Using tracing to tune and optimize EAS

Leo Yan & Daniel Thompson
Linaro Support and Solutions Engineering
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow

● Worked examples
  ○ Development platform for the worked examples
  ○ Analyze for task ping-pong issue
  ○ Analyze for small task staying on big core

● Further reading
Typical workflow for optimizing GTS

This simple workflow is easy to understand but has problems in practice.

Tunables are complex and interact with each other (making it hard to decide which tuneable to adjust).

Tuning for multiple use-cases is difficult.

Tuning is SoC specific, optimizations will not necessarily apply to other SoCs.
GTS tunables

up_threshold
down_threshold
packing_enable
load_avg_period_ms
frequency_invariant_load_scale

hispeed_freq
go_hispeed_load
target_loads
Timer_rate
min_sample_time
above_hispeed_delay

GTS

CPUFreq
Interactive Governor
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow

● Worked examples
  ○ Development platform for the worked examples
  ○ Task ping-pong issue
  ○ Small task staying on big core

● Further reading
Typical workflow for optimizing EAS systems

- **Benchmark**
- **Trace a use-case**
- **Examine traces**
- **Improve decisions**

Workflow is knowledge intensive.

Decisions can be improved by improving the power model or by finding new opportunities in the scheduler (a.k.a. debugging).

Optimizations are more portable.
- Can be shared for review
- Likely to benefit your new SoC
### Trace points for EAS

Has a set of stock trace points in kernel for diving into debugging.

Trace points are added by patches marked “DEBUG”.

Not posted to LKML, currently only found in product focused patchsets.

Enable kernel config: `CONFIG_FTRACE`

---

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Description</th>
<th>Commit Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>DEBUG: schedtune: add tracepoint for schedtune_tasks_up</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>DEBUG: schedtune: add tracepoint for CPU boost signal</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>DEBUG: schedtune: add tracepoint for SchedTune config</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Dietmar Eggemann</td>
<td>DEBUG: sched: add energy proofs interface</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Juri Lelli</td>
<td>DEBUG: sched,cpufreq: add cpu_capacity change tracepoint</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Juri Lelli</td>
<td>DEBUG: sched: add tracepoint for CPU load/util signals</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Juri Lelli</td>
<td>DEBUG: sched: add tracepoint for task load/util signals</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Juri Lelli</td>
<td>DEBUG: sched: add tracepoint for cpu/freq scale invariance</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>sched/fair: filter energy_diff() based on energy_payoff</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>sched/tune: add support to compute normalized energy</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>sched/fair: keep track of energy/capacity variations</td>
<td></td>
</tr>
<tr>
<td>2016-03-21</td>
<td>Patrick Bellasi</td>
<td>sched/fair: add boosted task util</td>
<td></td>
</tr>
</tbody>
</table>
Trace points for EAS - cont.

Tracepoints in mainline kernel

<table>
<thead>
<tr>
<th>PEI signals</th>
<th>Sched_switch</th>
<th>Sched_migrate_task</th>
<th>Sched_wakeup</th>
<th>Sched_wakeup_new</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_contrib_scale_f</td>
<td>sched_load_avg_task</td>
<td>sched_load_avg_cpu</td>
<td>sched_switch</td>
<td>sched_migrate_task</td>
</tr>
<tr>
<td>sched_load_avg_task</td>
<td>sched_load_avg_cpu</td>
<td>sched_load_avg_cpu</td>
<td>sched_wakeup</td>
<td>sched_wakeup_new</td>
</tr>
</tbody>
</table>

Tracepoints for EAS extension

<table>
<thead>
<tr>
<th>EAS core</th>
<th>SchedTune</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_energy_diff</td>
<td>sched_tune_config</td>
</tr>
<tr>
<td>sched_overutilized</td>
<td>sched_boost_cpu</td>
</tr>
<tr>
<td>cpufreq_sched_throttled</td>
<td>sched_tune_tasks_update</td>
</tr>
<tr>
<td>cpufreq_sched_request_opp</td>
<td>sched_tune_boostgroup_update</td>
</tr>
<tr>
<td>cpufreq_sched_update_capacity</td>
<td>sched_boost_task</td>
</tr>
<tr>
<td>cpufreq_sched_update_capacity</td>
<td>sched_tune_filter</td>
</tr>
</tbody>
</table>

E.g. enable trace points:

```
trace-cmd start -e sched_energy_diff -e sched_wakeup
```
### Summary

<table>
<thead>
<tr>
<th>Features</th>
<th>EAS</th>
<th>GTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make decision strategy</td>
<td>Power modeling</td>
<td>Heuristics thresholds</td>
</tr>
<tr>
<td>Frequency selection</td>
<td>Sched-freq or sched-util, integrated with scheduler</td>
<td>Governor’s cascaded parameters</td>
</tr>
<tr>
<td>Scenario based tuning</td>
<td>schedTune (CGroup)</td>
<td>None</td>
</tr>
</tbody>
</table>

Energy awared scheduling (EAS) has very few tunables and thus requires a significantly different approach to tuning and optimization when compared to global task scheduling (GTS).
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow
● Worked examples
  ○ Development platform for the worked examples
  ○ Task ping-pong issue
  ○ Small task staying on big core
● Further reading
LISA - interactive analysis and testing

- “Distro” of python libraries for interactive analysis and automatic testing
- Libraries support includes
  - Target control and manipulation (set cpufreq mode, run this workload, initiate trace)
  - Gather power measurement data and calculate energy
  - Analyze and graph trace results
  - Test assertions about the trace results (e.g. big CPU does not run more than 20ms)
- Interactive analysis using ipython and jupyter
  - Provides a notebook framework similar to Maple, Mathematica or Sage
  - Notebooks mix together documentation with executable code fragments
  - Notebooks record the output of an interactive session
  - All permanent file storage is on the host
  - Trace files and graphs can be reexamined in the future without starting the target
- Automatic testing
  - Notebooks containing assertion based tests that can be converted to normal python
General workflow for LISA

https://events.linuxfoundation.org/sites/events/files.slides/ELC16_LISA_20160326.pdf
LISA interactive test mode

http://127.0.0.1:8888 with ipython file

Menu & control buttons

Markdown (headers)

Execute box with python programming

Result box, the results of experiments are recorded when next time reopen this file
kernelshark

Filters for events, tasks, and CPUs

Details for events
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow

● Worked examples
  ○ Development platform for the worked examples
  ○ Task ping-pong issue
  ○ Small task staying on big core

● Further reading
Development platform for the worked examples

- All examples use artificial workloads to provoke a specific behaviour
  - It turned out to be quite difficult to deliberately provoke undesired behavior!
- Examples are reproducible on 96Boards HiKey
  - Octo-A53 multi-cluster (2x4) SMP device with five OPPs per cluster
    - Not big.LITTLE, and not using a fast/slow silicon process
  - We are able to fake a fast/slow system by using asymmetric power modeling parameters and artificially reducing the running/runnable delta time for “fast” CPU so the metrics indicate that is has a higher performance
- Most plots shown in these slides are copied from a LISA notebook
  - Notebooks and trace files have been shared for use after training
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow

● Worked examples
  ○ Development platform for the worked examples
  ○ Task ping-pong issue
  ○ Small task staying on big core

● Further reading
Testing environment

● **CPU capacity info**
  ○ The little core’s highest capacity is 447@850MHz
  ○ The big core’s highest capacity is 1024@1.1GHz
  ○ This case is running with correct power model parameters

● **Test case**
  ○ 16 small tasks are running with 15% utilization of little CPU (util ~ = 67)
  ○ A single large task is running with 40% utilization of little CPU (util ~ = 180)
General analysis steps

Step 1: Run workload and generate trace data

1. Connect with target
2. Generate workload
3. Filter out tasks
4. Platform description file

Step 2: Analyze trace data

1. Analyze events
2. Filter out tasks
3. Analyze tasks
4. LISA::TraceAnalysis()
Connect with target board

```python
In [ ]: my_target_conf = {
    "platform" : 'linux',
    "board" : 'hikey',
    "modules" : [
        'cputfreq'
    ],
    "host" : '192.168.0.20',
    "username" : 'root',
    "password" : 'root',
    "rtapp-calib": {
        "0": 258, "1": 250, "2": 253, "3": 251, "4": 251, "5": 251, "6": 252, "7": 250
    }
}
my_tests_conf = {
    "tools" : ['rt-app', 'taskset', 'trace-cmd'],
    "ftrace" : {
        "events" : [
            "sched_load_avg_cpu",
            "sched_load_avg_task",
        ],
        "buffersize" : 40960
    },
}

In [ ]:
t = TestEnv(target_conf=my_target_conf, test_conf=my_tests_conf)
target = tc.target

Specify target board info for connection

Calibration for CPUs

Tools copied to target board

Enable ftrace events

Create connection
Generate and execute workload

```
rtapp = RTA(target, 'simple', calibration=te.calibration())
heavy = Periodic(duty_cycle_pct=40, duration_s=5, period_ms=5)
light = Periodic(duty_cycle_pct=15, duration_s=5, period_ms=5)
rtapp.conf(
    kind='profile',
    params=[
        'task010': light.get(),
        'task011': light.get(),
        'task012': light.get(),
        'task013': light.get(),
        'task014': light.get(),
        'task015': light.get(),
        'task016': light.get(),
        'task017': light.get(),
        'task018': light.get(),
        'task019': light.get(),
        'task020': light.get(),
        'task021': light.get(),
        'task022': light.get(),
        'task023': light.get(),
        'task024': light.get(),
        'task025': light.get(),
        'task01': heavy.get(),
    ],
    run_dir=target.working_directory
);
```

Define workload

```
logging.info('##### Setup FTrace')
tftrace.start()
logging.info('##### Start energy sampling')
temeter.reset()
logging.info('##### Start RTApp execution')
rtapp.run(out_dir=te.res_dir)
logging.info('##### Read energy consumption: %s/energy.json', te.res_dir)
(nrg, nrg_file) = temeter.report(out_dir=te.res_dir)
logging.info('##### Stop FTrace')
tftrace.stop()
trace_file = os.path.join(te.res_dir, 'trace.dat')
logging.info('##### Save FTrace: %s', trace_file)
tftrace.get_trace(trace_file)
logging.info('##### Save platform description: %s/platform.json', te.res_dir)
(plt, plt_file) = teplatform_dump(te.res_dir)
```

Capture Ftrace data
Capture energy data
Execute workload
Graph showing task placement in LISA

In [5]:
```python
trace_file = "/home/leoy/Work/disk/tools/x86/lisa-us/results/test_case_task_ping_pong/trace.dat"
print trace_file
events_to_parse = [
    "sched_wakeup",
    "sched_wakeup_new",
    "sched_load_avg_cpu",
    "sched_load_avg_task",
    "sched_boost_cpu",
    "sched_boost_task",
    "sched_energy_diff",
    "sched_switch",
    "sched_migrate_task",
    "sched_overutilized"]
ftrace = trappy.FTrace(trace_file, normalize_time=True, events=events_to_parse, window=(0, None))
/home/leoy/Work/disk/tools/x86/lisa-us/results/test_case_task_ping_pong/trace.dat
```

Specify events to be extracted

Specify time interval

In [6]:
```python
trappy.plotter.plot_trace(ftrace)  Display task placement graph
```
First make a quick graph showing task placement...

Too many tasks, need method to quickly filter out statistics for every task.
... and decide how to tackle step 2 analysis

Step 1: Run workload and generate trace data

Connect with target

LISA:: TestEnv() → Generate workload → LISA:: rtapp()

Platform description file

platform.json

Step 2: Analyze trace data

Analyze events

LISA:: Trace() → Filter out tasks → LISA:: Filters() → Analyze tasks → LISA:: TraceAnalysis()
Analyze trace data for events

```python
platform.json
{
    "clusters": {
        "big": [4, 5, 6, 7],
        "little": [0, 1, 2, 3]
    },
    "cpus_count": 8,
    "freqs": {
        "big": [208000, 432000, 729000, 960000, 1200000],
        "little": [208000, 432000, 729000, 960000, 1200000]
    }
}
```

```python
trace.dat
Format:
"SYSTRACE" or "Ftrace"
```
Selecting only task of interest (big tasks)

```python
from filters import Filters
fl = Filters(trace)
fl.setXTTimeRange(t_min, t_max)

# Get a list of tasks which are the most big in the trace
top_big_tasks = fl.topBigTasks(
    max_tasks=15,  # Maximum number of tasks to report
    min_utilization=10,  # Minimum utilization to be considered "big"
    # default: LITTLE CPUs max capacity
    min_samples=100,  # Number of samples over the minimum utilization
)
```

top_big_tasks =
{'mmcqd/0': 733,
 'Task01' : 2441,
 'task010': 2442,
 'task011': 2443,
 'task012': 2444,
 'task015': 2447,
 'task016': 2448,
 'task017': 2449,
 'task019': 2451,
 'task020': 2452,
 'task021': 2453,
 'task022': 2454,
 'task023': 2455,
 'task024': 2456,
 'task025': 2457}
Plot big tasks with TraceAnalysis

```python
from trace_analysis import TraceAnalysis

ta = TraceAnalysis(
    trace, # LISA::Trace object
tasks=top_big_tasks, # (optional) list of tasks to plot
plotsdir=res_dir
)

# Define time ranges for all the time based plots
ta.setXTimeRange(t_min, t_max)

ta.plotTasks(top_big_tasks)
```
TraceAnalysis graph of task residency on CPUs

1. At beginning task is placed on big core
2. Then it ping-pongs between big cores and LITTLE cores
TraceAnalysis graph of task PELT signals

Big core’s highest capacity

load_avg
= PELT(running time + runnable time) * weight
= PELT(running time + runnable time) (if NICE = 0)

LITTLE core’s highest capacity

util_avg = PELT(running time)

The difference between load_avg and util_avg is task’s runnable time on rq (for NICE=0)
System cross tipping point for “over-utilized”

static int select_task_rq_fair(struct task_struct *p, int prev_cpu, int sd_flag, int wake_flags)
{
    [...]
    if (!sd) {
        if (energy_aware() && !cpu_rq(cpu)->rd->overutilized)
            new_cpu = energy_aware_wake_cpu(p, prev_cpu);
        else if (sd_flag & SD_BALANCE_WAKE) /* XXX always ? */
            new_cpu = select_idle_sibling(p, new_cpu);
    } else while (sd) {
        [...]
    }
}

static void enqueue_task_fair(struct rq *rq, struct task_struct *p, int flags)
{
    [...]
    if (!se) {
        add_nr_running(rq, 1);
        if (!task_new && !rq->rd->overutilized &&
            cpu_overutilized(rq->cpu))
            rq->rd->overutilized = true;
    }
}

static struct sched_group *find_busiest_group(struct lb_env *env)
{
    if (energy_aware() && !env->dst_rq->rd->overutilized)
        goto out_balanced;
    [...]
}

static int select_task_rq_fair(struct task_struct *p, int prev_cpu, int sd_flag, int wake_flags)
{
    [...]
    if (!sd) {
        if (energy_aware() && !cpu_rq(cpu)->rd->overutilized)
            new_cpu = energy_aware_wake_cpu(p, prev_cpu);
        else if (sd_flag & SD_BALANCE_WAKE) /* XXX always ? */
            new_cpu = select_idle_sibling(p, new_cpu);
    } else while (sd) {
        [...]
    }
}
Write function to Analyze tipping point

```python
def analysis_tipping_point(pid):
    colors=["c", "b", "r"]
    gs = gridspec.GridSpec(1, 1)
    fig = plt.figure(figsize=(16, 5))
    fig.subplots_adjust(top=0.92)
    plt.suptitle("Tipping point analysis", fontsize=14, fontweight='bold')
    axes = plt.subplot(gs[0, 0])

dest_cpu = df_migrate_task[df_migrate_task.pid == pid]["dest_cpu"]
dest_cpu.plot(ax=axes, drawstyle='steps-post', color=colors[1], linewidth=3, ylim=(0, 8))

overutilized = df_overutilized[['overutilized']]
tipping_point.plot(ax=axes, drawstyle='steps-post', color=colors[2], linewidth=3, ylim=(0, 8))
    axes.grid(True)
```
Plot for tipping point

System is over tipping point, migrate task from CPU3 (little core) to CPU4 (big core)

System is under tipping point, migrate task from CPU4 (big core) to CPU3 (little core)
Detailed trace log for migration to big core

Migrate big task from CPU3 to CPU4

nohz_idle_balance() for tasks migration
Issue 1: migration big task back to LITTLE core

- Migrate task to LITTLE core
- Migrate big task from CPU4 to CPU3
Issue 2: migration small tasks to big core

CPU is overutilized again

Migrate small tasks to big core
Tipping point criteria

**Over tipping point**

Any CPU: \( \text{cpu\_util(cpu)} > \text{cpu\_capacity(cpu)} \times 80\% \)

- E.g. LITTLE core: \( \text{cpu\_capacity(cpu0)} = 447 \)
  - Util: 90%
  - 80% of capacity

- E.g. Big core: \( \text{cpu\_capacity(cpu4)} = 1024 \)
  - Util: 90%
  - 80% of capacity

**Under tipping point**

ALL CPUs: \( \text{cpu\_util(cpu)} < \text{cpu\_capacity(cpu)} \times 80\% \)

- E.g. LITTLE core: \( \text{cpu\_capacity(cpu0)} = 447 \)
  - Util: 70%
  - 80% of capacity

- E.g. Big core: \( \text{cpu\_capacity(cpu4)} = 1024 \)
  - Util: 70%
  - 80% of capacity
Phenomenon for ping-pong issue

Over tipping point

LITTLE cluster

big cluster

Task1

big cluster

Task1

Under tipping point

LITTLE cluster

big cluster

Task1

big cluster

Migration
Fixes for ping-pong issue

Over tipping point
- Filter out small tasks to avoid migrating to big core

Under tipping point
- Avoid migrating big tasks back to LITTLE cluster
Fallback to LITTLE cluster after it is idle

Over tipping point

LITTLE cluster

big cluster

Task 1

Migration

Under tipping point

LITTLE cluster

big cluster

Task 1

Migrate big task back to LITTLE cluster if it’s idle
Filter out small tasks: task running time < ¼ LITTLE CPU capacity. These tasks will NOT be migrated to big core after return 0. Result: Only big tasks has a chance to migrate to big core.
Avoid migrating big task to LITTLE cluster

```
static int select_task_rq_fair(struct task_struct *p, int prev_cpu, int sd_flag, int wake_flags)
{
    [...]
    if (!sd) {
        if (energy Aware() && 
            (!need_spread_task(cpu) || need_filter_task(p)))
            new_cpu = energy aware wake_cpu(p, prev_cpu);
        else if (sd_flag & SD_BALANCE_WAKE) /* XXX always ? */
            new_cpu = select_idle_sibling(p, new_cpu);
    } else while (sd) {
        [...]
    }
}
```

```
static bool need_spread_task(int cpu)
{
    struct sched_domain *sd;
    int spread = 0, i;
    if (cpu_rq(cpu)->rd->overutilized)
        return 1;
    sd = rcu_dereference_check_sched_domain(cpu_rq(cpu)->sd);
    if (!sd)
        return 0;
    for_each_cpu(i, sched_domain_span(sd)) {
        if (cpu_rq(i)->cfs.h_nr_running >= 1 &&
            cpu_halfutilized(i)) {
            spread = 1;
            Break;
        }
    }
    return spread;
}
```

Check if cluster is busy or not as well as checking system tipping point:
- Easier to spread tasks within cluster if cluster is busy
- Fallback to migrating big task when cluster is idle
static bool need_filter_task(struct task_struct *p) {
    int cpu = task_cpu(p);
    int origin_max_cap = capacity_orig_of(cpu);
    int target_max_cap = cpu_rq(cpu)->rd->max_cpu_capacity.val;
    struct sched_domain *sd;
    struct sched_group *sg;

    sd = rcu_dereference(per_cpu(sd_ea, cpu));
    sg = sd->groups;
    do {
        int first_cpu = group_first_cpu(sg);

        if (capacity_orig_of(first_cpu) < target_max_cap &&
            task_util(p) * 4 < capacity_orig_of(first_cpu))
            target_max_cap = capacity_orig_of(first_cpu);
    } while (sg = sg->next, sg != sd->groups);

    if (target_max_cap < origin_max_cap)
        return 1;

    return 0;
}

Two purposes of this function:
  ● Select small tasks (task running time < \frac{1}{4} LITTLE CPU capacity)
    and keep them on the energy aware path
  ● Prevent energy aware path for big tasks on the big core from
doing harm to little tasks.
Results after applying patches

The big task always run on CPU6 and small tasks run on LITTLE cores!
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow

● Worked examples
  ○ Development platform for the worked examples
  ○ Task ping-pong issue
  ○ Small task staying on big core

● Further reading
Testing environment

- Testing environment
  - The LITTLE core’s highest capacity is 447@850MHz
  - The big core’s highest capacity is 1024@1.1GHz
  - Single small task is running with 9% utilization of big CPU (util ~= 95)

- Phenomenon
  - The single small task runs on big CPU for long time, even though its utilization is well below the tipping point
Global View For Task’s Placement

Small task run at big core for about 3s, during this period the system is not busy.
Analyze task utilization

```python
trappy.LinePlot(
    ftrace,
    signals=[
        'sched_load_avg_task:util_avg',
    ],
    pivot='pid',
    filters={'comm': ['task1']},
    drawstyle='steps-post',
    marker = '+').view()
```

Analyze task’s utilization signal

Filter only related tasks
PELT Signals for task utilization

The task utilization is normalized to value ~95 on big core, this utilization does not exceed the LITTLE core’s tipping point of $447 \times 80\% = 358$. Thus the LITTLE core can meet the task’s requirement for capacity, so scheduler should place this task on a LITTLE core.
Use kernelshark to check wake up path

In energy aware path, we would expect to see "sched_boost_task", but in this case the event is missing, implying the scheduler performed normal load balancing because “overutilized” flag is set. Thus the balancer is run to select an idle CPU in the lowest schedule domain. If previous CPU is idle the task will stick on previous CPU so it can benefit from a “hot cache”.
The “tipping point” has been set for long time

```c
static inline void update_sg_lb_stats(struct lb_env *env,
   struct sched_group *group, int load_idx,
   int local_group, struct sg_lb_stats *sgs,
   bool *overload, bool *overutilized)
{
  unsigned long load;
  int i, nr_running;

  memset(sgs, 0, sizeof(*sgs));

  for_each_cpu_and(i, sched_group_cpus(group), env->cpus) {
    [...]
    if (cpu_overutilized(i)) {
      *overutilized = true;
      if (!sgs->group_misfit_task && rq->misfit_task)
        sgs->group_misfit_task = capacity_of(i);
    }
    [...]
  }
}
```

*overutilized is initialized as ‘false’ before we commence the update, so if any CPU is over-utilized, then this is enough to keep us over the tipping-point.

So need analyze the load of every CPU.
def analysis_cpu_idle_state_and_util_avg(cpu):
    colors=['c', 'b', 'r']

    gs = gridspec.GridSpec(1, 1)
    fig = plt.figure(figsize=(16, 5))
    fig.subplots_adjust(top=0.92)
    plt.suptitle("Idle State and Utilization", fontsize=14, fontweight='bold')

    axes = plt.subplot(gs[0, 0])

    cpu_state = df_idle_state[df_idle_state.cpu_id == cpu][['state']]
    cpu_state[cpu_state.state == 4294967295] = -1
    cpu_state.plot(ax=axes, drawstyle='steps-post', color=colors[1], linewidth=3, ylim=(-1,20))

    load_avg_cpu = df_load_avg_cpu[df_load_avg_cpu.cpu == cpu][['util_avg']]
    load_avg_cpu['util_avg'] = load_avg_cpu['util_avg'] / 100
    print load_avg_cpu
    load_avg_cpu.plot(ax=axes, drawstyle='steps-post', color=colors[0], linewidth=3)

    axes.grid(True)
CPU utilization does not update during idle

CPU utilization is updated when CPU is woken up after long time
Fix Method: ignore overutilized state for idle CPUs

```c
static inline void update_sg_lb_stats(struct lb_env *env,
    struct sched_group *group, int load_idx,
    int local_group, struct sg_lb_stats *sgs,
    bool *overload, bool *overutilized)
{
    unsigned long load;
    int i, nr_running;

    memset(sgs, 0, sizeof(*sgs));

    for_each_cpu_and(i, sched_group_cpus(group), env->cpus) {
        [...]  
        if (cpu_overutilized(i) && !idle_cpu(i)) {
            *overutilized = true;
            if (!sgs->group_misfit_task && rq->misfit_task)
                sgs->group_misfit_task = capacity_of(i);
        }
        [...]  
    }
}
```

Code flow is altered so we only consider the overutilized state for non-idle CPUs
After applying patch to fix this...
Agenda

● Background
  ○ Review of typical workflow for GTS tuning
  ○ Introduce a workflow for EAS tuning
  ○ Quick introduction of the tools that support the new workflow

● Worked examples
  ○ Development platform for the worked examples
  ○ Task ping-pong issue
  ○ Small task staying on big core

● Further reading
Related materials

● Notebooks and related materials for both worked examples
  ○ https://fileserver.linaro.org/owncloud/index.php/s/5gpVpzN0FdxMmGI
  ○ ipython notebooks for workload generation and analysis
  ○ Trace data before and after fixing together with platform.json

● Patches are under discussion on eas-dev mailing list
  ○ sched/fair: support to spread task in lowest schedule domain
  ○ sched/fair: avoid small task to migrate to higher capacity CPU
  ○ sched/fair: filter task for energy aware path
  ○ sched/fair: consider over utilized only for CPU is not idle
Next steps

● **You** can debug the scheduler
  ○ Try to focus on decision making, not hacks
  ○ New decisions should be as generic as possible (ideally based on normalized units)
  ○ Sharing resulting patches for review is highly recommended
    ■ Perhaps fix can be improved or is already expressed differently by someone else

● Understanding tracepoint patches and the tooling from ARM
  ○ Basic python coding experience is needed to utilize LISA libraries

● Understanding SchedTune
  ○ SchedTune interferes with the task utilization levels for CPU selection and CPU utilization levels to bias CPU and OPP selection decisions
  ○ Evaluate energy-performance trade-off
  ○ Without tools, it’s hard to define and debug SchedTune boost margin on a specific platform
Thank You

#LAS16
For further information: www.linaro.org or support@linaro.org
LAS16 keynotes and videos on: connect.linaro.org