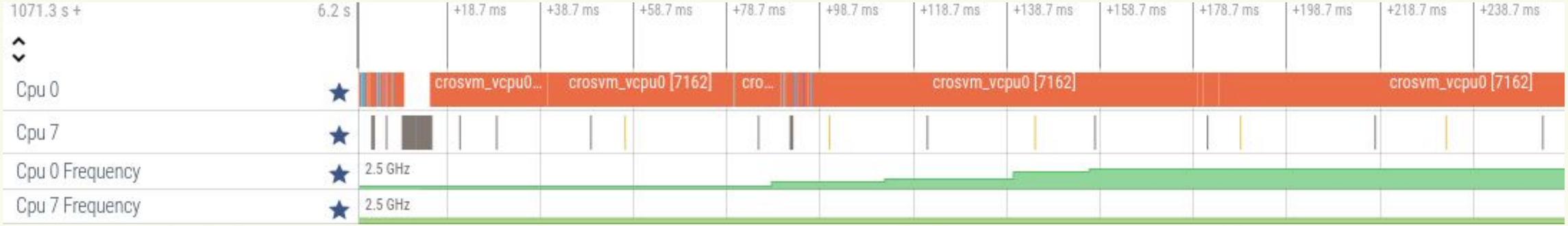
Does it work at a fundamental level? Synthetic workloads

Fmin on Little to Fmax on big

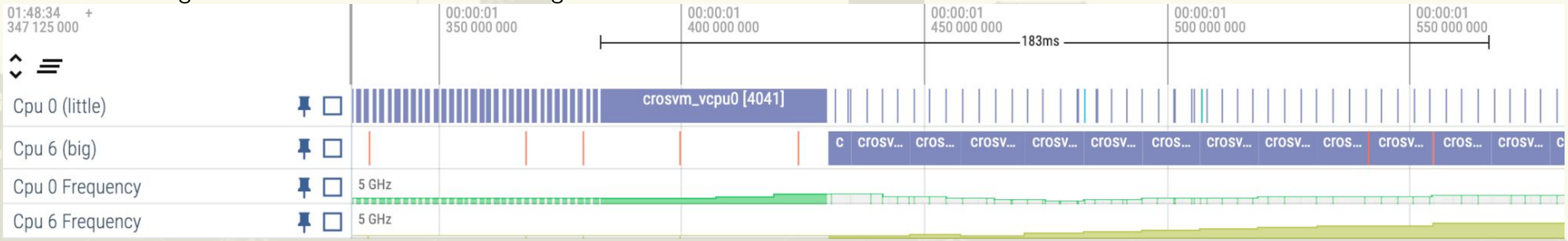
Host: **180ms** to go from Fmin on Little to Fmax on big



VM before: **Infinity**. Never upmigrates!

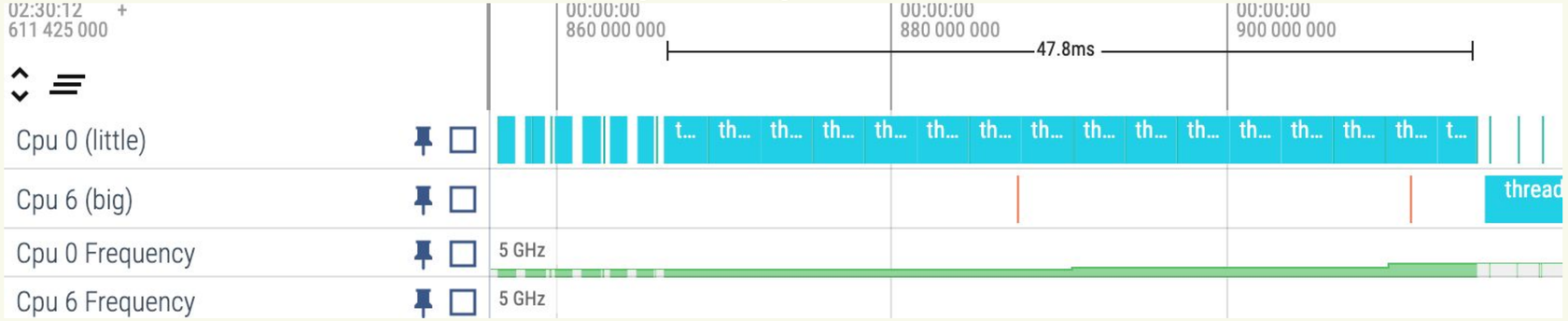


VM after: **183ms** to go from Fmin on Little to Fmax on big

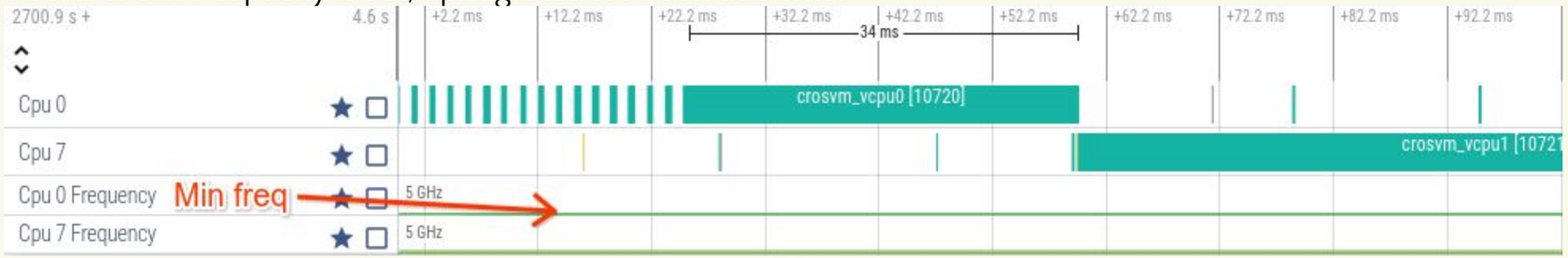


Time to migrate to big

Host: **47ms** to hit Fmax on Little and then upmigrate to big



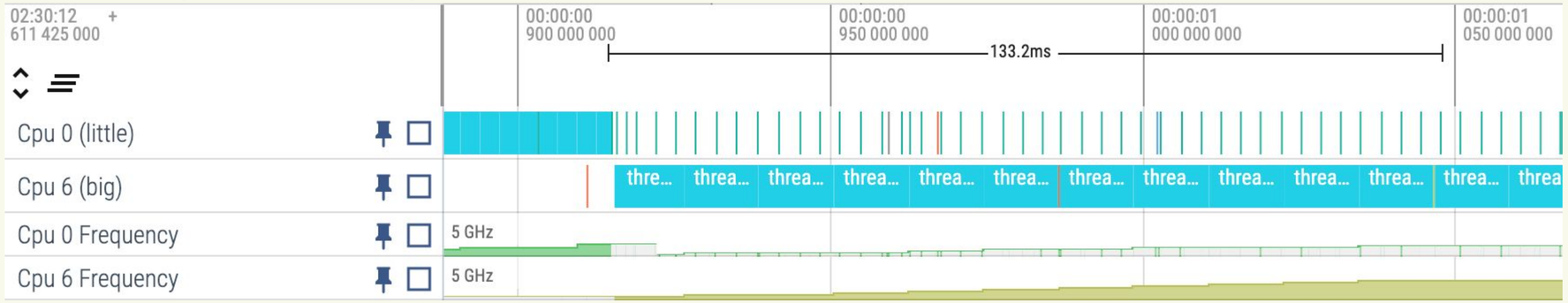
VM before: Once capacity is set, upmigrates too soon at **34ms**.



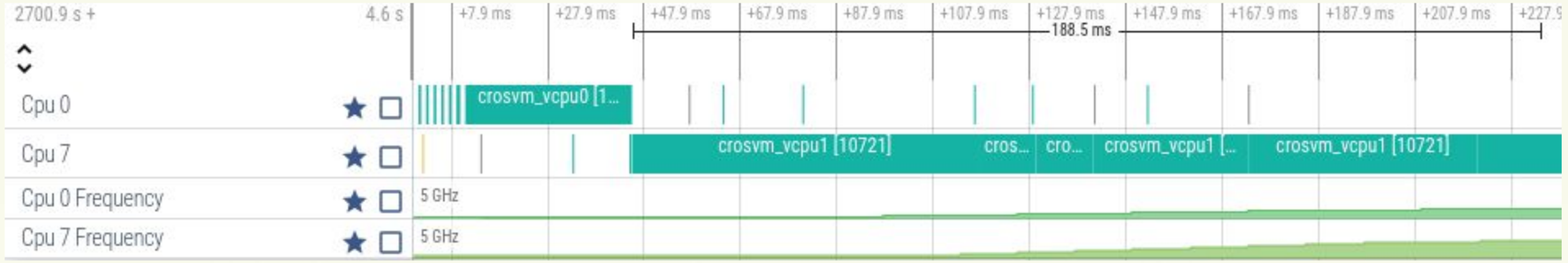
VM after: **46ms** to hit Fmax on Little and then upmigrate to big

Time to Fmax on big

Host: **133ms** to get to Fmax on big



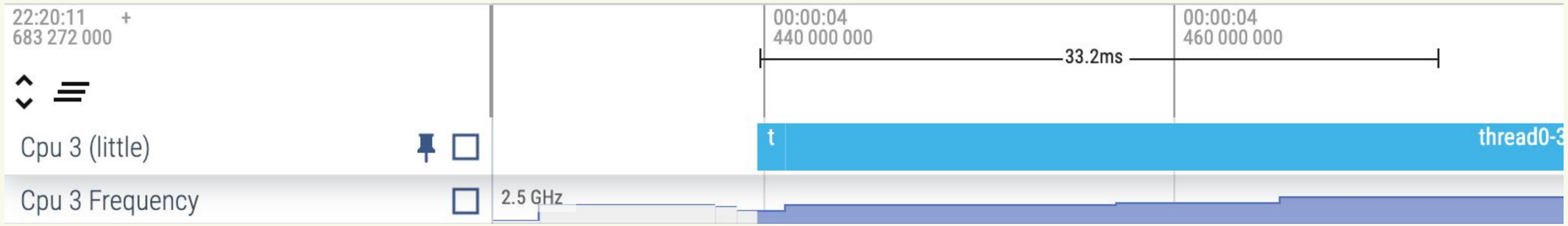
VM before: **188ms** to get to Fmax on big



VM after: **135ms** to get to Fmax on big

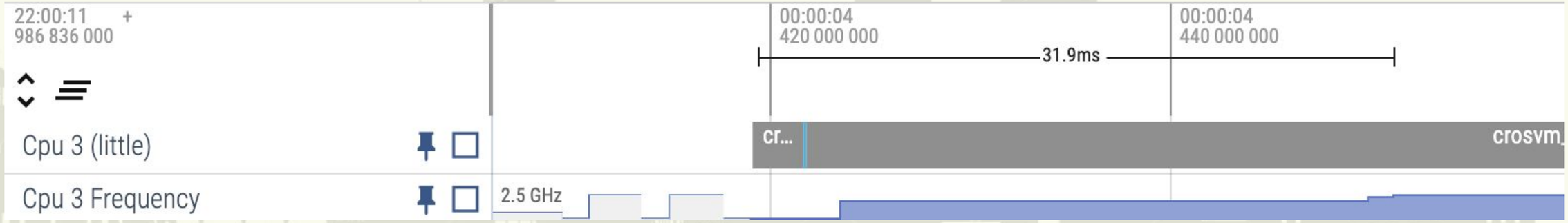
New thread boost

Host: **33ms** to Fmax on Little



VM before: **78ms** to Fmax on Little

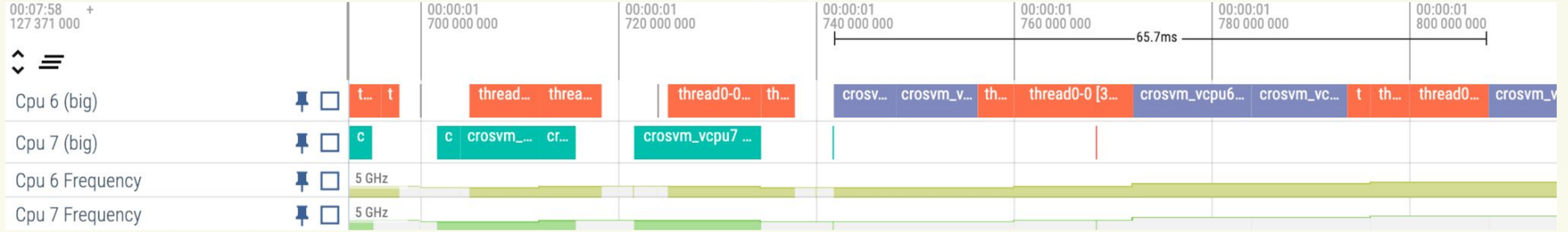
VM after: **32ms** to Fmax on Little



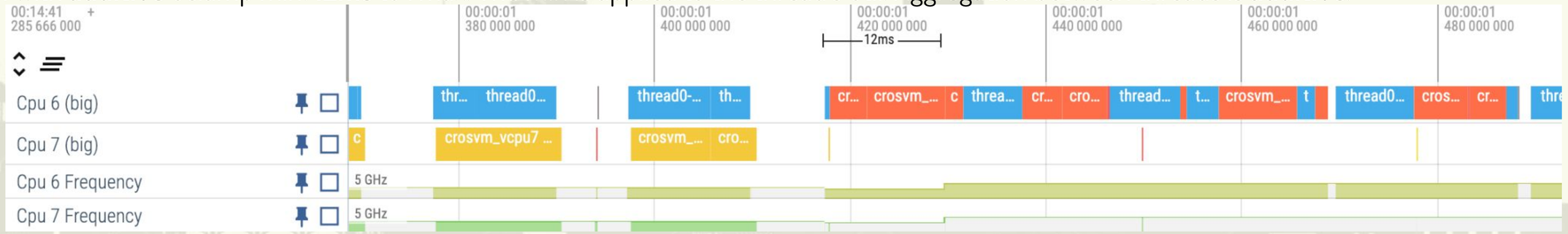
uclamp aggregation issues

Host: **12ms** to go to Fmax

VM with **default** uclamp: **66ms** to Fmax. The min is applied to rq util. Aggregation between threads is **"max of min"**



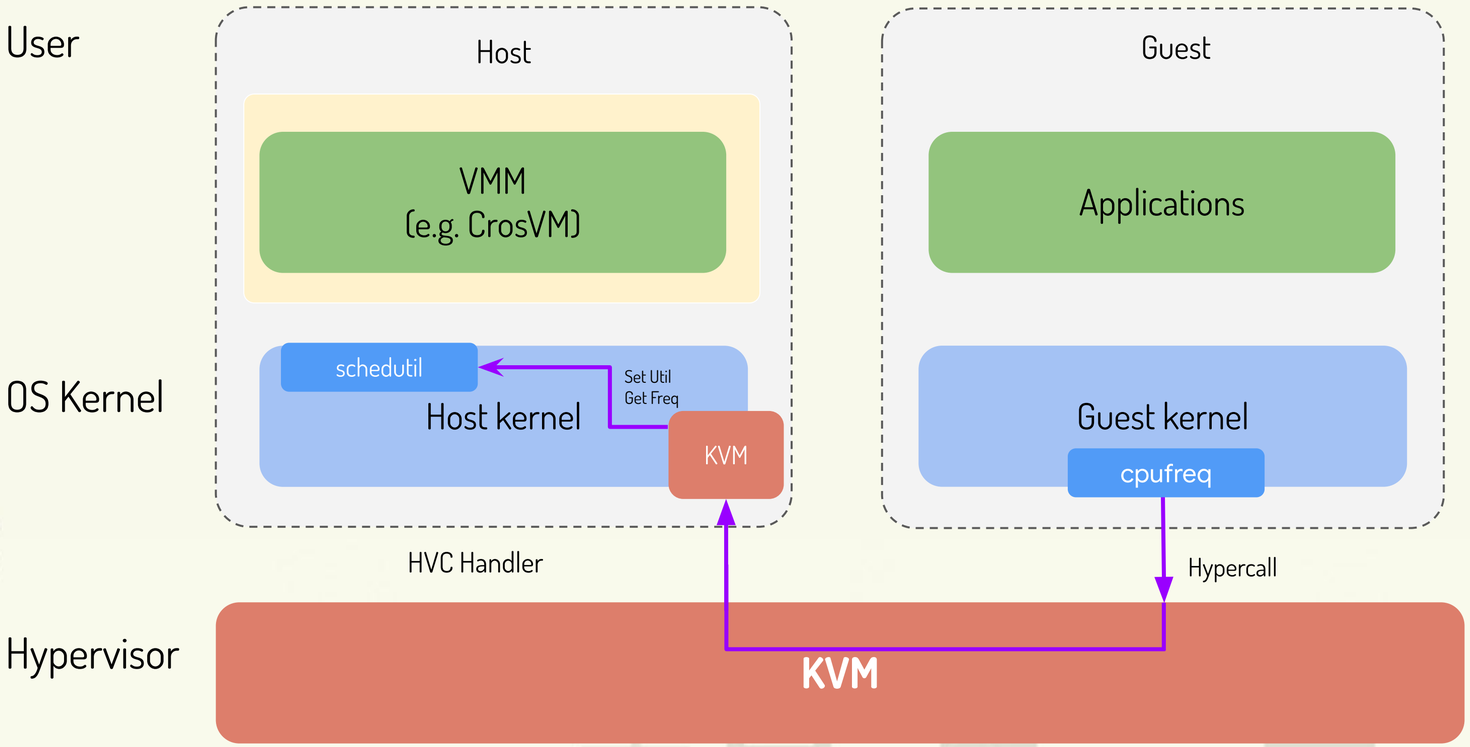
VM with **additive** uclamp min: **12ms** to Fmax. The min is applied to the thread's util. Aggregation between threads is **additive**.



Deeper dive: upstream patches/plans

Deeper dive

v1: Hypercall + util_guest prototype



Pros:

- No expensive MMIO exit to userspace required, resulting in fast performance
- Adding a new scheduler signal in the host kernel allows for more accurate/better scheduler behavior

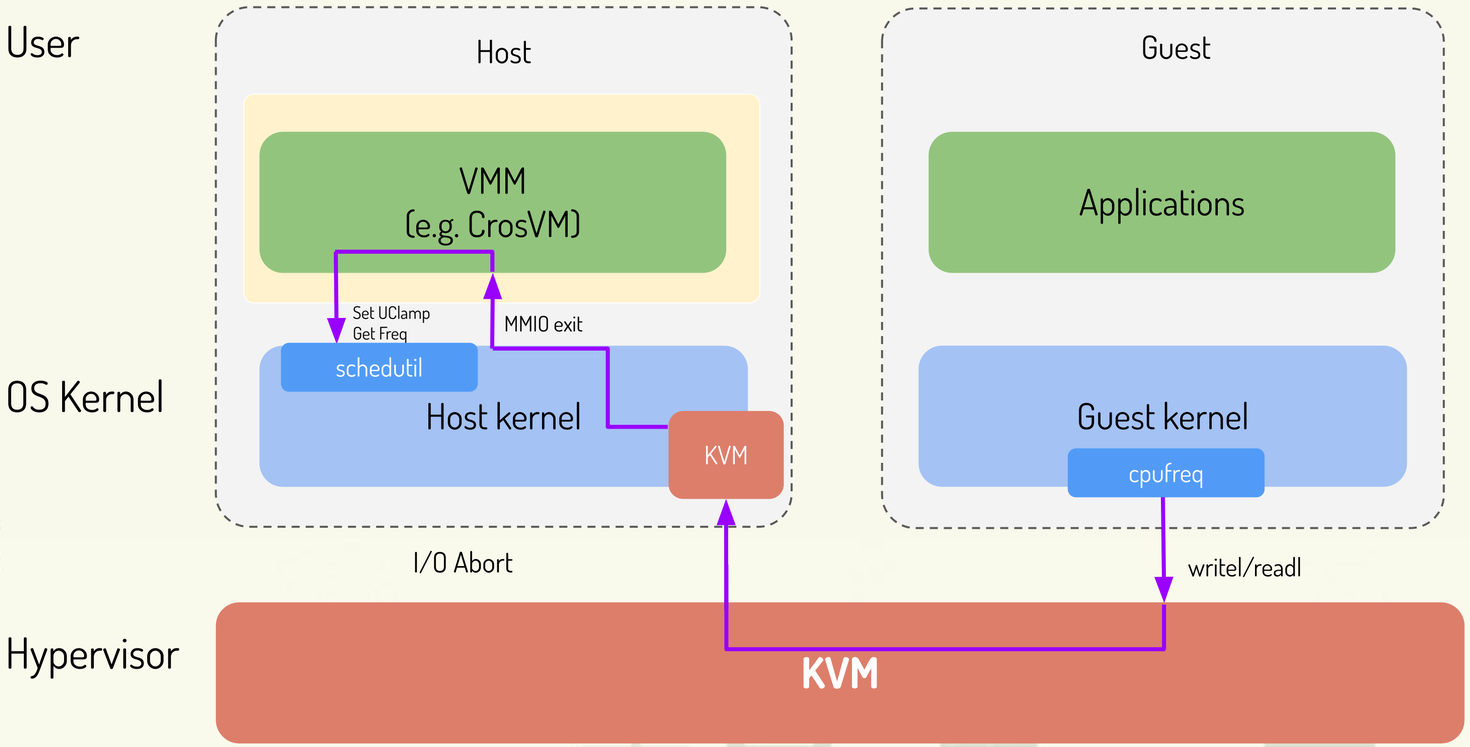
Cons:

- Hypercalls are hypervisor specific and require custom implementation
- Inflexible, policy is determined by the kernel
- Difficult to maintain due to guest ABI considerations [1]

1. Discussion on LKML

Deeper dive

v3: MMIO + uclamp



Offset	0x0	0x4	0x8	...
Register	CPU0_GET_FREQ	CPU0_SET_FREQ	CPU1_GET_FREQ	...

Pros:

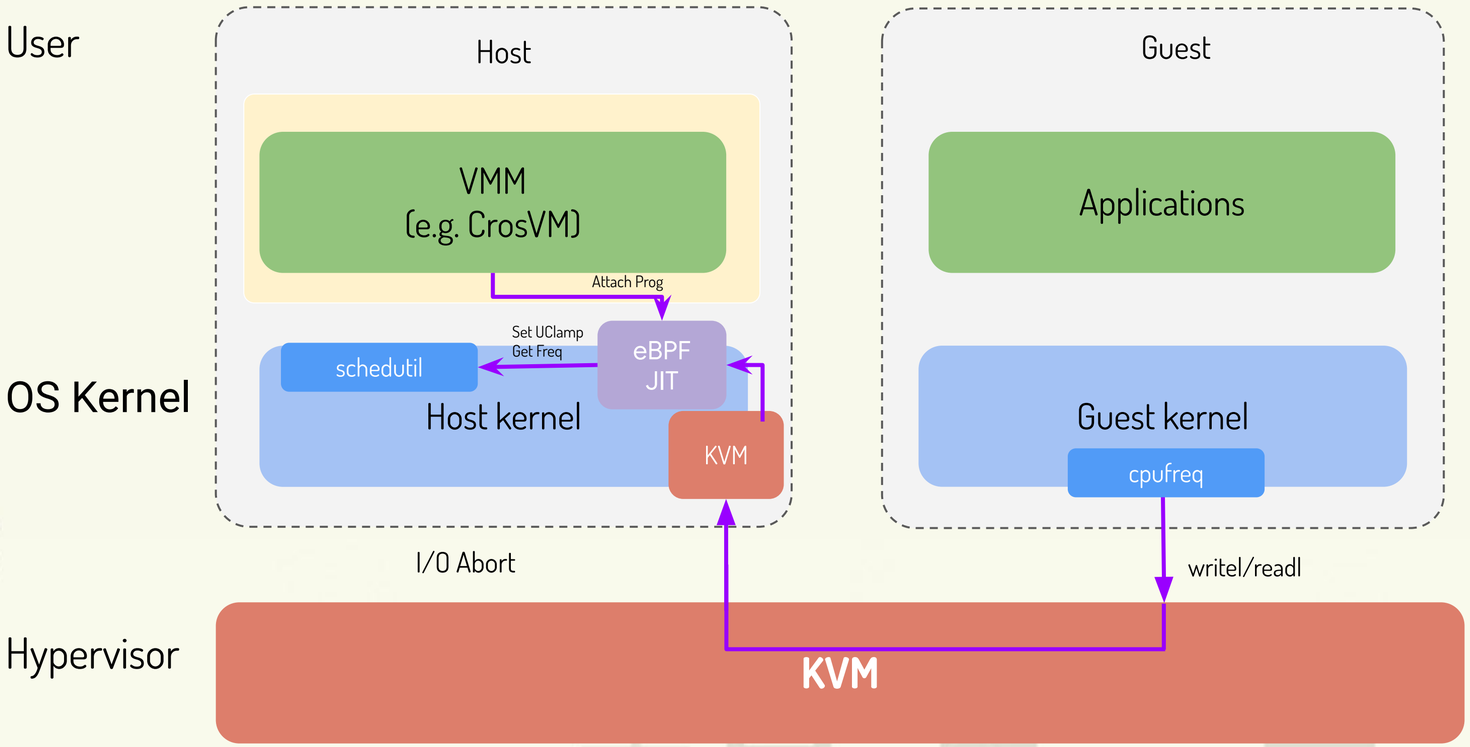
- No host side kernel changes required.
- Simple/Generic interface between host and guest
- Host OS/Hypervisor agnostic
- Policy decided by userspace

Cons:

- Performance suffers due to expensive context switches(High latency per exit)
- Limited by the syscall interface where some UClamp may lacks expressiveness

Deeper dive

v3 + eBPF



Offset	0x0	0x4	0x8	...
Register	CPU0_GET_FREQ	CPU0_SET_FREQ	CPU1_GET_FREQ	...

Pros:

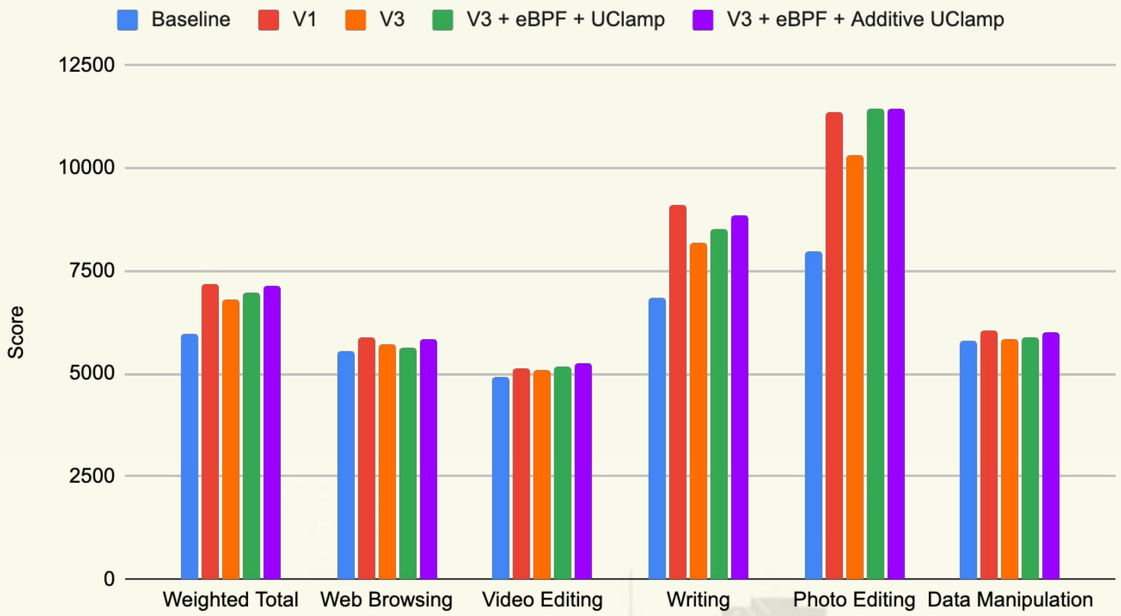
- Flexible/Portable ABIs
- Handles Guest MMIO aborts without exiting to userspace
- eBPF progs can optionally defer MMIO exits to userspace
- eBPF progs can be updated without having to update kernel/userspace

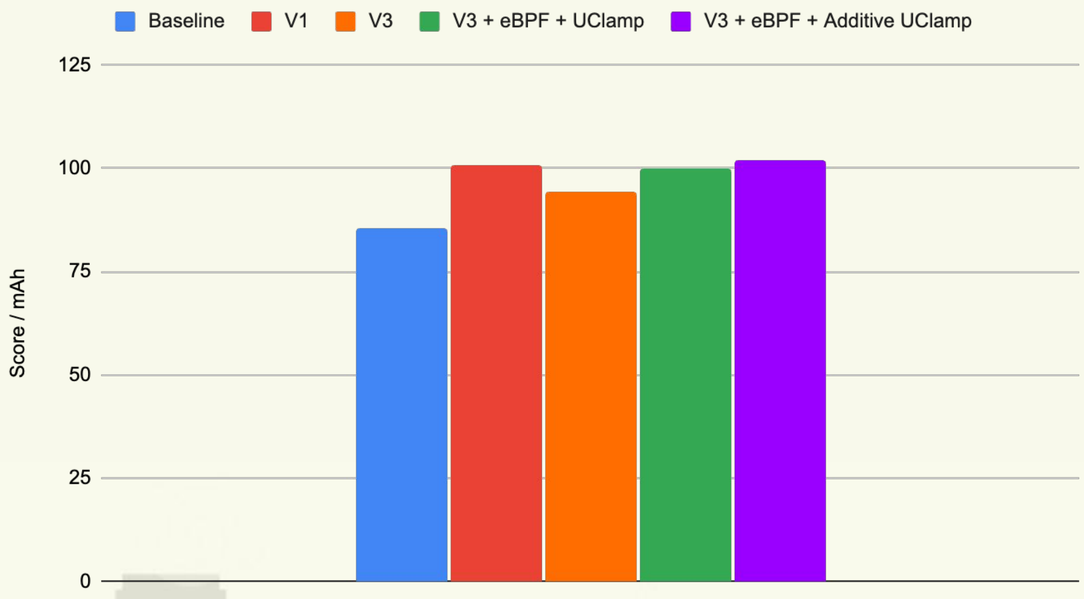
Cons:

- eBPF permission controls are less flexible(no SECCOMP for bpf helpers)

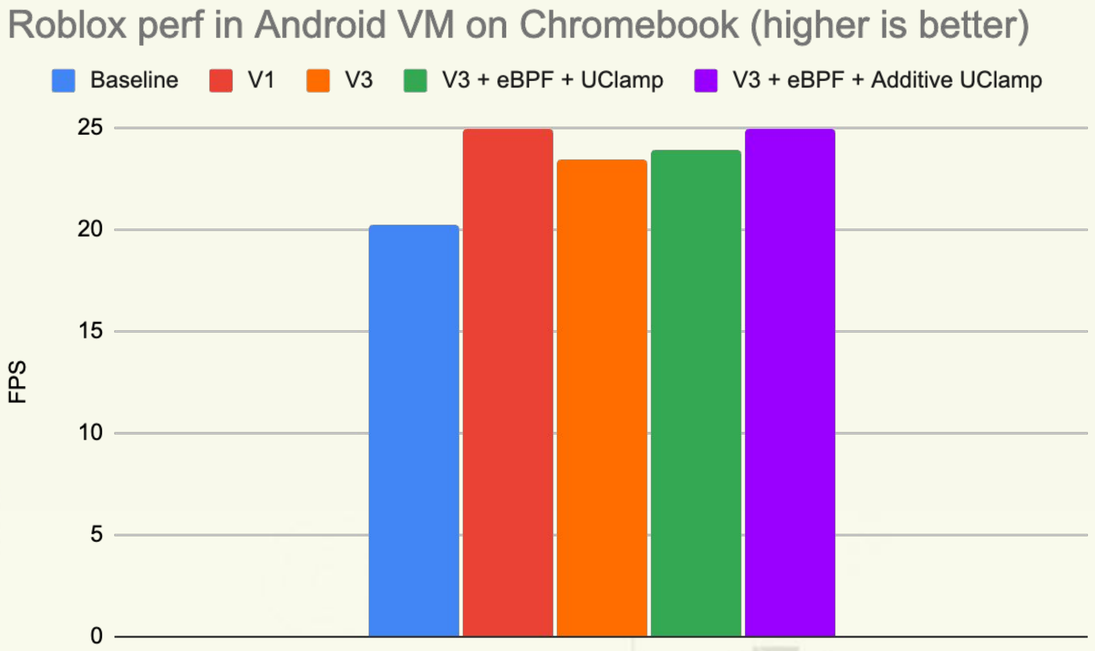
Real world results

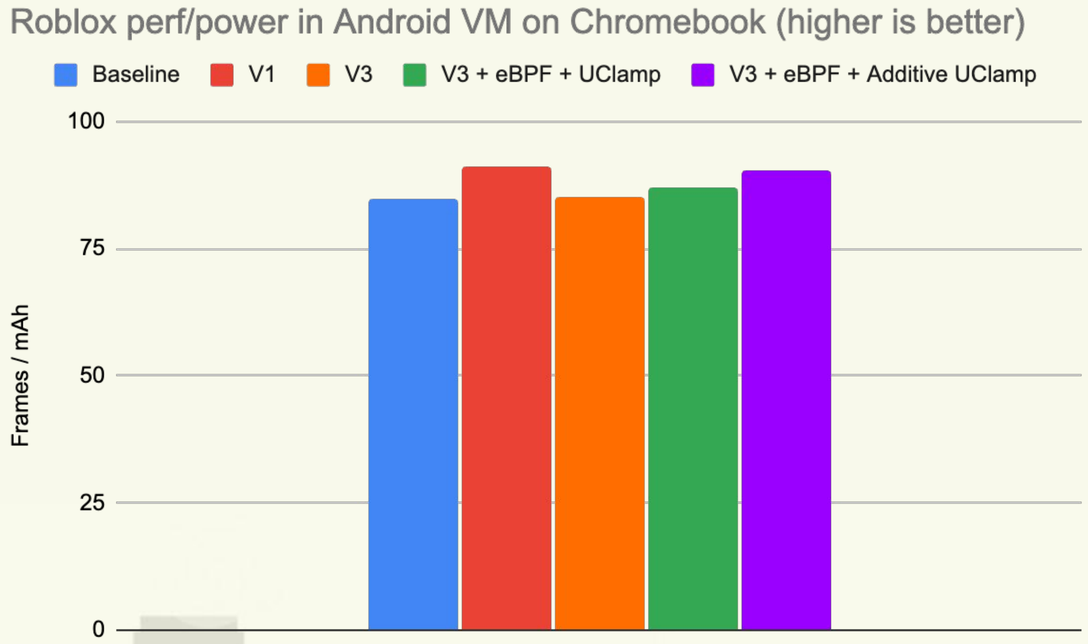
Results: PCMark



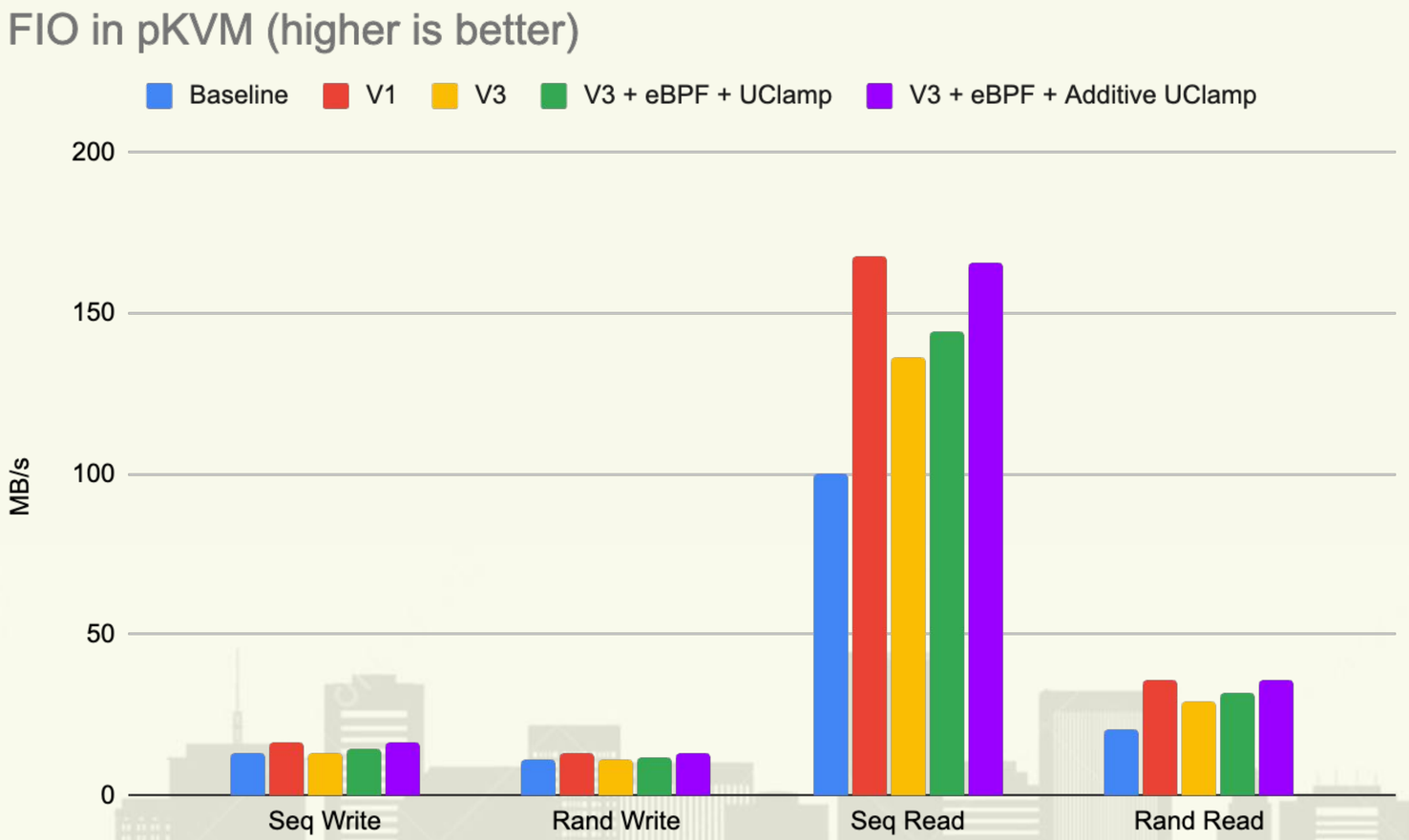


Results: Roblox game

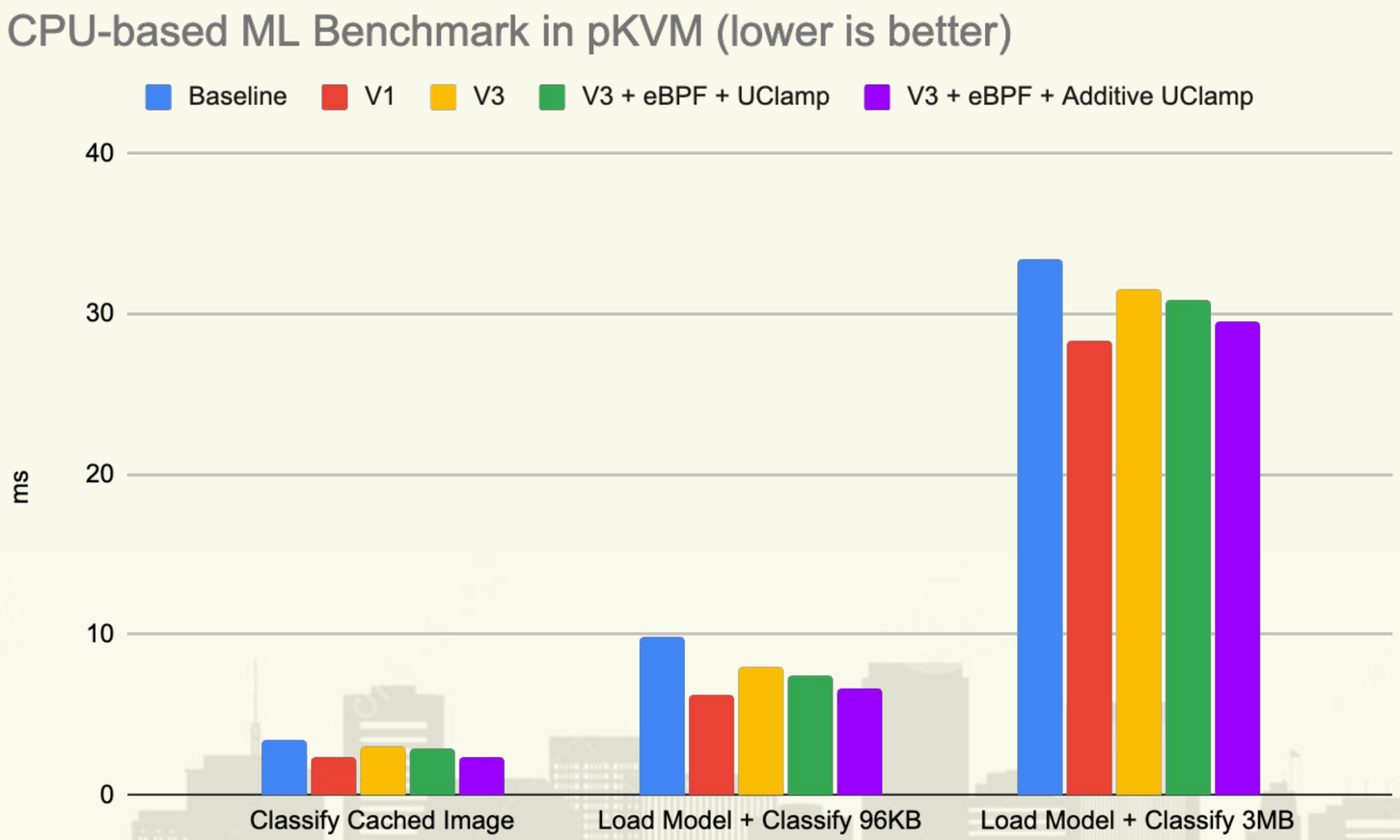




Results: FIO


FIO in pKVM (higher is better)
Baseline
V1
V3
V3 + eBPF + UClamp
V3 + eBPF + Additive UClamp
MB/s
200
150
100
50
0
Seq Write
Rand Write
Seq Read
Rand Read

Results: CPU ML workload



Discussions

Discussions

- Additive uclamp
  - Simple proposal: Add a sched attr flag that also applies uclamp min/max to task util.
- eBPF for virtual MMIO devices
  - Need basic support - Will Deacon talked about this in KVM summit
  - Need to optimize it for our needs - Call BPF programs without migration disable/enable

additive uclamp

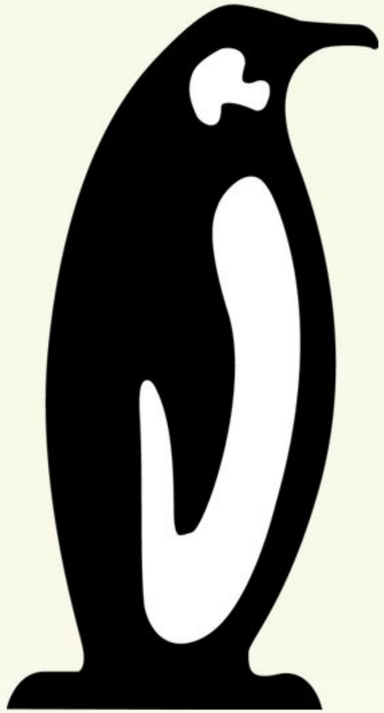
```
--- b/include/uapi/linux/sched.h
+++ b/include/uapi/linux/sched.h
 #define SCHED_FLAG_KEEP_PARAMS          0x10
 #define SCHED_FLAG_UTIL_CLAMP_MIN       0x20
 #define SCHED_FLAG_UTIL_CLAMP_MAX       0x40
+#define SCHED_FLAG_UTIL_CLAMP_TASK      0x80


--- b/kernel/sched/fair.c
+++ b/kernel/sched/fair.c

+unsigned long uclamp_task_value(struct task_struct *p, enum uclamp_id clamp_id)
+{
+       if (!p->uclamp_req[clamp_id].apply_to_task)
+               return uclamp_none(clamp_id);
+       return uclamp_eff_value(p, clamp_id);
+
+
 static inline unsigned long task_util(struct task_struct *p)
 {
-       return READ_ONCE(p->se.avg.util_avg);
+       return clamp_val(READ_ONCE(p->se.avg.util_avg),
+                        uclamp_task_value(p, UCLAMP_MIN),
+                        uclamp_task_value(p, UCLAMP_MAX));
 }

 static inline unsigned long _task_util_est(struct task_struct *p)
 {
        struct util_est ue = READ_ONCE(p->se.avg.util_est);

-       return max(ue.ewma, (ue.enqueued & ~UTIL_AVG_UNCHANGED));
+       return clamp_val(max(ue.ewma, (ue.enqueued & ~UTIL_AVG_UNCHANGED))
+                        uclamp_task_value(p, UCLAMP_MIN),
+                        uclamp_task_value(p, UCLAMP_MAX));
 }
```



Linux Plumbers Conference

Richmond, Virginia | November 13-15, 2023