VM-CPUFreq for x86

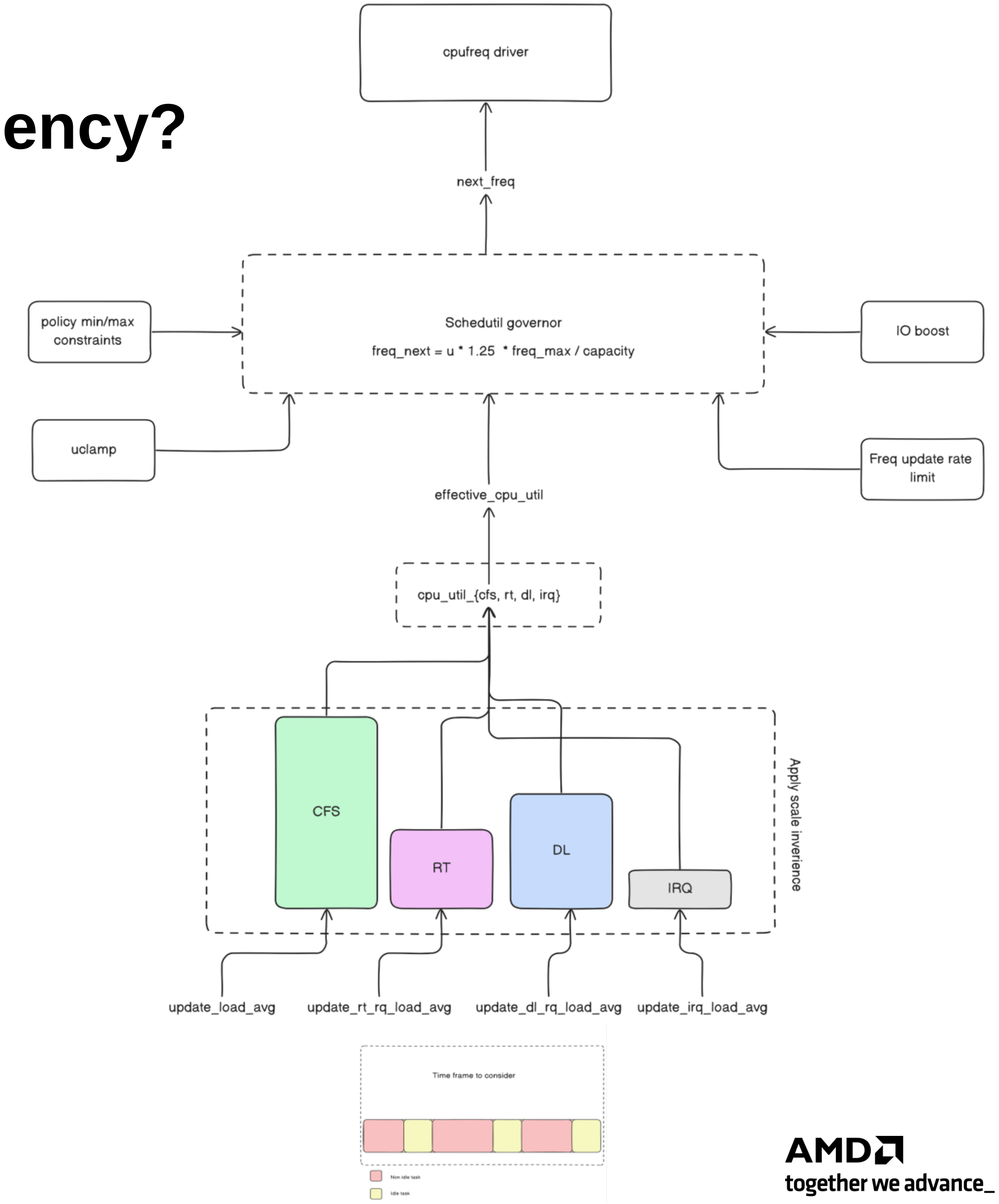
Scaling the guest frequency for performance and power savings

Wyes Karny, Gautham R. Shenoy

AMD
together we advance_

How Linux kernel scales core frequency?

- How Linux kernel decides next frequency to run?
  - Linux kernel determines the next frequency according to current utilization
  - **F_next = 1.25 * utilization * F_max / capacity**
    - Here utilization is effective utilization
    - F_max is the maximum frequency of the core
    - Capacity is the max capacity of the core
- How is effective utilization calculated?
  - Effective utilization is aggregated value of utilizations of all run queues on a CPU
- How is utilization calculated?
  - This is calculated for each task (PELT) and every task_tick:
    - How much time the task ran is the utilization of that task
  - Also, kernel adds the previous utilizations with EWMA algorithm



AMD
together we advance_

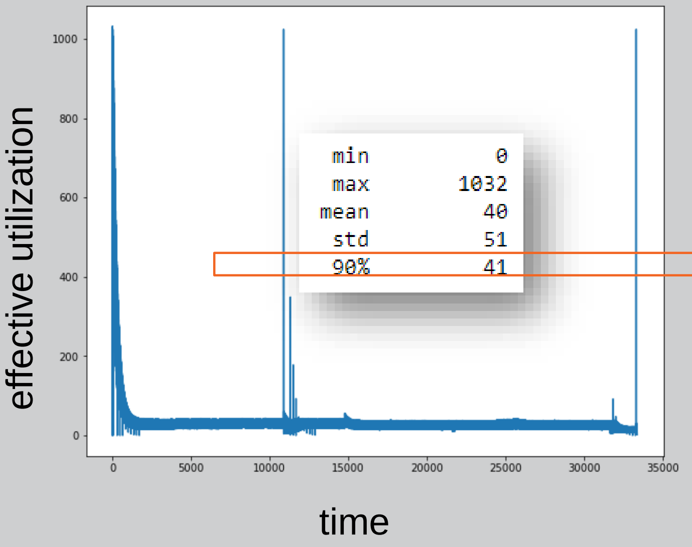
Does it work when Guest is running?

Workload:
Loop: (Busy: 0%, Idle: 100%)

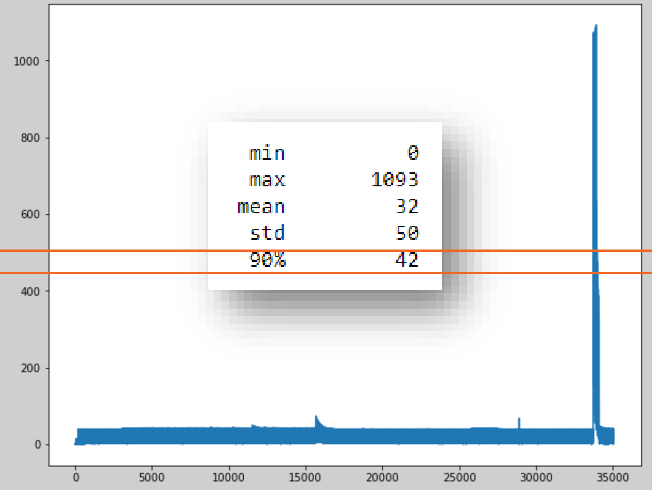
Test system:2 socket server with 4th Generation AMD EPYC™ Processors each with 128 Cores 256 Threads
Max frequency: P0: 1.9 GHz, Turbo: 3.1 GHz, Guest size: 8 vcpu
Core configuration: Pinning vcpu to pcpu, all cores are same capacity

Virtualization case

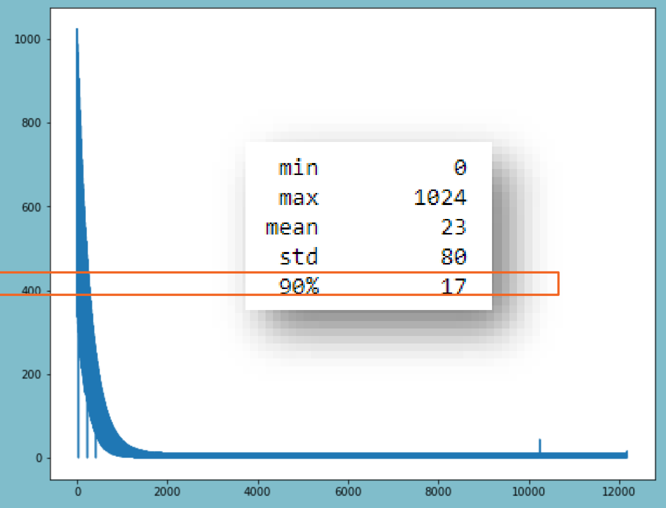
Guest's view



Host's view



BareMetal case



Note: Guest performs HLT on idle causing VM-exit

AMD together we advance_

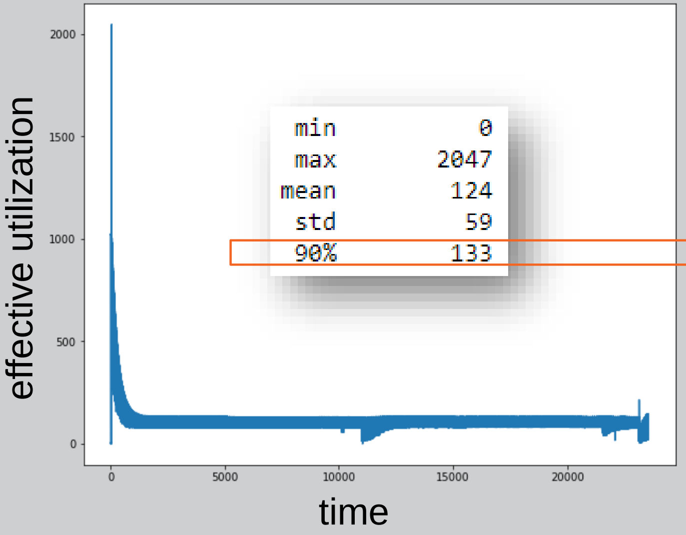
Does it work when Guest is running?

Workload:
Loop: (Busy: 10%, Idle: 90%)

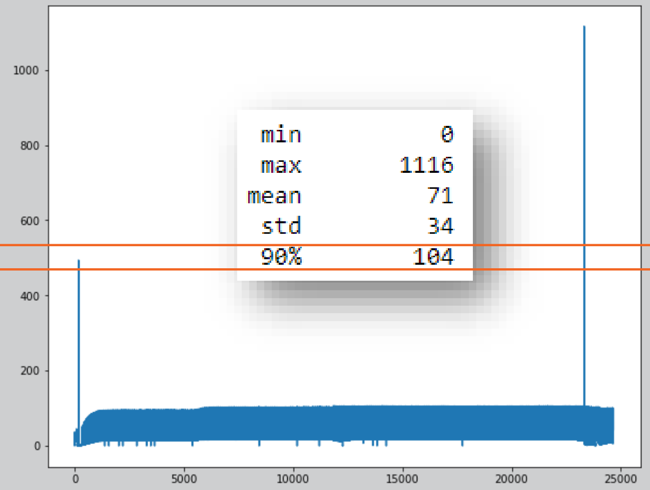
Test system:2 socket server with 4th Generation AMD EPYC™ Processors each with 128 Cores 256 Threads
Max frequency: P0: 1.9 GHz, Turbo: 3.1 GHz, Guest size: 8 vcpu
Core configuration: Pinning vcpu to pcpu, all cores are same capacity

Virtualization case

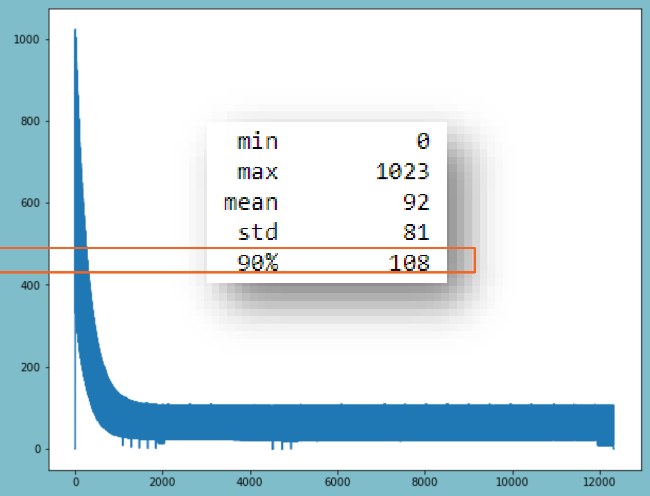
Guest's view



Host's view



BareMetal case



Note: Guest performs HLT on idle causing VM-exit

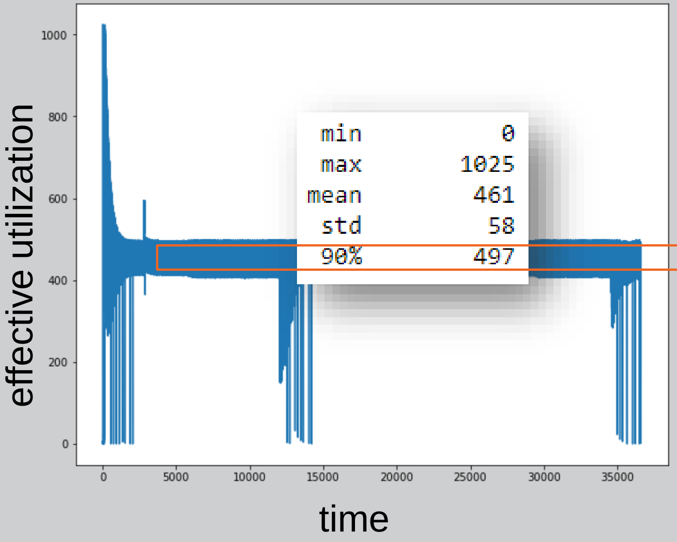
Does it work when Guest is running?

Workload:
Loop: (Busy: 50% , Idle: 50%)

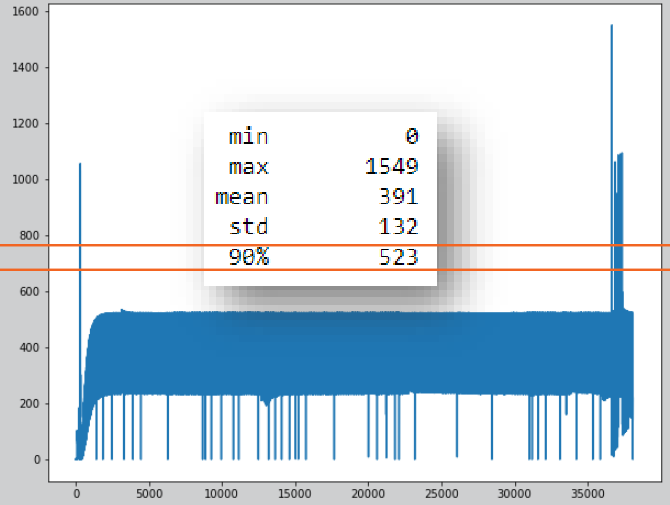
Test system:2 socket server with 4th Generation AMD EPYC™ Processors each with 128 Cores 256 Threads
Max frequency: P0: 1.9 GHz, Turbo: 3.1 GHz, Guest size: 8 vcpu
Core configuration: Pinning vcpu to pcpu, all cores are same capacity

Virtualization case

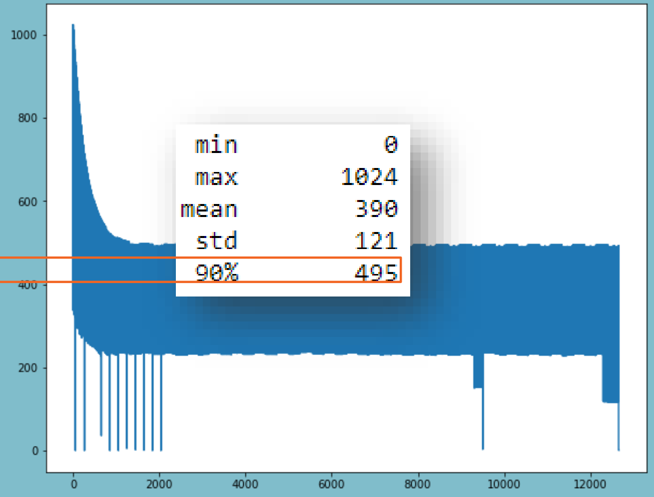
Guest's view



Host's view



BareMetal case

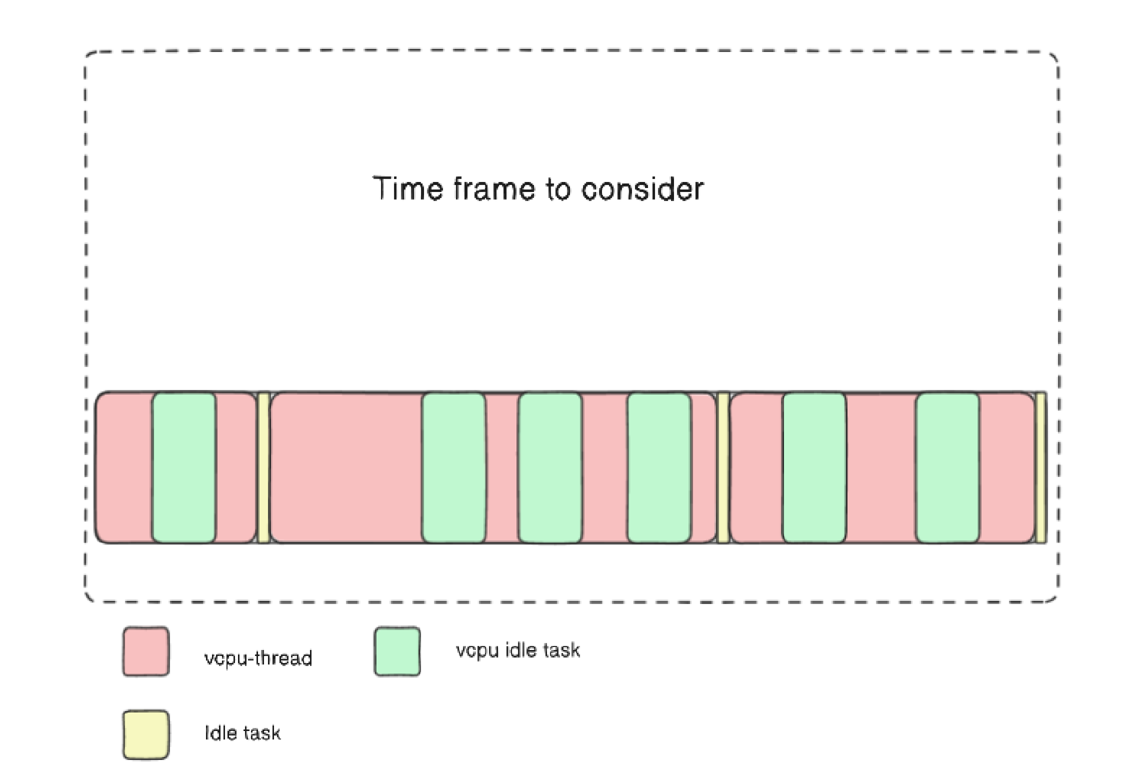


Note: Guest performs HLT on idle causing VM-exit

AMD
together we advance_

What happens when Guest does not exit when idle

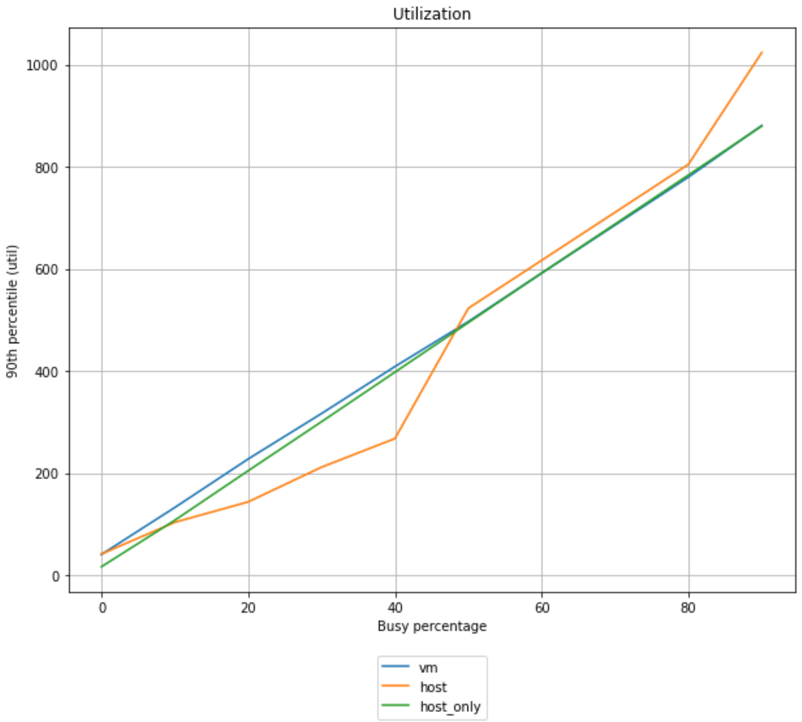
- Some **cloud providers do not want guest to perform vm-exit** when guest thread is idle
  - Reason: To achieve **lower latency**
  - Side effect: **Higher power consumption** and host's utilization calculation goes wrong
- Common approaches to avoid vm-exit
  - **Partially** avoid vm-exit by using haltpoll driver
  - **Completely** avoid vm-exit with idle=poll or idle=mwait
  - Use MWAIT inside the guest (Certain x86 processors, including AMD EPYC ones allow this)



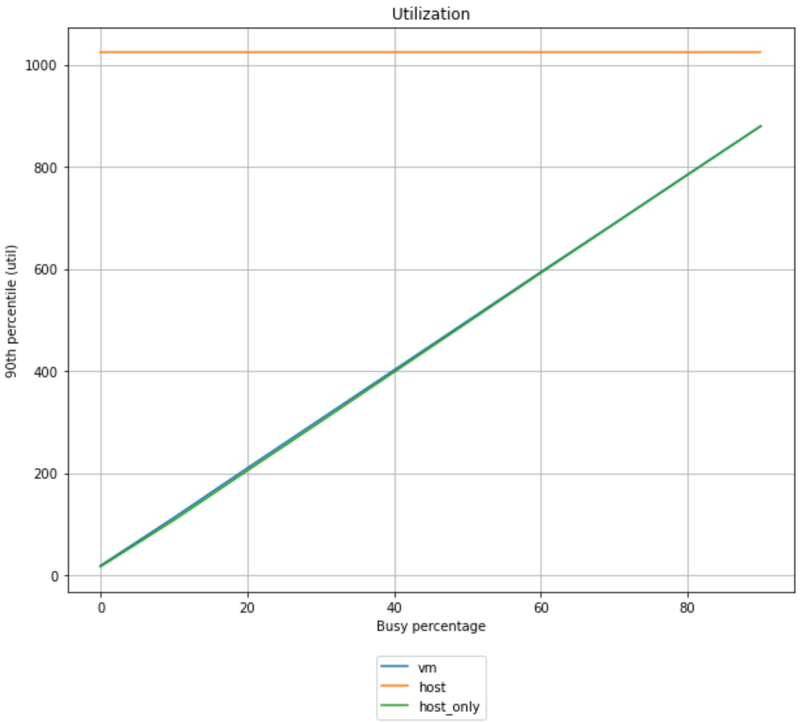
AMD
together we advance_

CPU’s util_avg When Guest is Running

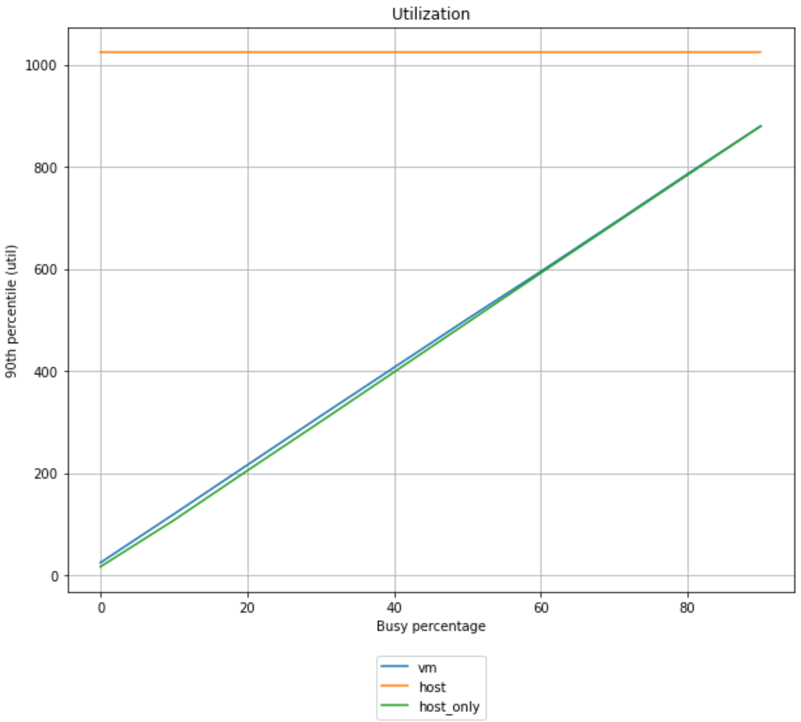
guest halt



guest poll



guest mwait



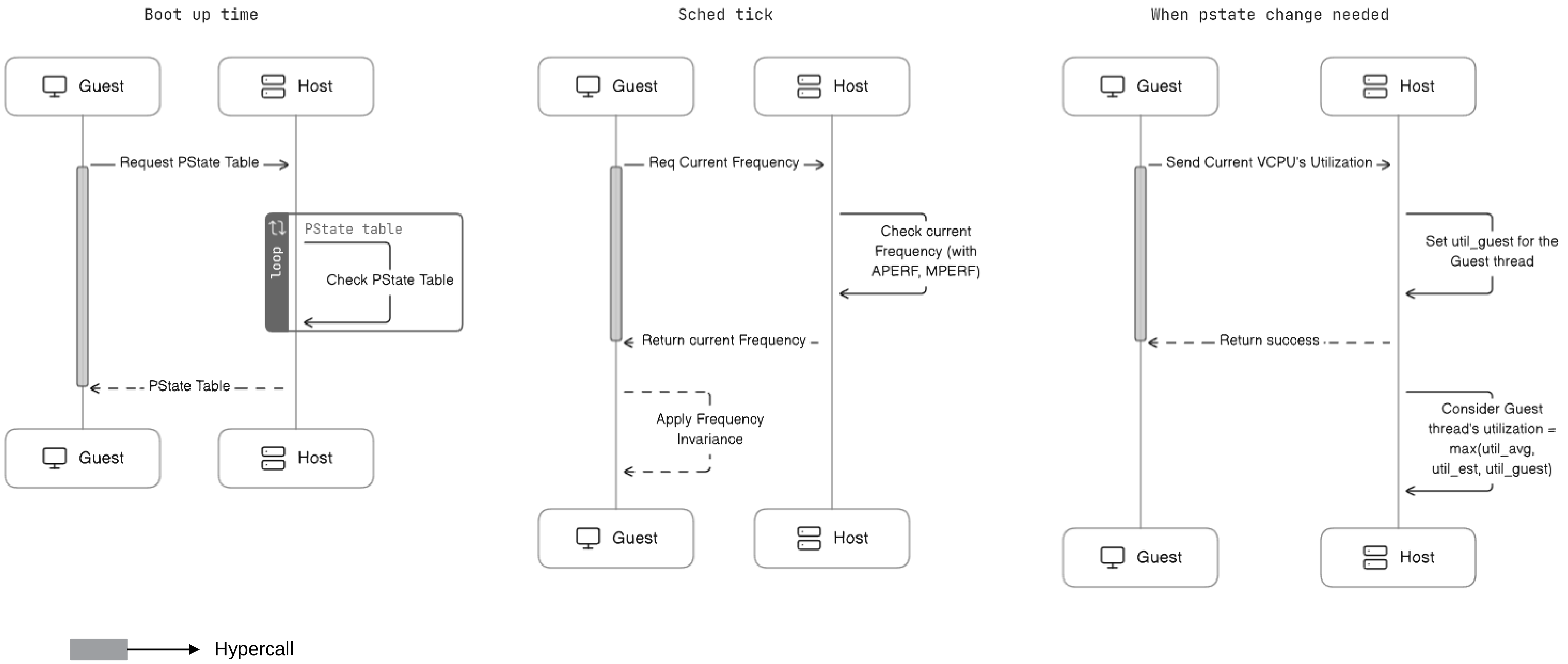
Test system:2 socket server with 4th Generation AMD EPYC™ Processors each with 128 Cores 256 Threads
Max frequency: P0: 1.9 GHz, Turbo: 3.1 GHz, Guest size: 8 vcpu
Core configuration: Pinning vcpu to pcpu, all cores are same capacity

AMD together we advance_

VM-CPUFreq Patches by Google

- Patch: "[RFC PATCH v2 0/6] Improve VM CPUfreq and task placement behavior"
  - https://lore.kernel.org/lkml/20230331014356.1033759-1-davidai@google.com/
  - Focuses on ARM big LITTLE architecture and their problem for guest workload's task placement
  - Focuses on pinned VM but VM can exit when idle.
  - Uses a hypercall to communicate the guest utilization data with the host.
- Patch: "[PATCH v3 0/2] Improve VM CPUfreq and task placement behavior"
  - https://lore.kernel.org/lkml/20230731174613.4133167-1-davidai@google.com/
  - Shares the guest utilization with the host via a MMIO mechanism.
  - VMM patches are for CrosVM
  - VMM needs to update the util_clamp to clamp utilization of the vcpu thread based on the utilization obtained via MMIO
  - No performance improvements compared to v2
- For x86 servers the problem is bit different
  - The problem is not about task placement as x86 servers have cores with same capacity
  - The problem is when guest doesn't exit on idle, host's view of vcpu thread's utilization is incorrect.
  - There is a scope of performance and power improvements
  - We choose v2 for our evaluation on x86.

How VM-CPUFreq is implemented for x86

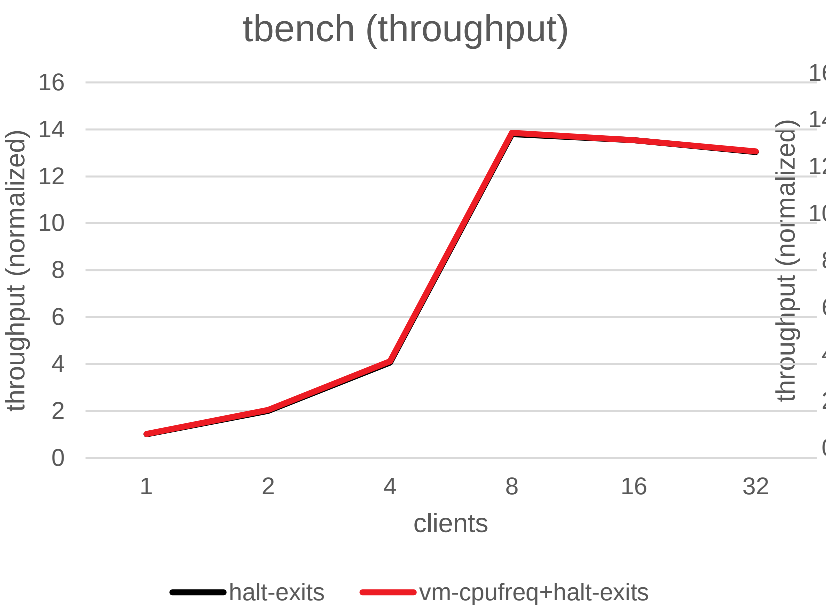


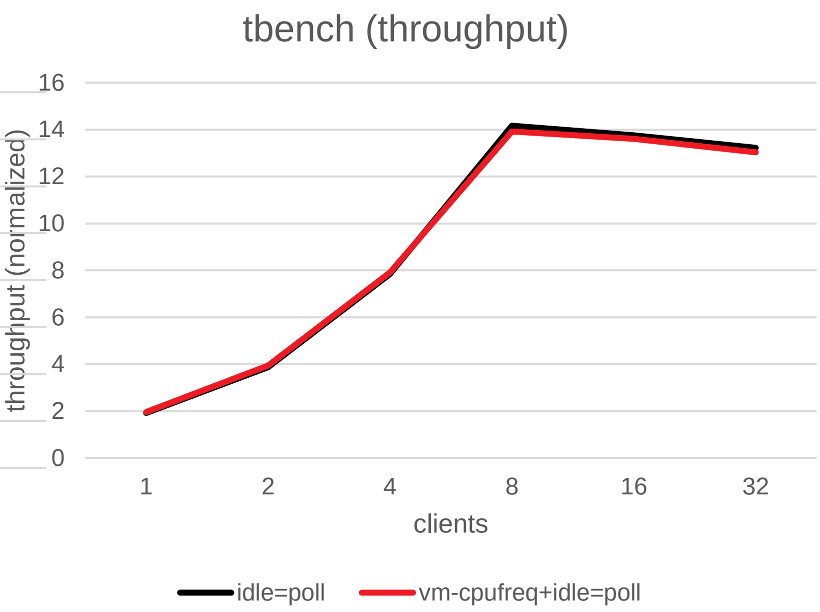
Results

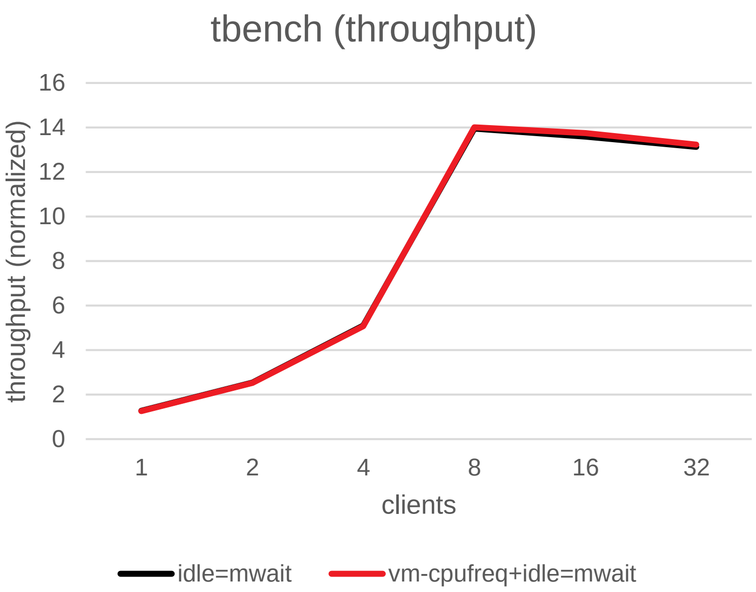
- Test setup
  - 2 socket server with 4th Generation AMD EPYC™ Processors each with 128 Cores 256 Threads
  - Max frequency: P0: 1.9 GHz, Turbo: 3.1 GHz
  - Guest size: 8 vcpu / 16 vcpu
  - Guest's vcpu threads are pinned to PCPUs.
  - Core configuration: all cores are same capacity
- Kernel versions
  - Test kernel 1: 6.6-rc6
  - Test kernel 2: 6.6-rc6 + vm-cpufreq v2 rebased and added support for x86
- Workloads

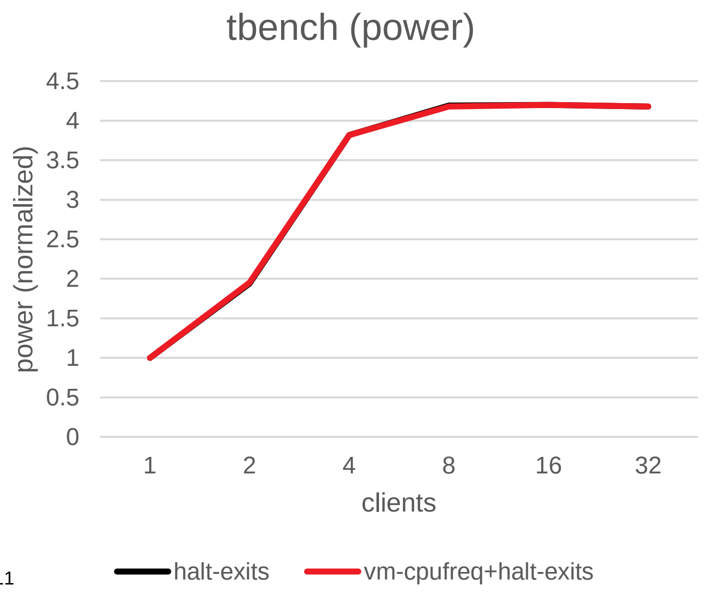
Benchmark name	License	Repository/source
tbench/dbench	GPL 3	https://www.samba.org/ftp/unpacked/dbench/
kernbench	GPL 2	https://github.com/linux-test-project/ltp/tree/master/utils/benchmark/kernbench-0.42

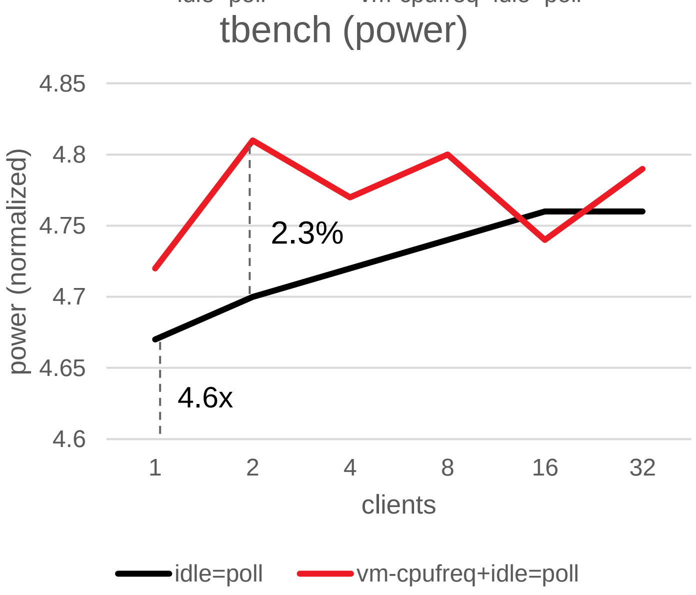
Results (8 vcpu)

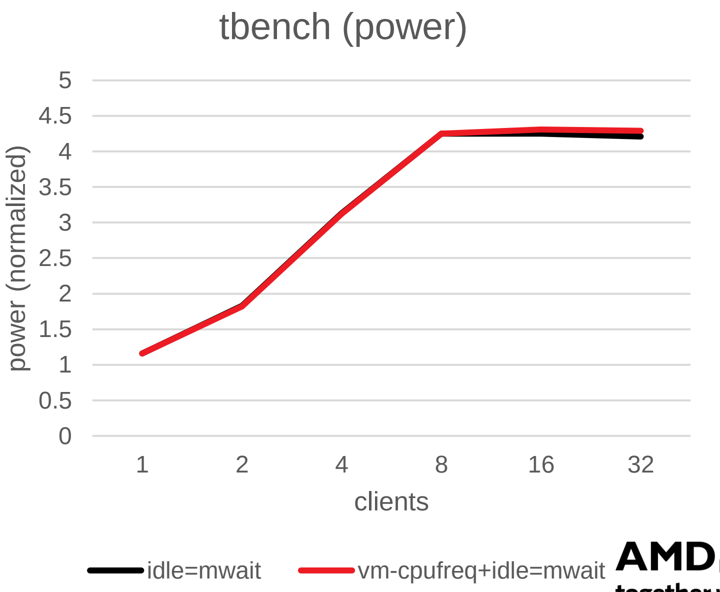




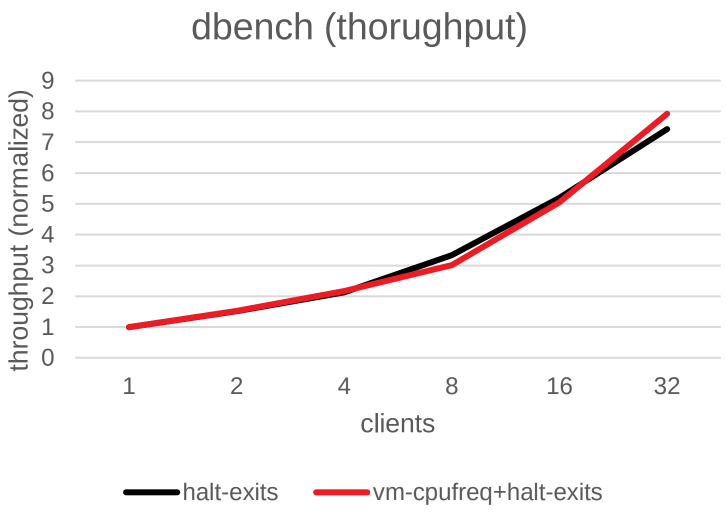


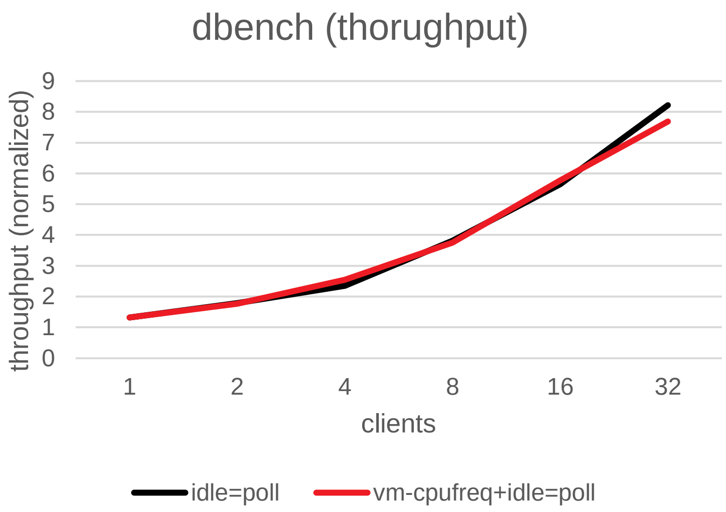


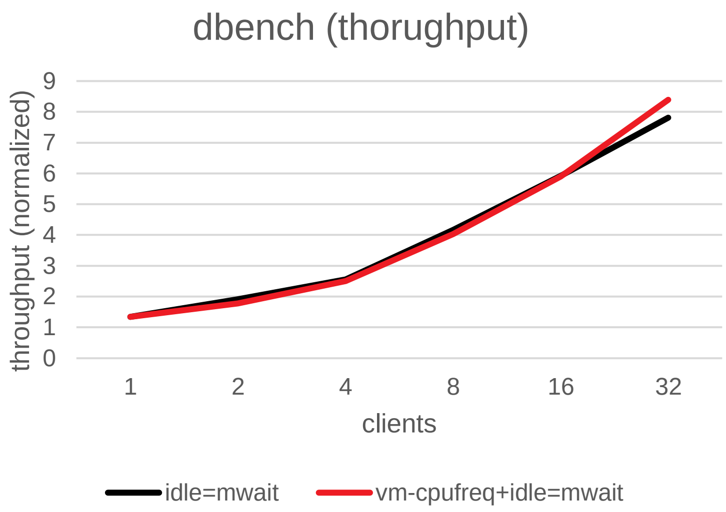


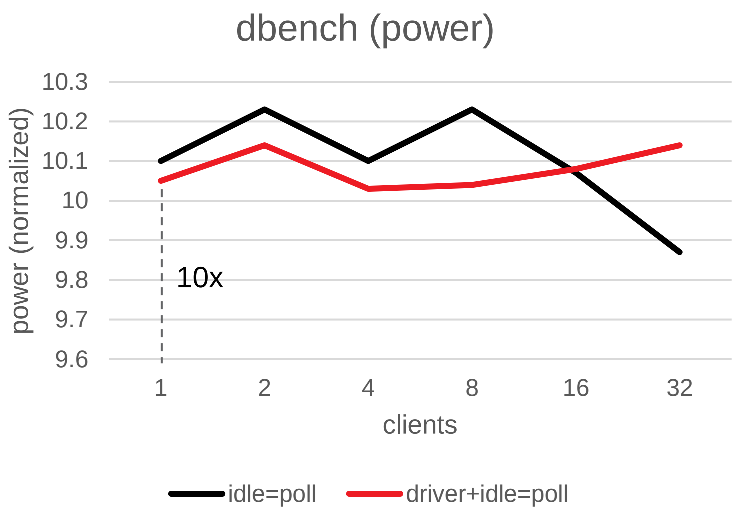


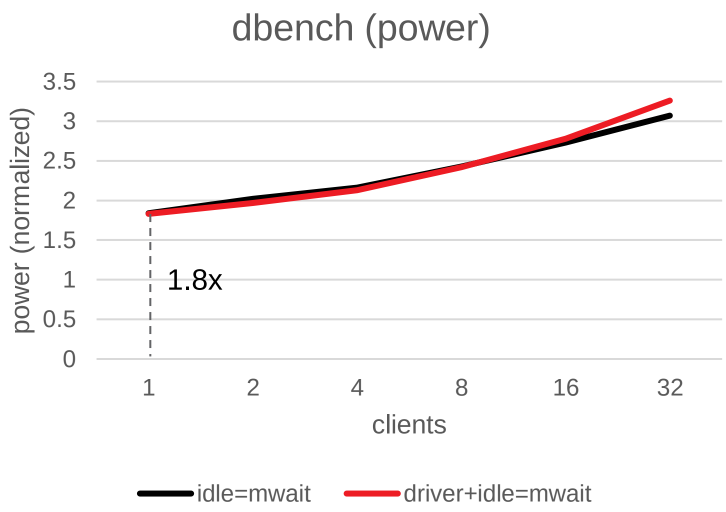
Results (8 vcpu)





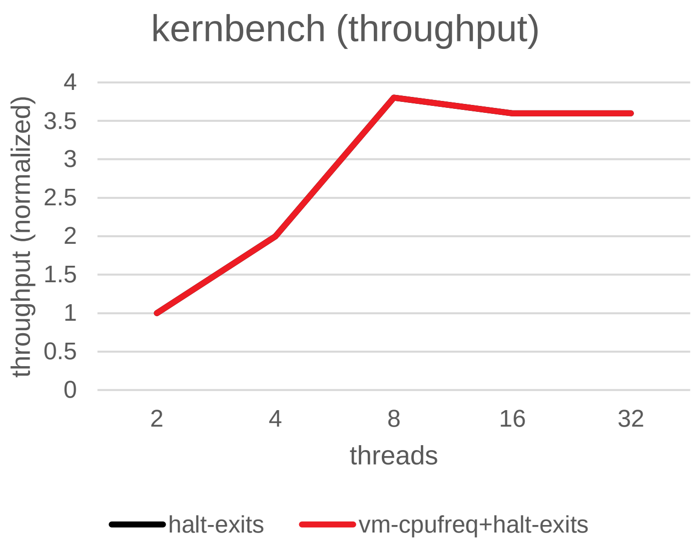


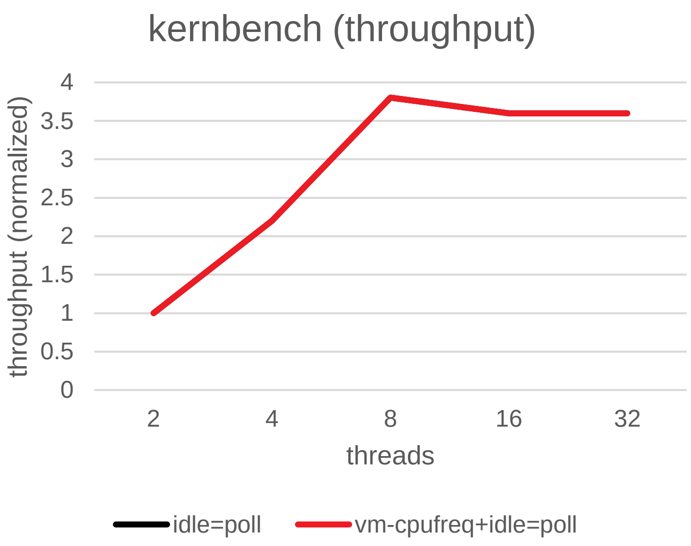


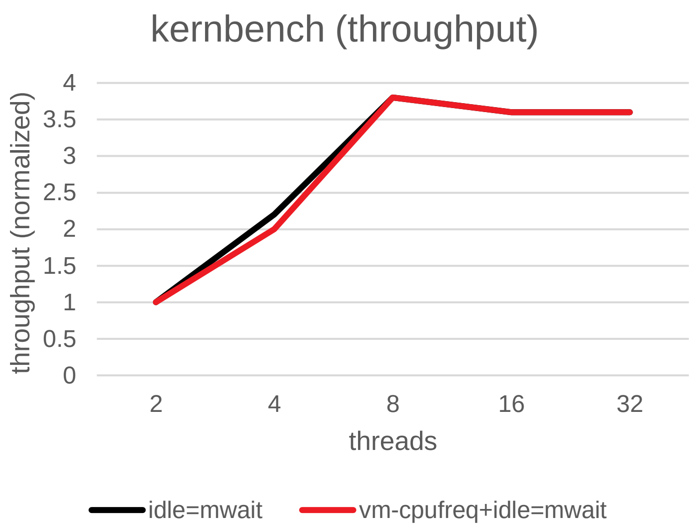


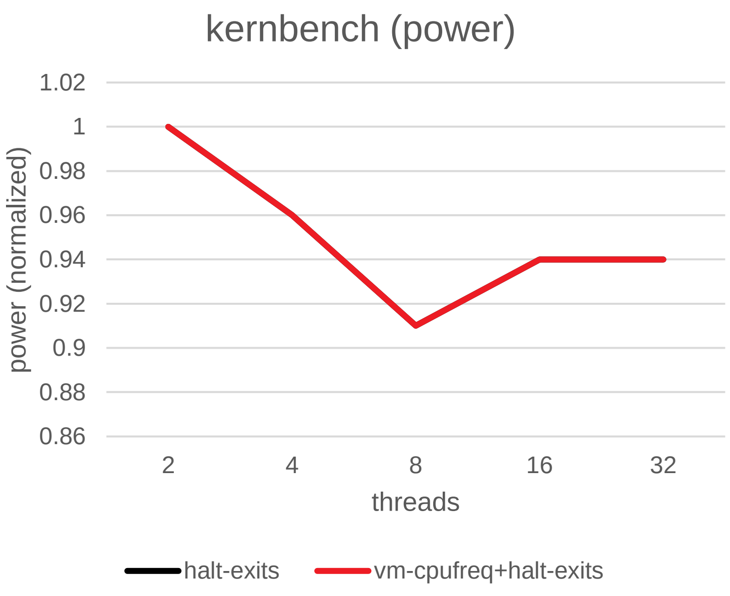
AMD
together we advance_

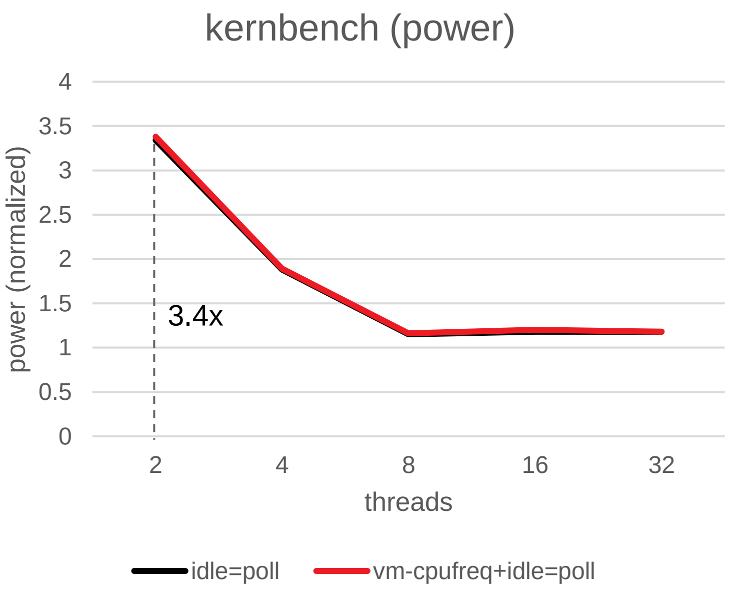
Results (8 vcpu)

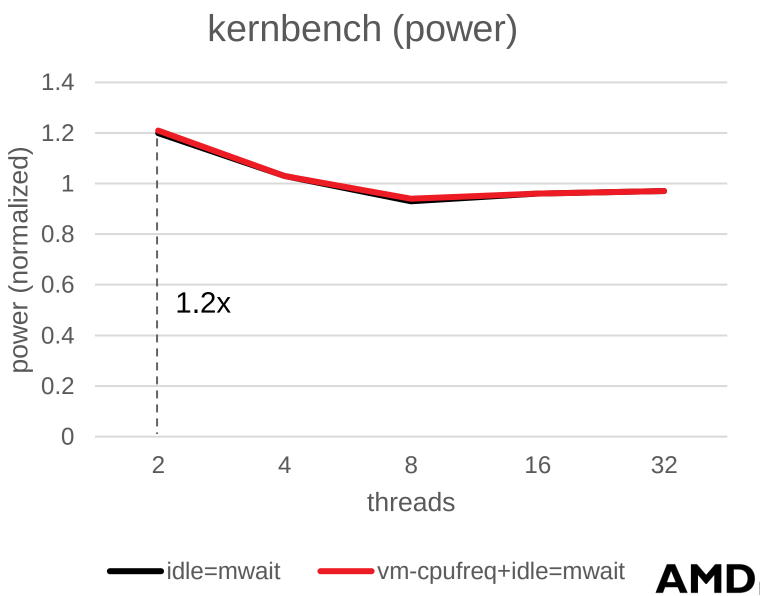




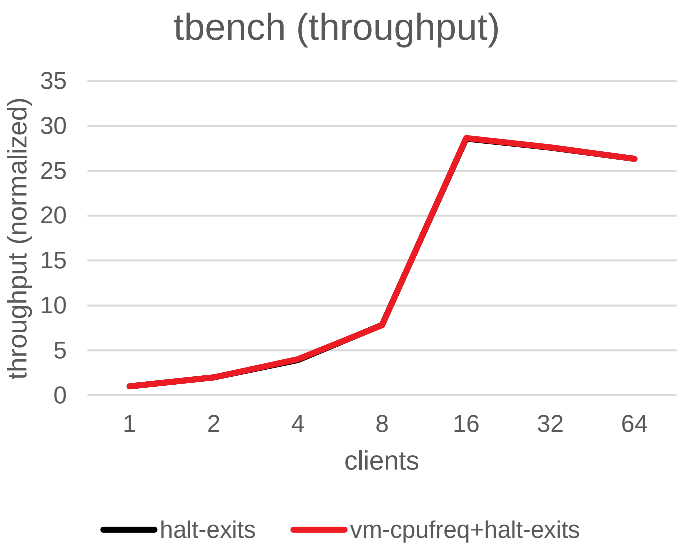


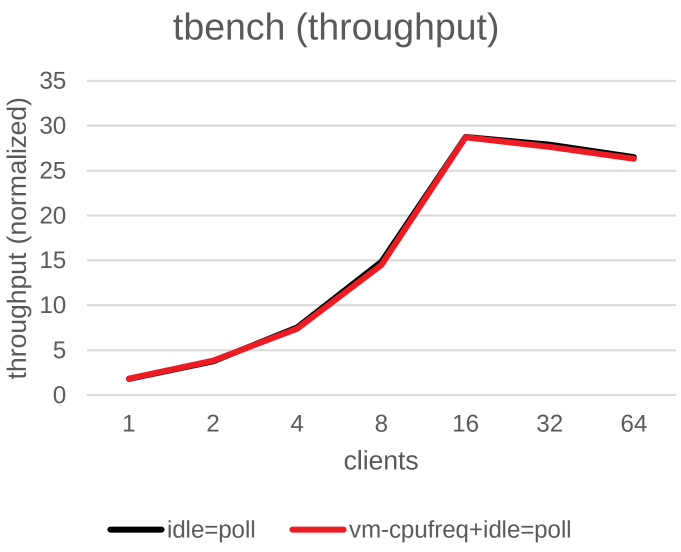


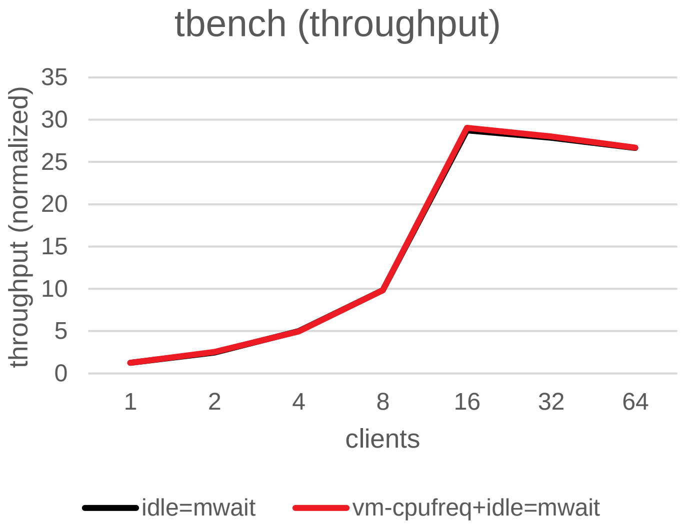


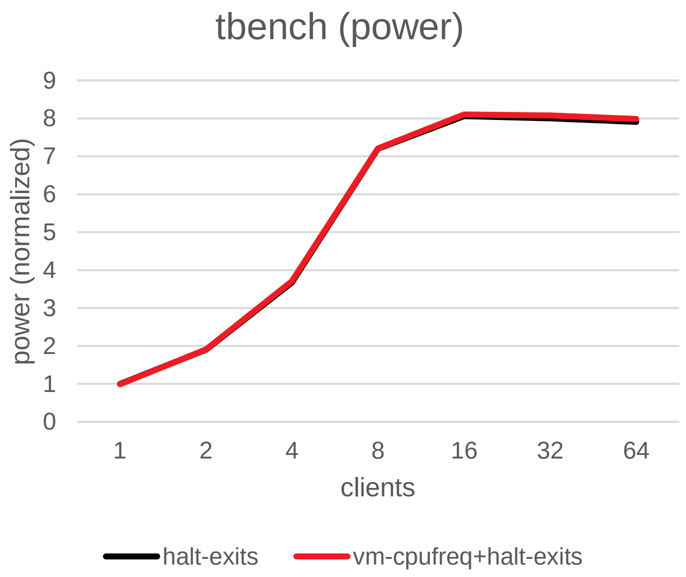


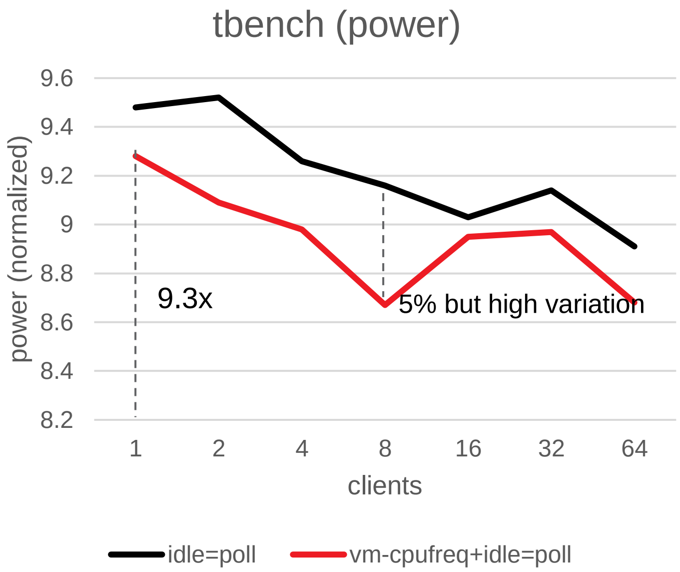
Results (16 vcpu)

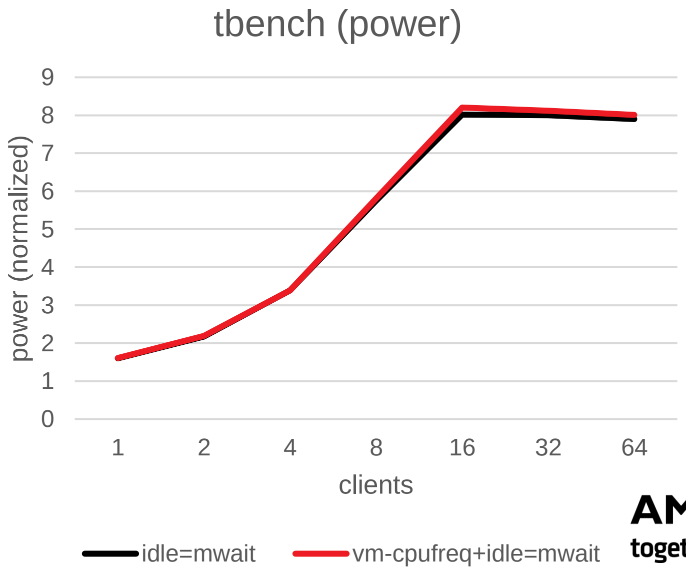




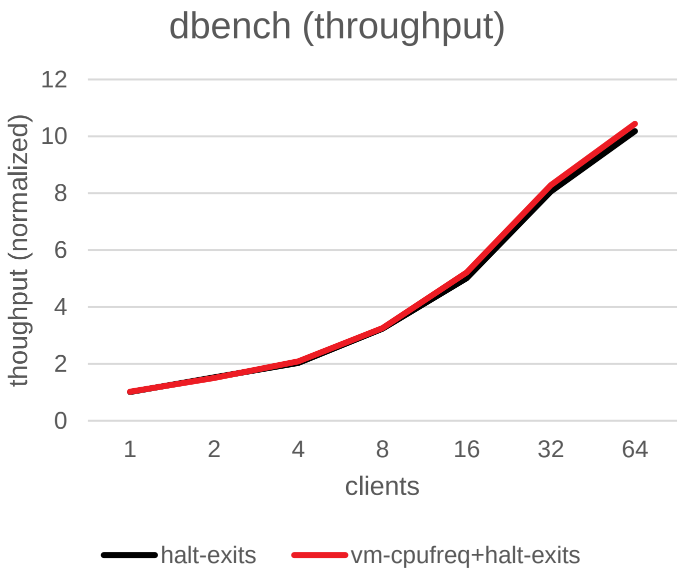


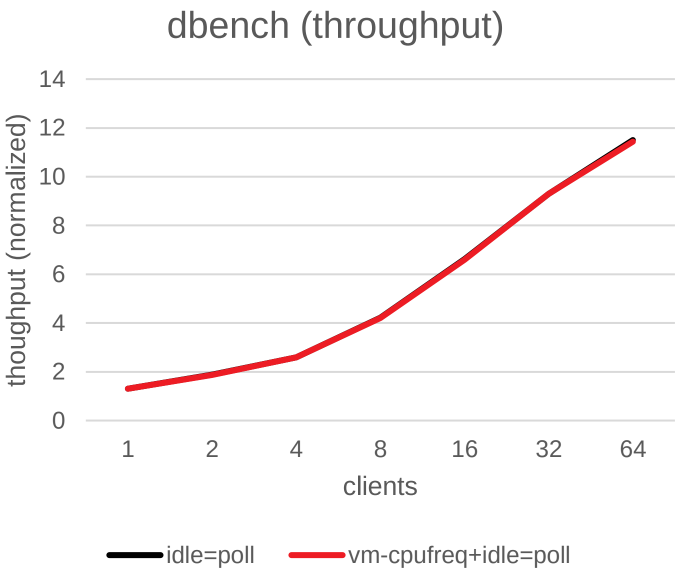


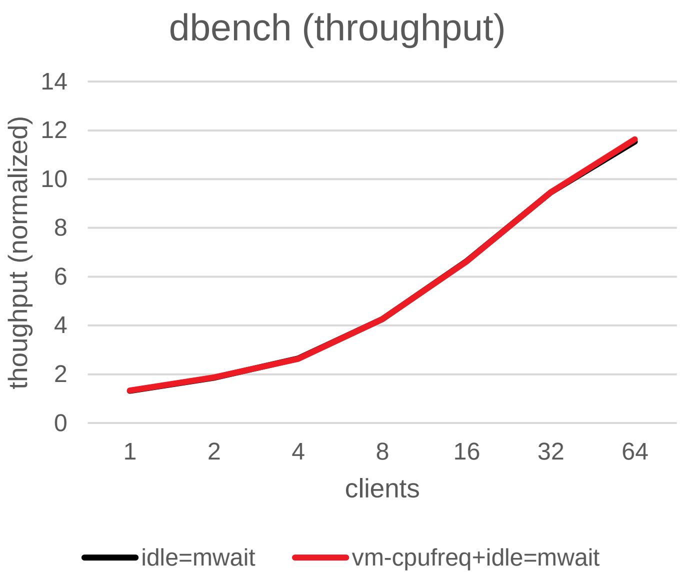


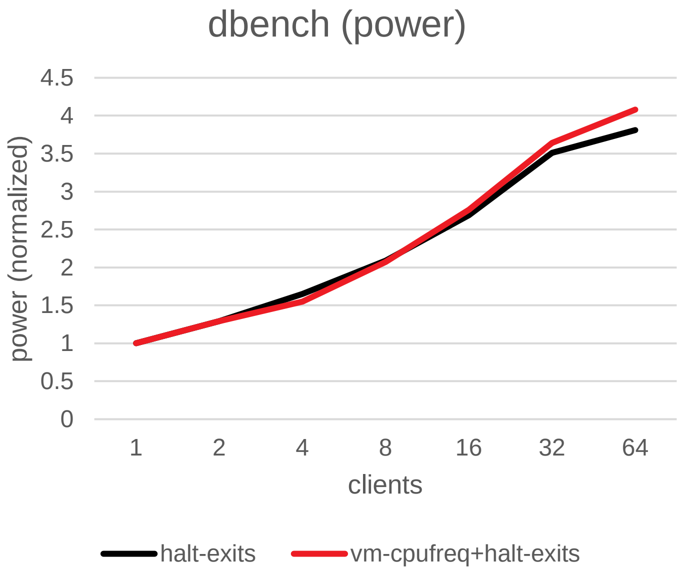


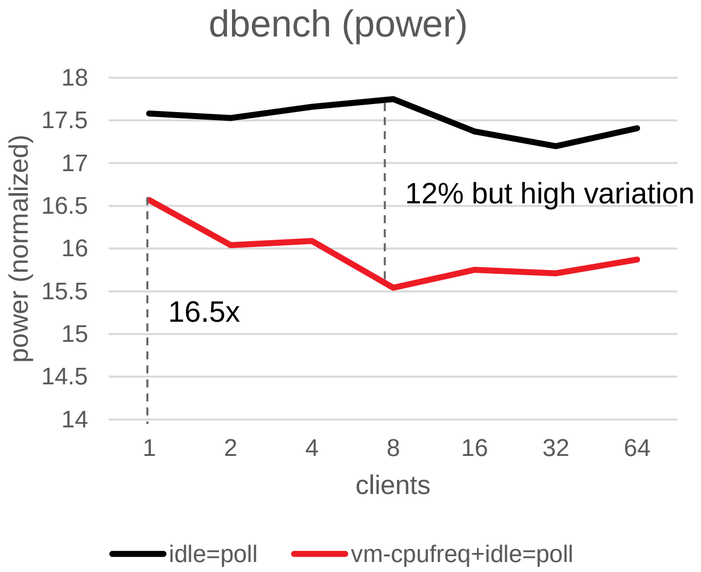
Results (16 vcpu)

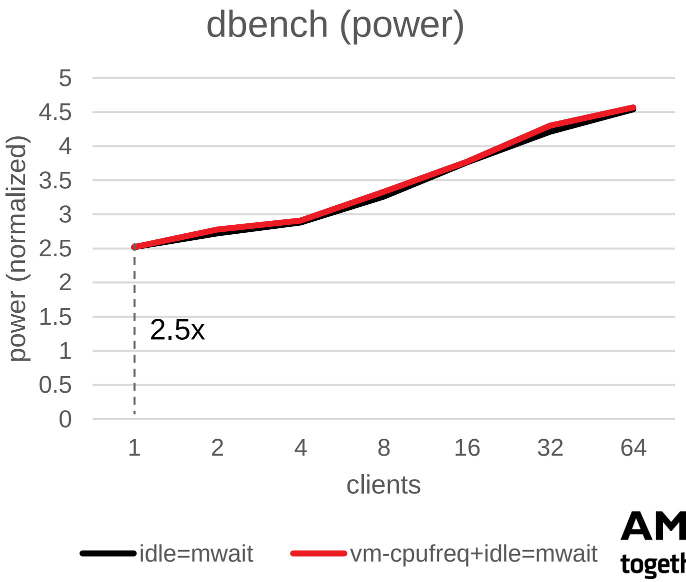






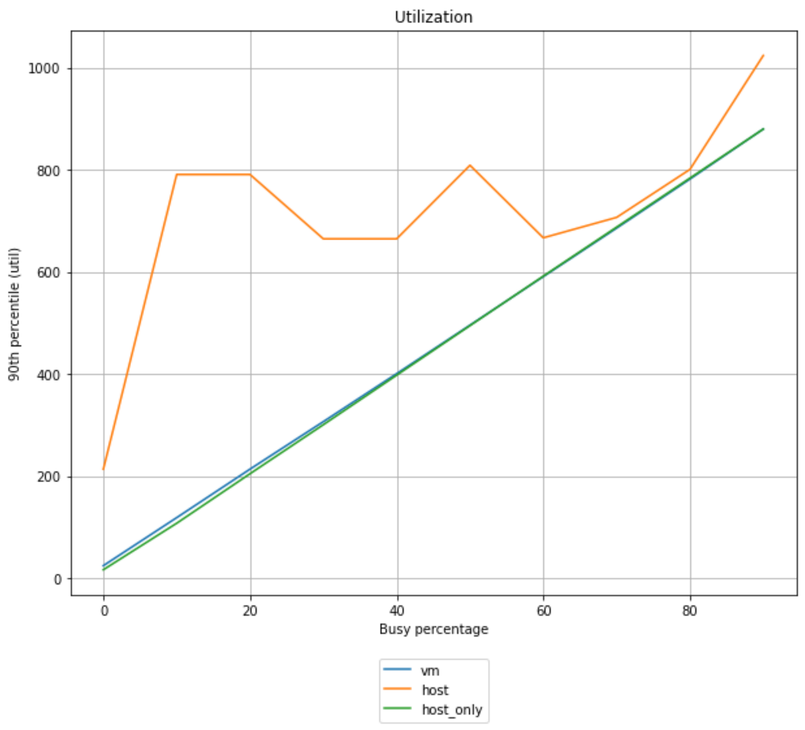




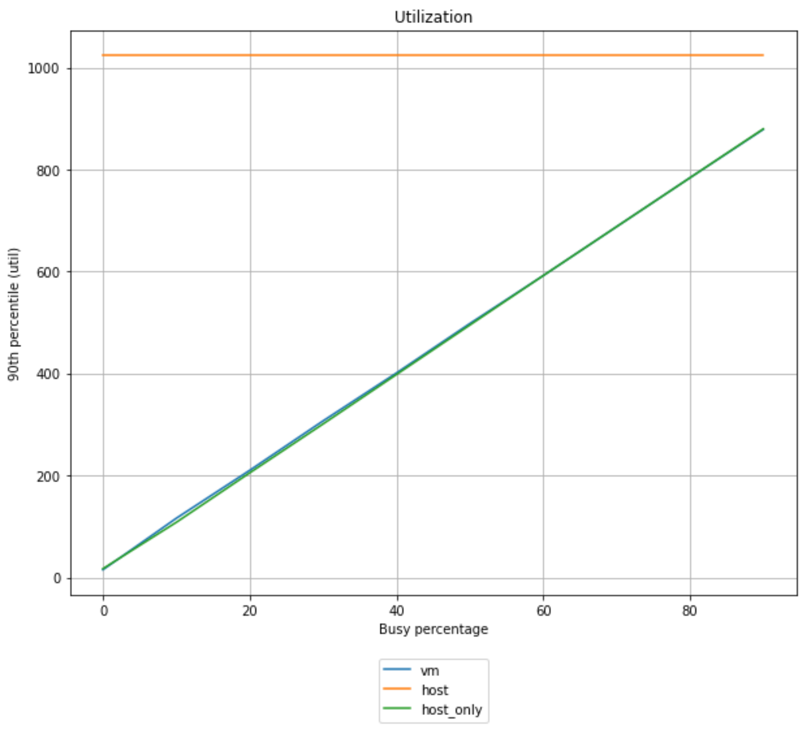


CPU's util_avg When Guest is Running with VM-CPUFreq

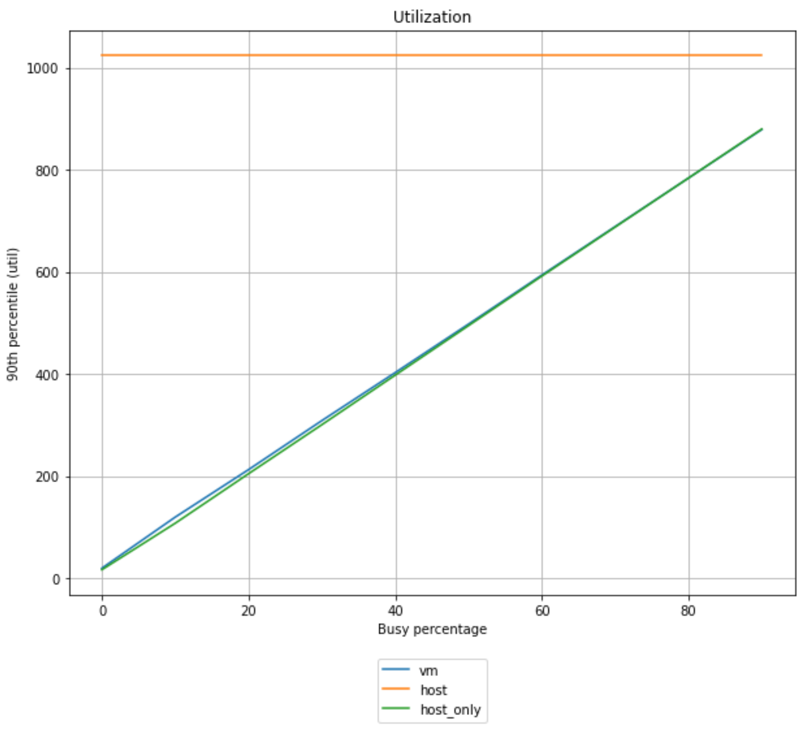
guest halt



guest poll



guest mwait



Test system:2 socket server with 4th Generation AMD EPYC™ Processors each with 128 Cores 256 Threads
Max frequency: P0: 1.9 GHz, Turbo: 3.1 GHz, Guest size: 8 vcpu
Core configuration: Pinning vcpu to pcpu, all cores are same capacity

Summary and Next steps

- Not seeing any significant performance or power improvements with vm-cpufreq.

- Current vm-cpufreq v2 patches only add lower bound for the vcpu thread utilization
  - Between the guest's utilization and the host CPU utilization, we pick the one which is higher.
  - This is not useful for saving power when the guest vCPU is POLL-idling.

- Even if we use uclamp min and max to exactly match Guest’s view of utilization to Host’s utilization, we don’t see any improvements.

COPYRIGHT AND DISCLAIMER

AMD

Backup slides

Patch to match VCPU threads utilization exactly

```
diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
index d48acf62b2d1..2d2a3bfc8282 100644
--- a/arch/x86/kvm/x86.c
+++ b/arch/x86/kvm/x86.c
@@ -9828,11 +9828,12 @@ static int kvm_sched_get_cur_cpufreq(struct kvm_vcpu *vcpu)
 static int kvm_sched_set_util(struct kvm_vcpu *vcpu, u64 val)
 {
        struct sched_attr attr = {
-               .sched_flags = SCHED_FLAG_UTIL_GUEST,
+               .sched_flags = SCHED_FLAG_UTIL_CLAMP,
        };
        int ret;

        attr.sched_util_min = val;
+       attr.sched_util_max = val;

        ret = sched_setattr_nocheck(current, &attr);
```


AMD
together we advance_