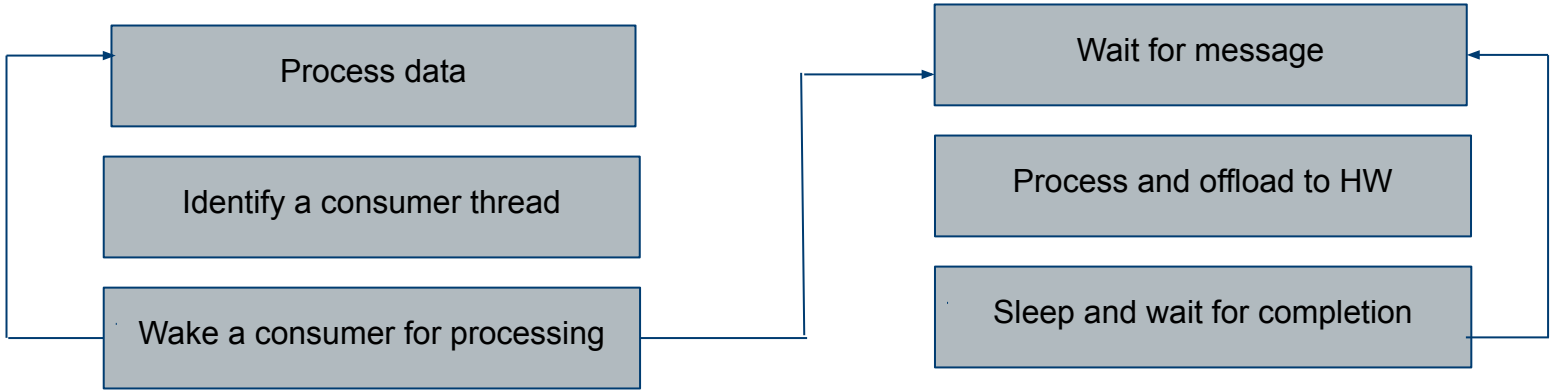
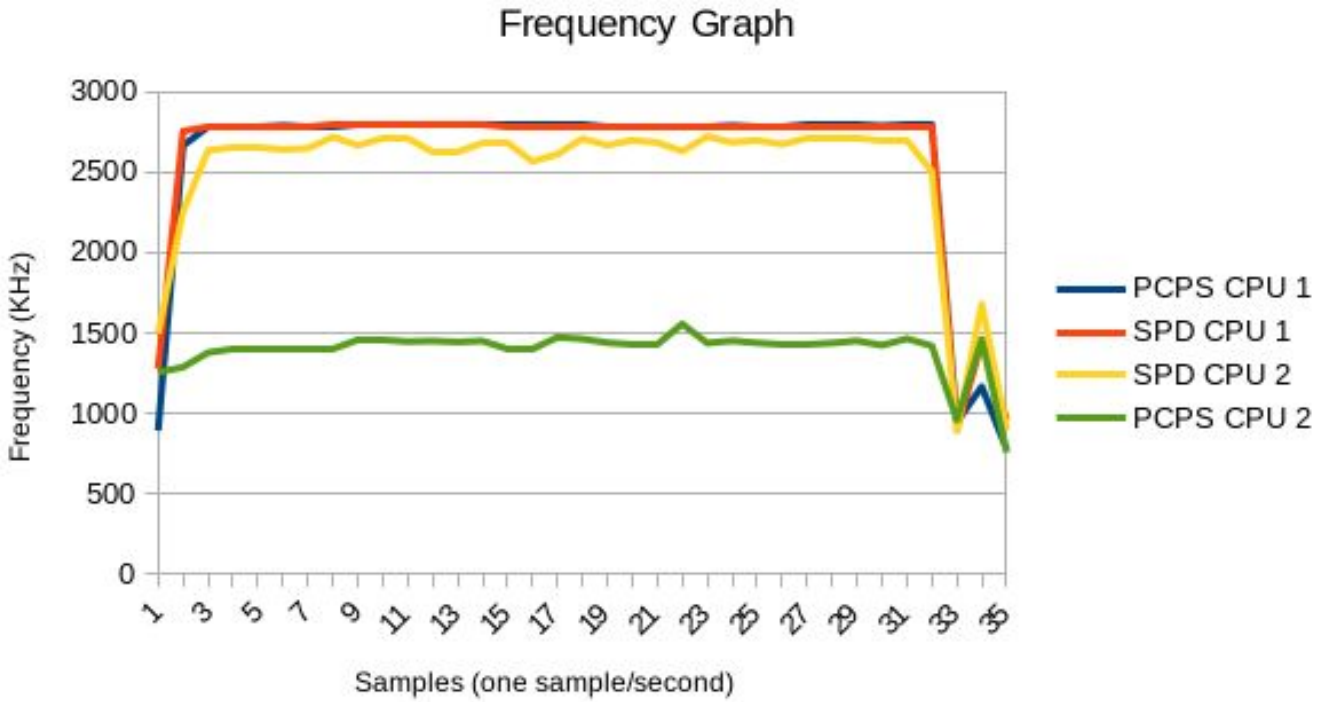
Per Core P-States

- Some CPUs support Per Core P-states
- Significant improvement in active power savings
- Small performance loss with producer consumer workloads
  - HWP has in built optimizations to mitigate
    - Can use some OS hints for further optimizations

Producer Consumer Workload Example

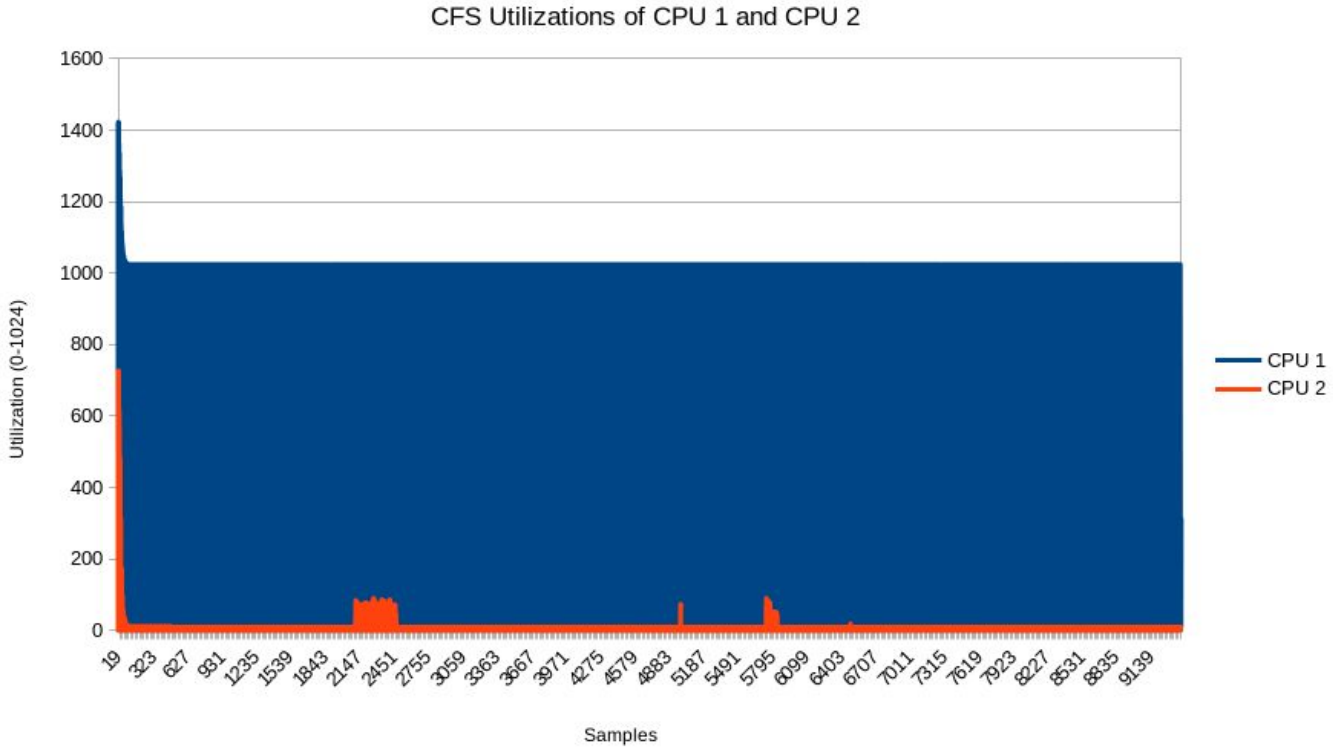


Performance Issues



Perf loss for schbench: 8-10%

CFS Utilization



Some prior work

http://retis.sssup.it/luca/ospm-summit/2017/Downloads/OSPM_PELT_DecayClampingVsUtilEst.pdf

"Tasks with long sleep periods loose too much of their accumulated utilization leading to wrong utilization estimates"

Experiments

Get a new hint for Task Wake up

`SCHED_CPUFREQ_TASK_WAKE`

During enque_task_fair

```
If new_task->state == TASK_WAKING and current_cpu != cpu_of(rq) && (task_util_est(current) -
    (int)rq->cfs.avg.util_est.enqueued) > 512)
        cpufreq_update_util(rq, SCHED_CPUFREQ_WAKE_REMOTE);
```

Process `SCHED_CPUFREQ_TASK_WAKE`

```
Ignored in schedutil governor based on the new flag to reduce impact
Intel P-state
        Boost EPP/HWP min or
        Soc specific mechanism to handle save/restore
```

Suggestions?

Open | 0001-PCPS-improvement.patch ~/development/linux-dev/linux | Save

```
diff --git a/include/linux/sched/cpufreq.h b/include/linux/sched/cpufreq.h
index afa940cd50dc..8dad3395fc5a 100644
--- a/include/linux/sched/cpufreq.h
+++ b/include/linux/sched/cpufreq.h
@@ -10,6 +10,7 @@

 #define SCHED_CPUFREQ_IOWAIT   (1U << 0)
 #define SCHED_CPUFREQ_MIGRATION        (1U << 1)
+#define SCHED_CPUFREQ_WAKE_REMOTE      (1U << 1)

 #ifdef CONFIG_CPU_FREQ
 struct update_util_data {
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
index 036be95a87e9..448f57674300 100644
--- a/kernel/sched/fair.c
+++ b/kernel/sched/fair.c
@@ -5186,8 +5186,23 @@ static inline void update_overutilized_status(struct rq *rq)
                trace_sched_overutilized_tp(rq->rd, SG_OVERUTILIZED);
        }
 }
+
+static inline void notify_task_waking(struct rq *rq, struct task_struct *p, int flags)
+{
+       if (!sched_feat(UTIL_EST))
+               return;
+
+       trace_printk("enqueue_task_fair flags:%x task-state:%lx util:[%lu:%u] current-task-util:[%lu]\n",
+                    flags, p->state, task_util_est(p), rq->cfs.avg.util_est.enqueued,
+                    task_util(current));
+
+       if (p->state == TASK_WAKING && smp_processor_id() != cpu_of(rq) &&
+            ((int)task_util(current) - (int)rq->cfs.avg.util_est.enqueued) > (SCHED_CAPACITY_SCALE >> 1))
+               cpufreq_update_util(rq, SCHED_CPUFREQ_WAKE_REMOTE);
+}
 #else
 static inline void update_overutilized_status(struct rq *rq) { }
+static inline void notify_task_waking(struct rq *rq, struct task_struct *p) { }
 #endif

 /*
@@ -5216,6 +5231,8 @@ enqueue_task_fair(struct rq *rq, struct task_struct *p, int flags)
         */
        if (p->in_iowait)
                cpufreq_update_util(rq, SCHED_CPUFREQ_IOWAIT);
+       else
+               notify_task_waking(rq, p, flags);

        for_each_sched_entity(se) {
                if (se->on_rq)
--
2.17.1
```

Diff | Tab Width: 8 | Ln 159, Col 24 | INS