Microsoft

Linux Plumbers Conference 2024

System Boot & Security Micro-Conference

Vienna, Austria - Sep 18, 2024

Accelerating Linux Kernel Boot-Up for Large Multi-Core Systems

Saurabh Singh Sengar ssengar@microsoft.com
Principal Software Engineer

Srivatsa Bhat srivatsa@csail.mit.edu
Principal Software Engineer

Microsoft Linux Systems Group

Outline

- Problem of boot time increase rapidly with increase in CPUs and NUMA nodes
- Linux Kernel Boot Phases and SMP initialization
- Overall boot optimization and future proposals

Problem

Linux boot time increases rapidly with increase in CPUs and NUMA nodes

For a 1780 CPUs machine with 16 NUMA nodes, booting takes about 45 seconds.

A similar system with 32 CPUs boots up in less than a second.

# CPU	# NUMA	BootTime (Sec)
32	1	0.5
960	15	11
1024	16	15
1780	16	45

Parallel bring-up of secondary CPU

- A “parallel CPU bringup” patch series was upstreamed in 6.5 kernel, but didn’t help Hyper-V VMs.

- Claims to save at most 720 ms for 112 CPU machines.

- Appears to be more useful for baremetal.

- Unlike the name suggests its not completely parallel … ☹

- It parallelized the sequence where BP waits for AP alive status after “Kick AP alive”. [3]

Linux Kernel Boot Phases

As per the time taken, Linux kernel boot time can be divided into 4 stages:

1. Boot Processor (BP) bring up
2. Symmetric Multi-Processor initialization (SMP init)
3. "Scheduler Domain" initialization (build sched domains)
4. Various init level calls (Init_calls).

Linux kernel boot phases	Time(Sec)
BP	1
SMP init	25
build sched domains	10
initcalls	9.5

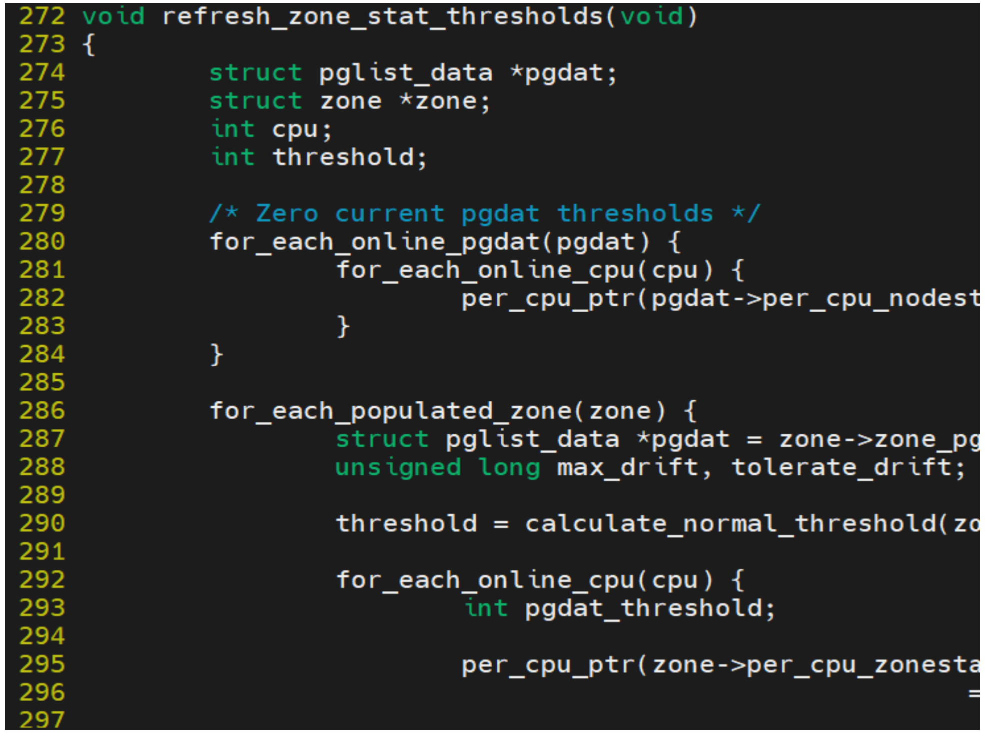
SMP initialization

- Major task is to invoke callbacks for each of states for each of CPU.
- Callbacks are serial for each CPU. (237 callbacks)
- vmstat callback which does the repeated calculation (*refresh_zone_stat_thresholds*) takes the most time out of all.

SMP init boot-speed improvement with vmstat fix

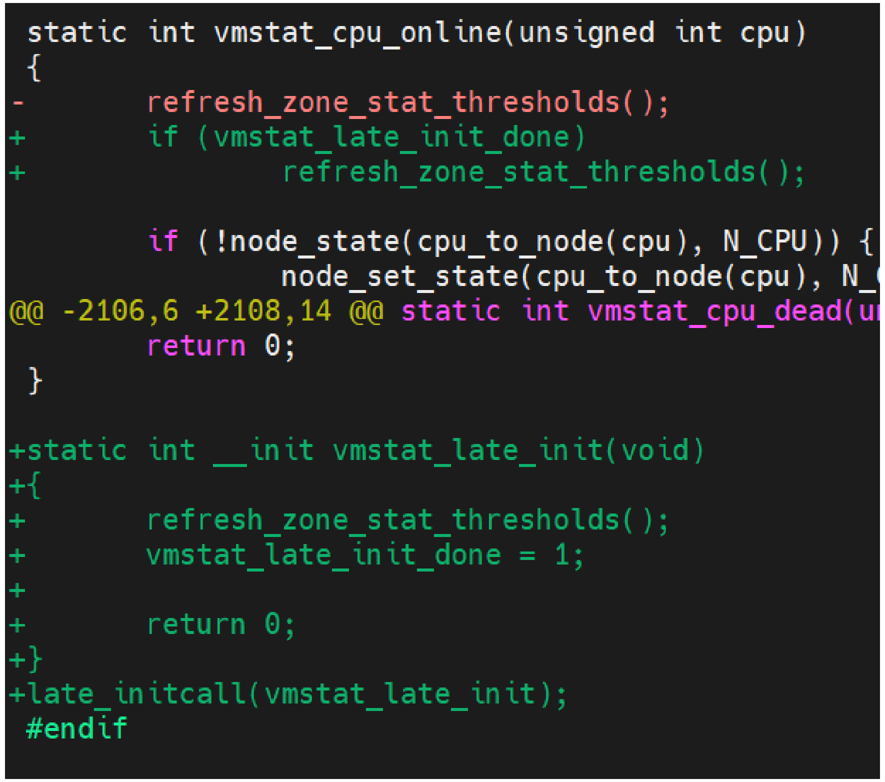
Phase	Notes	Further breakup	Time before (Sec)	Time after (Sec)	Savings (Sec)
SMP init	CPU callbacks	#198, workqueue_online_cpu	5	5	**+ 14**
		#154, hv_cpu_init	1.5	1.5	
		#204, vmstat	14	0	
		Others	4.5	4.5	
			25	11	

vmstat fix



```
void refresh_zone_stat_thresholds(void)
{
        struct pglist_data *pgdat;
        struct zone *zone;
        int cpu;
        int threshold;

        /* Zero current pgdat thresholds */
        for_each_online_pgdat(pgdat) {
                for_each_online_cpu(cpu) {
                        per_cpu_ptr(pgdat->per_cpu_nodest
                }
        }

        for_each_populated_zone(zone) {
                struct pglist_data *pgdat = zone->zone_pg
                unsigned long max_drift, tolerate_drift;

                threshold = calculate_normal_threshold(zo

                for_each_online_cpu(cpu) {
                        int pgdat_threshold;

                        per_cpu_ptr(zone->per_cpu_zonesta
                                                         =

```

refresh_zone_stat_threshold function



```
 static int vmstat_cpu_online(unsigned int cpu)
 {
-        refresh_zone_stat_thresholds();
+        if (vmstat_late_init_done)
+                refresh_zone_stat_thresholds();

         if (!node_state(cpu_to_node(cpu), N_CPU)) {
                 node_set_state(cpu_to_node(cpu), N_(
@@ -2106,6 +2108,14 @@ static int vmstat_cpu_dead(u
         return 0;
 }

+static int __init vmstat_late_init(void)
+{
+        refresh_zone_stat_thresholds();
+        vmstat_late_init_done = 1;
+
+        return 0;
+}
+late_initcall(vmstat_late_init);
 #endif
```

Potential fix

*Total calculations = "number of CPU" * 2 * Numa * "mean CPU count"*

eg:

*1 NUMA, 32 CPU = 32*2*1*16 = **1024** loop iterations*

*16 NUMA, 1780 CPU = 1780*2*16*890 = **50 million** loop iterations.*

- ➢Lot of this calculation is throwaway and serial. Can possibly avoid it ?

- ➢Fix for vmstat based on above idea under review [1].

“Scheduler Domain” initialization

- topology_span_sane() checks that each processor's non-NUMA scheduling domains are completely equal.
- Sufficient confidence on the topology can help taking the risk of avoiding this sanity check.
- Can move this function as debug option (SCHED_DEBUG/sched_verbose).

sched-domain init boot-speed improvement without topology sanity check

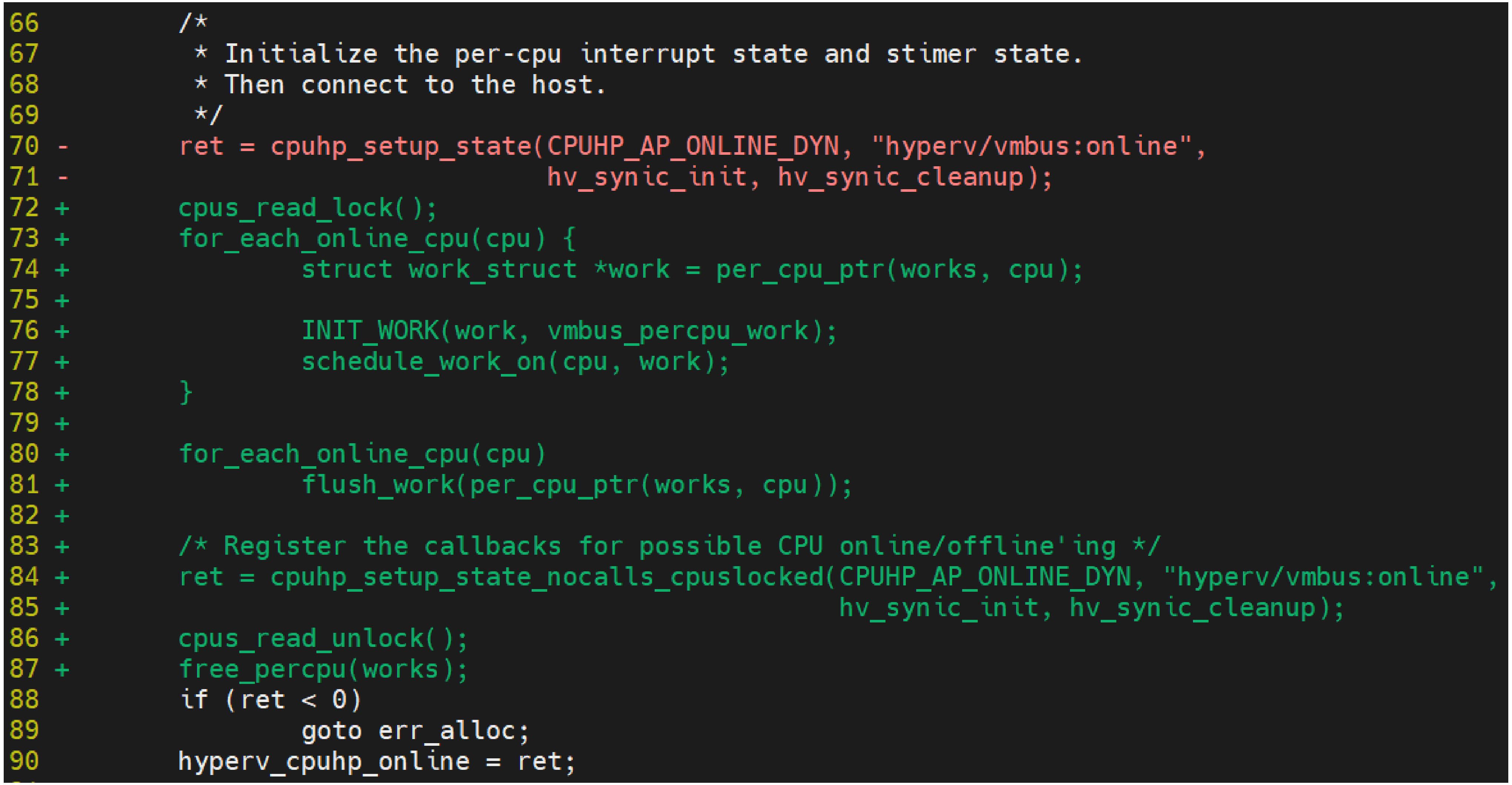
Phase	Further breakup	Time before (Sec)	Time after (Sec)	Savings (Sec)
sched_init_domains	topology_span_sane	8	0	**+ 8**

Init level calls

- All the CPUs are online at this stage.
- Callbacks are serial for each CPU.
- VMBus initcall took 3 seconds.
- Implemented workqueue to enable parallel scheduling of these callbacks.
- Saved 2 seconds, compared to standard CPU callback APIs (*cpuhp_setup_state_nocalls_cpuslocked*)
- Patch has been accepted upstream [2]

Phase	Notes	Further breakup	Time before (Sec)	Time after (Sec)	Savings in seconds
Initcalls	Various initcalls	VMBus init	3	1	+2
		msr_init	1.7	1.7	
		percpu_counter_startup	1	1	
		cacheinfo_sysfs_init	2	2	
		mshv_vtl_init	1.7	1.7	
			9.4	7.4	

VMBus init optimization fix



```
          /*
           * Initialize the per-cpu interrupt state and stimer state.
           * Then connect to the host.
           */
-         ret = cpuhp_setup_state(CPUHP_AP_ONLINE_DYN, "hyperv/vmbus:online",
-                                 hv_synic_init, hv_synic_cleanup);
+         cpus_read_lock();
+         for_each_online_cpu(cpu) {
+                 struct work_struct *work = per_cpu_ptr(works, cpu);
+
+                 INIT_WORK(work, vmbus_percpu_work);
+                 schedule_work_on(cpu, work);
+         }
+
+         for_each_online_cpu(cpu)
+                 flush_work(per_cpu_ptr(works, cpu));
+
+         /* Register the callbacks for possible CPU online/offline'ing */
+         ret = cpuhp_setup_state_nocalls_cpuslocked(CPUHP_AP_ONLINE_DYN, "hyperv/vmbus:online",
+                                                    hv_synic_init, hv_synic_cleanup);
+         cpus_read_unlock();
+         free_percpu(works);
          if (ret < 0)
                  goto err_alloc;
          hyperv_cpuhp_online = ret;
```

Overall boot optimization

Phase	Notes	Further breakup	Time before (Sec)	Time after (Sec)	Savings (Sec)
BP		BP time	1	1	
SMP init	CPU callbacks	#198, workqueue_online_cpu	5	5	+ 14
		#154, hv_cpu_init	1.5	1.5	
		vmstat	14	0	
		Others	4.5	4.5	
sched_init_domains	Creating CPU schedule domain	sched_init_domains	10	2	+ 8
Initcalls	Various initcalls	VMBus init	3	1	+2
		msr_init	1.7	1.7	
		percpu_counter_startup	1	1	
		cacheinfo_sysfs_init	2	2	
		mshv_vtl_init	1.7	1.7	
			45.4	21.4	

Overall boot optimization

Phase	Notes	Further breakup	Time before (Sec)	Time after (Sec)	Savings (Sec)
BP		BP time	1	1	
SMP init	CPU callbacks	#198, workqueue_online_cpu	5	5	+ 14
		#154, hv_cpu_init	1.5	1.5	
		vmstat	14	0	
		Others	4.5	4.5	
sched_init_domains	Creating CPU schedule domain	sched_init_domains	10	2	+ 8
Initcalls	Various initcalls	VMBus init	3	1	+2
		msr_init	1.7	1.7	
		percpu_counter_startup	1	1	
		cacheinfo_sysfs_init	2	2	
		mshv_vtl_init	1.7	1.7	
			45.4	21.4	

Future enhancements

Phase	Notes	Further breakup	Time before (Sec)	Time after (Sec)	Savings (Sec)
BP		BP time	1	1	
SMP init	CPU callbacks	#198, workqueue_online_cpu	25	**11**	**+ 14**
		#154, hv_cpu_init			
		vmstat			
		Others			
sched_init_domains	Creating CPU schedule domain	sched_init_domains	10	2	**+ 8**
Initcalls	Various initcalls	VMBus init	9.4	**7.4**	**+2**
		msr_init			
		percpu_counter_startup			
		cacheinfo_sysfs_init			
		mshv_vtl_init			
			45.4	21.4	

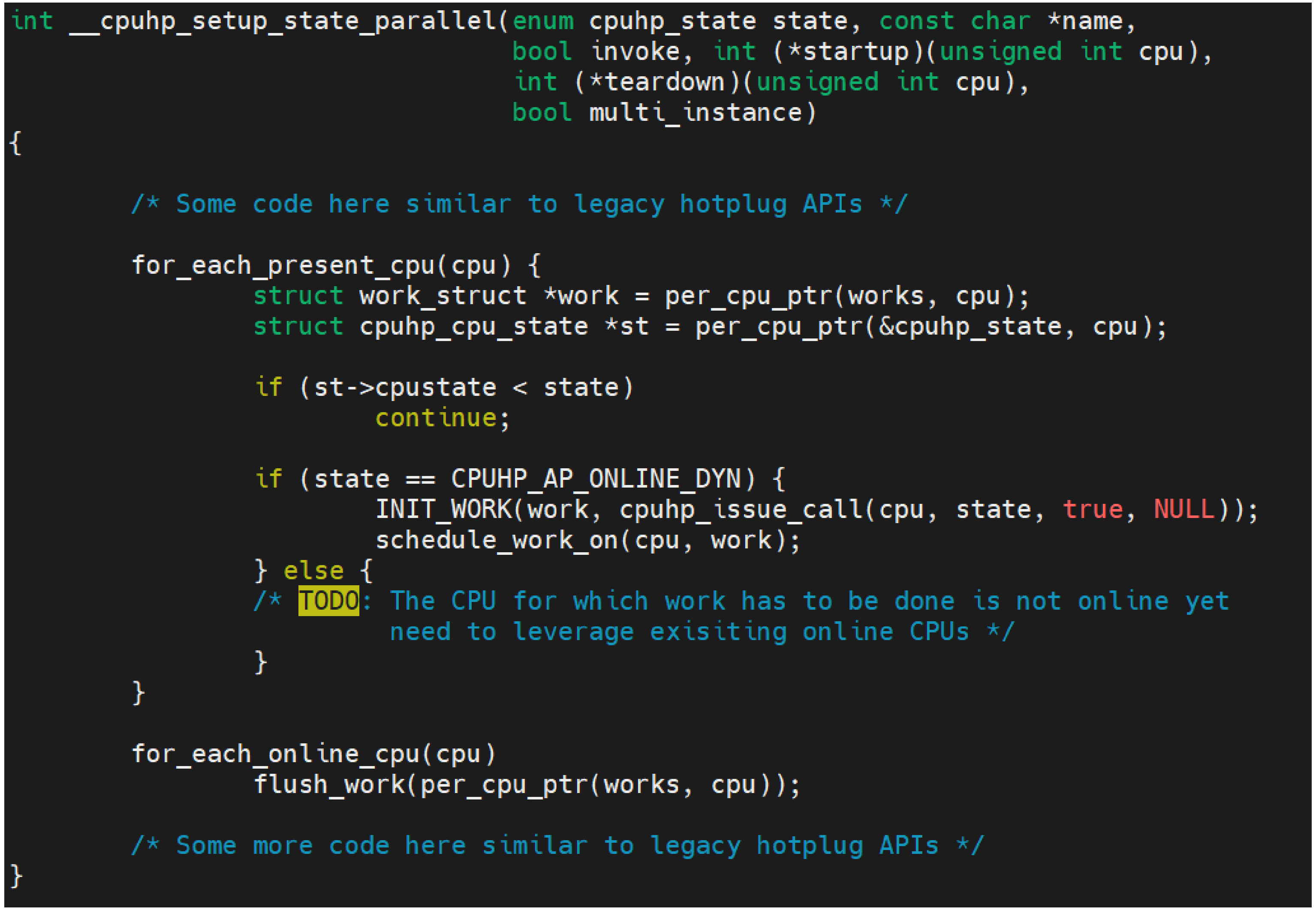
Opportunities for further boot time reduction:

SMP init: 11 seconds

Initcalls: 7.4 seconds

Parallelizing CPU hotplug callbacks

- ➢ Require more parallel callbacks.
- ➢ Idea similar to VMBus init fix
- ➢ Review all the CPU callbacks for potential parallelization.



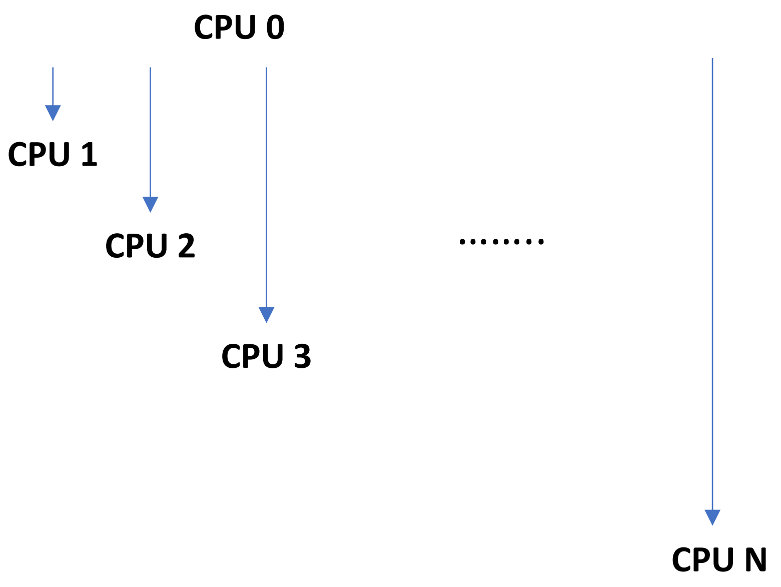
```
int __cpuhp_setup_state_parallel(enum cpuhp_state state, const char *name,
                                 bool invoke, int (*startup)(unsigned int cpu),
                                 int (*teardown)(unsigned int cpu),
                                 bool multi_instance)
{

        /* Some code here similar to legacy hotplug APIs */

        for_each_present_cpu(cpu) {
                struct work_struct *work = per_cpu_ptr(works, cpu);
                struct cpuhp_cpu_state *st = per_cpu_ptr(&cpuhp_state, cpu);

                if (st->cpustate < state)
                        continue;

                if (state == CPUHP_AP_ONLINE_DYN) {
                        INIT_WORK(work, cpuhp_issue_call(cpu, state, true, NULL));
                        schedule_work_on(cpu, work);
                } else {
                /* TODO: The CPU for which work has to be done is not online yet
                         need to leverage exisiting online CPUs */
                }
        }

        for_each_online_cpu(cpu)
                flush_work(per_cpu_ptr(works, cpu));

        /* Some more code here similar to legacy hotplug APIs */
}
```

Group CPU bring-up

Legend:

↓ onlines

Current Approach
CPU 0 onlines all CPUs sequentially



CPU 0

CPU 1

CPU 2

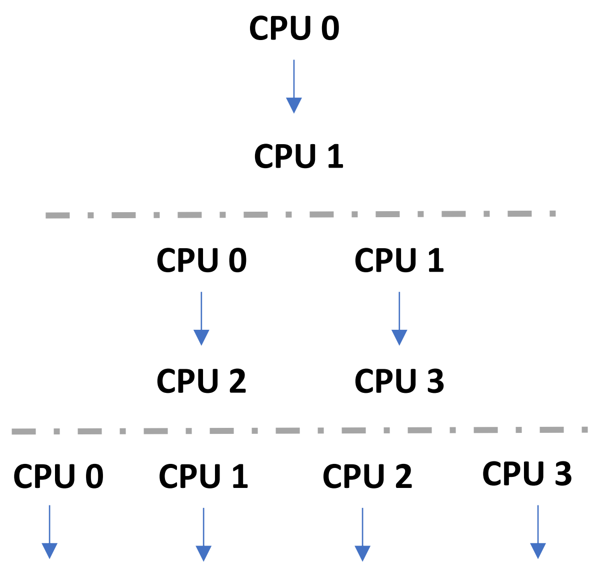
CPU 3

....

CPU N

Theoretical time-complexity: O(N), where N is no. of CPUs

Proposed Approach
CPUs online other CPUs in parallel



CPU 0

CPU 1

CPU 0 CPU 1

CPU 2 CPU 3

CPU 0 CPU 1 CPU 2 CPU 3

Theoretical time-complexity: O(logN), where N is no. of CPUs

Q & A

References

[1] vmstat patch to speed up booting: https://lore.kernel.org/linux-mm/20240812043754.GA7619@linuxonhyperv3.guj3yctzbm1etfxqx2vob5hsef.xx.internal.cloudapp.net/T/#m95acf3fae05f186fc3e8479723cf4adbfda4acfb

[2] VMBus parallel callbacks: https://lore.kernel.org/linux-hyperv/1722488136-6223-1-git-send-email-ssengar@linux.microsoft.com/

[3] Parallel CPU Bringup: https://www.phoronix.com/news/Parallel-CPU-Bringup-Linux-6.5