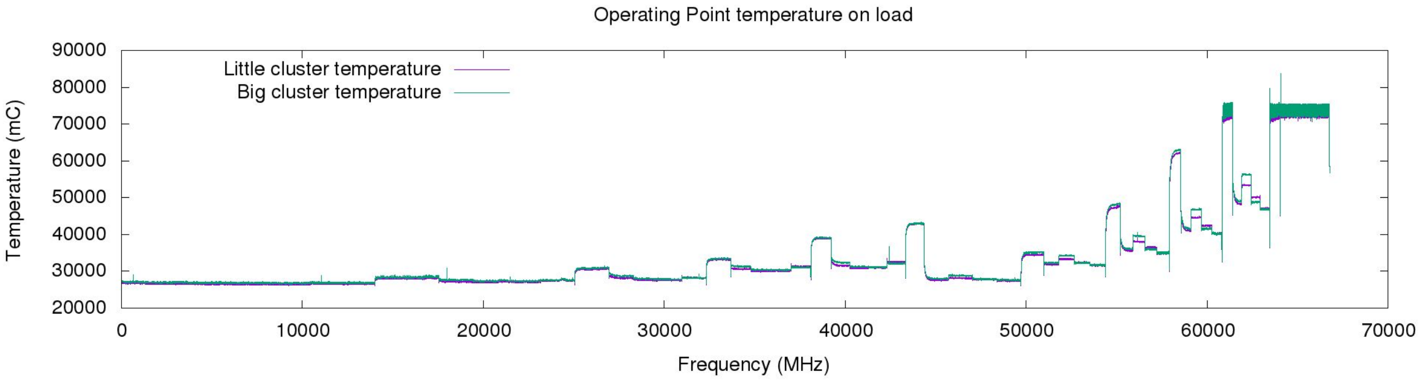
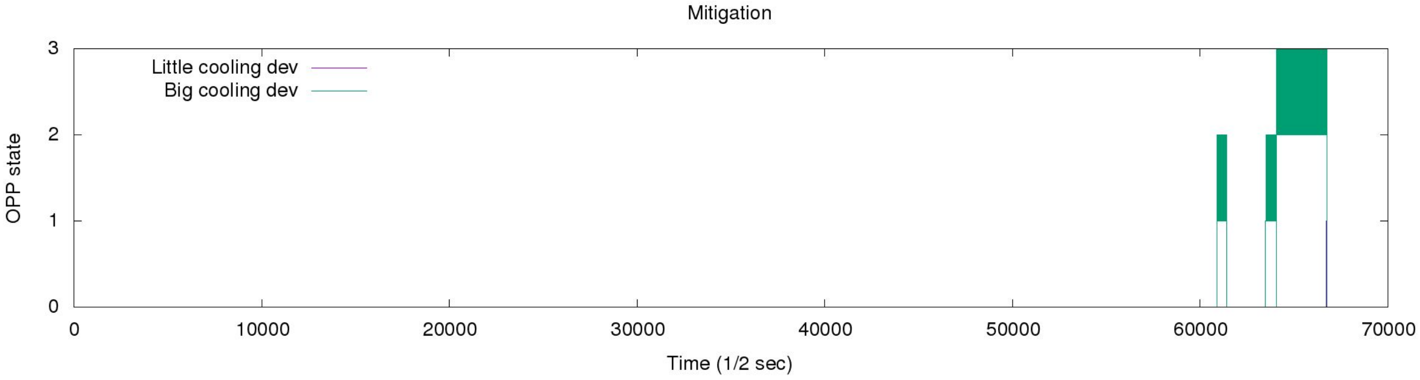
Idle injection

Daniel Lezcano

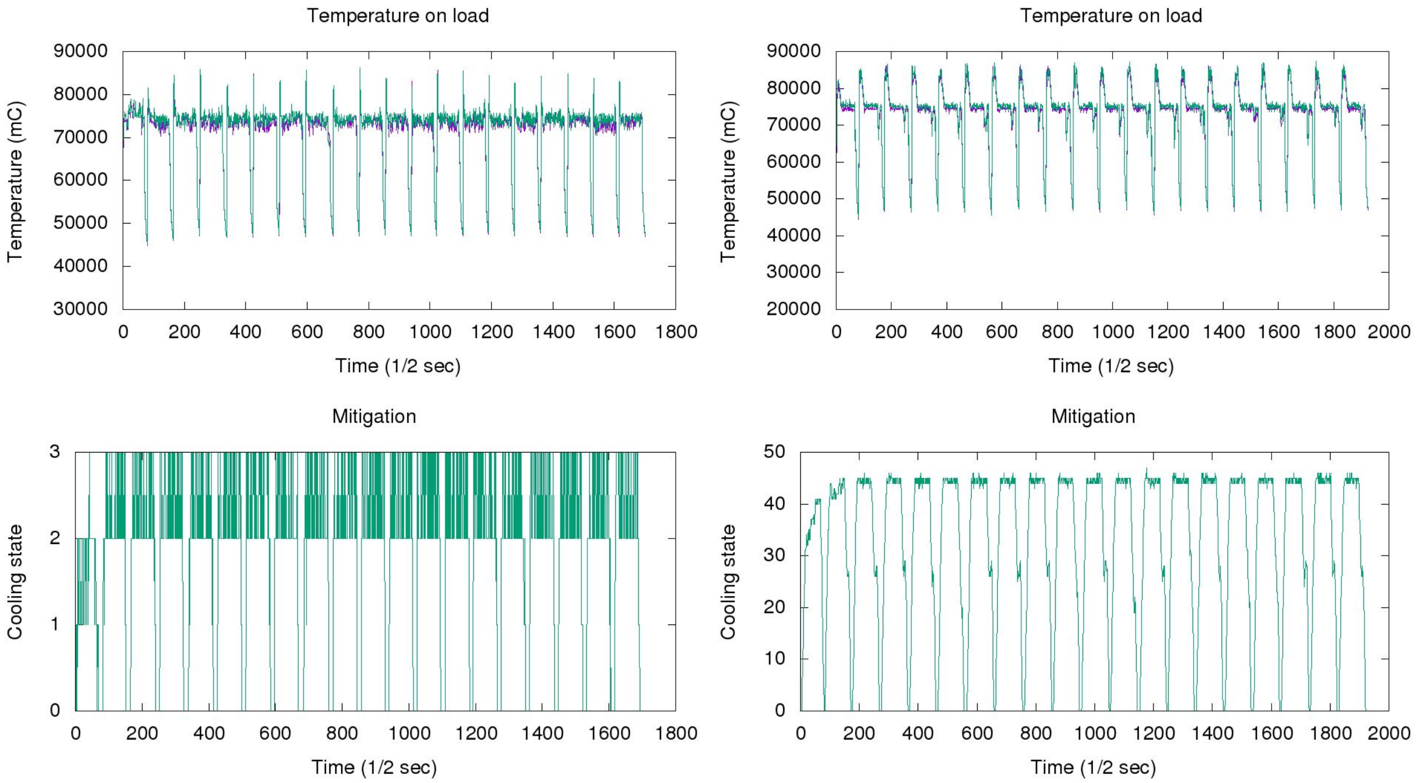
LEADING
COLLABORATION
IN THE ARM
ECOSYSTEM

Temperature and mitigation





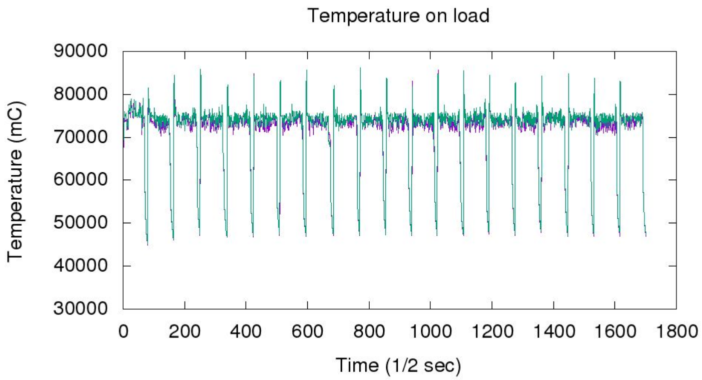
Load and mitigation

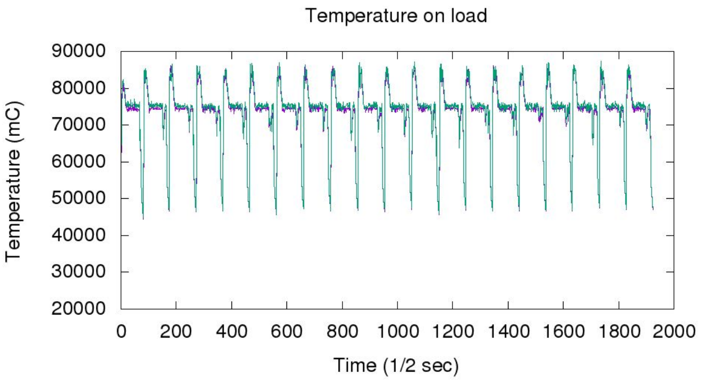


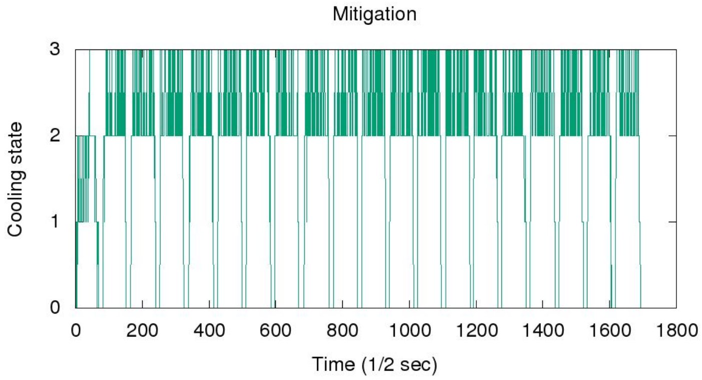
Combo cooling device

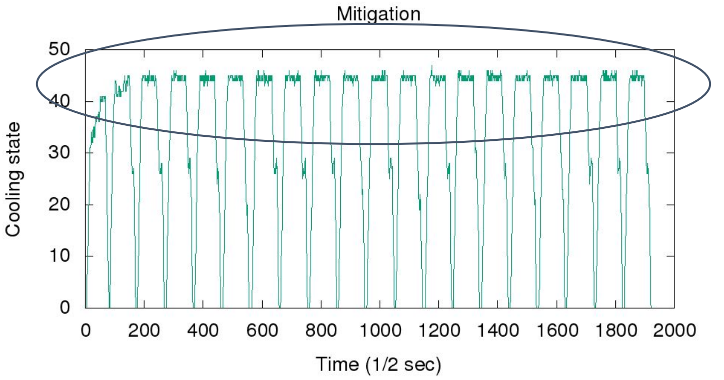
Is there a way to get benefit of both approaches ?

Load and mitigation









Capacity reduction

- With a 40% idle duration, we have ~30% real idle time
  - Cost to enter / exit the idle state
  - Cache flushing and refill
  - Latency added

- Compute capacity is theoretically reduced by 30%, so **716**

- And ...

Compute capacity / Frequency

Frequency (MHz)	Capacity
2362	**1024**
2112	**916**
1805	**783**
1421	**616**
903	**391**

716

Compute capacity / Frequency

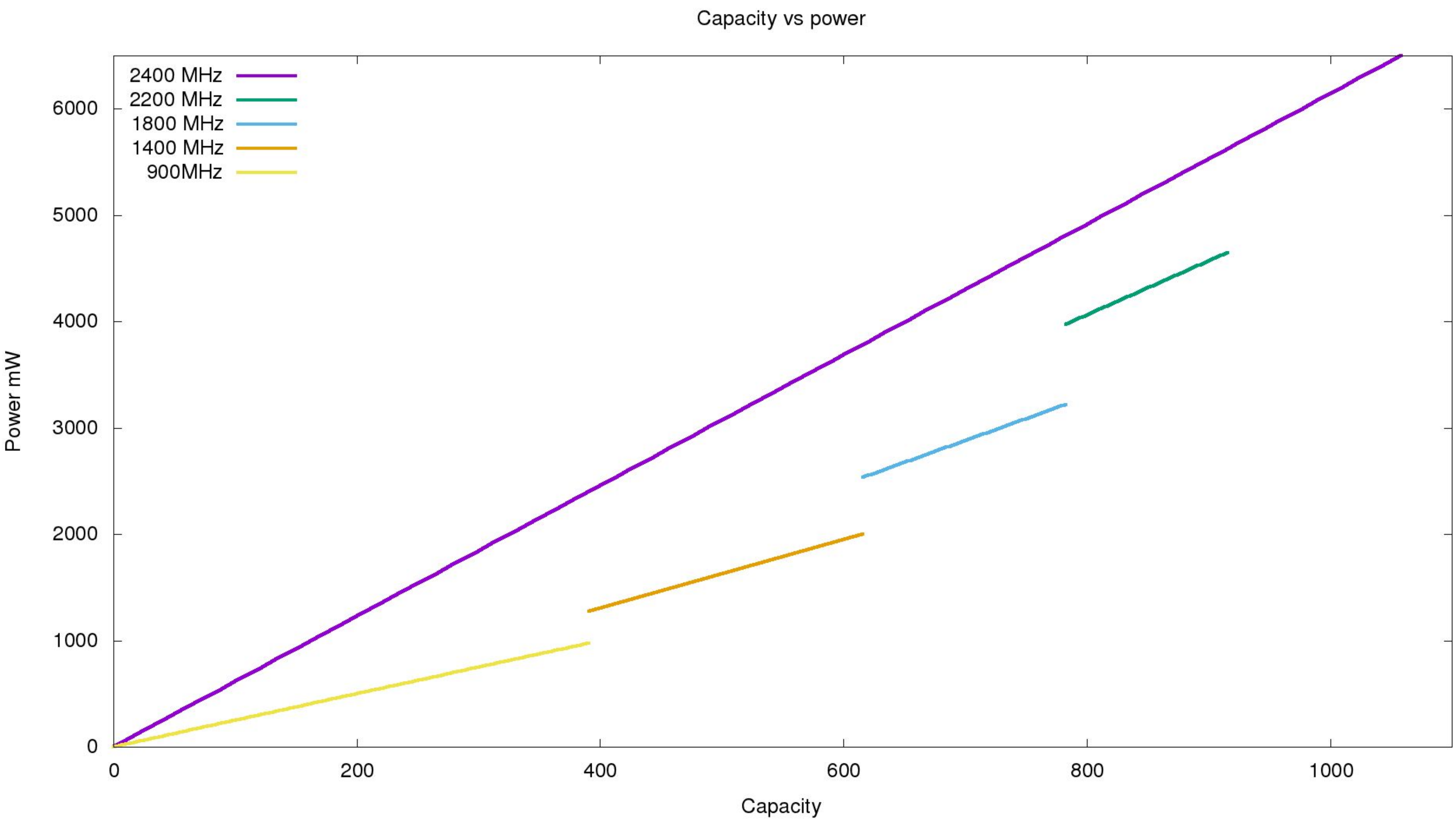
Frequency (MHz)	Capacity
2362	**1024**
2112	**916**
1805	**783**
1421	**616**
903	**391**

716

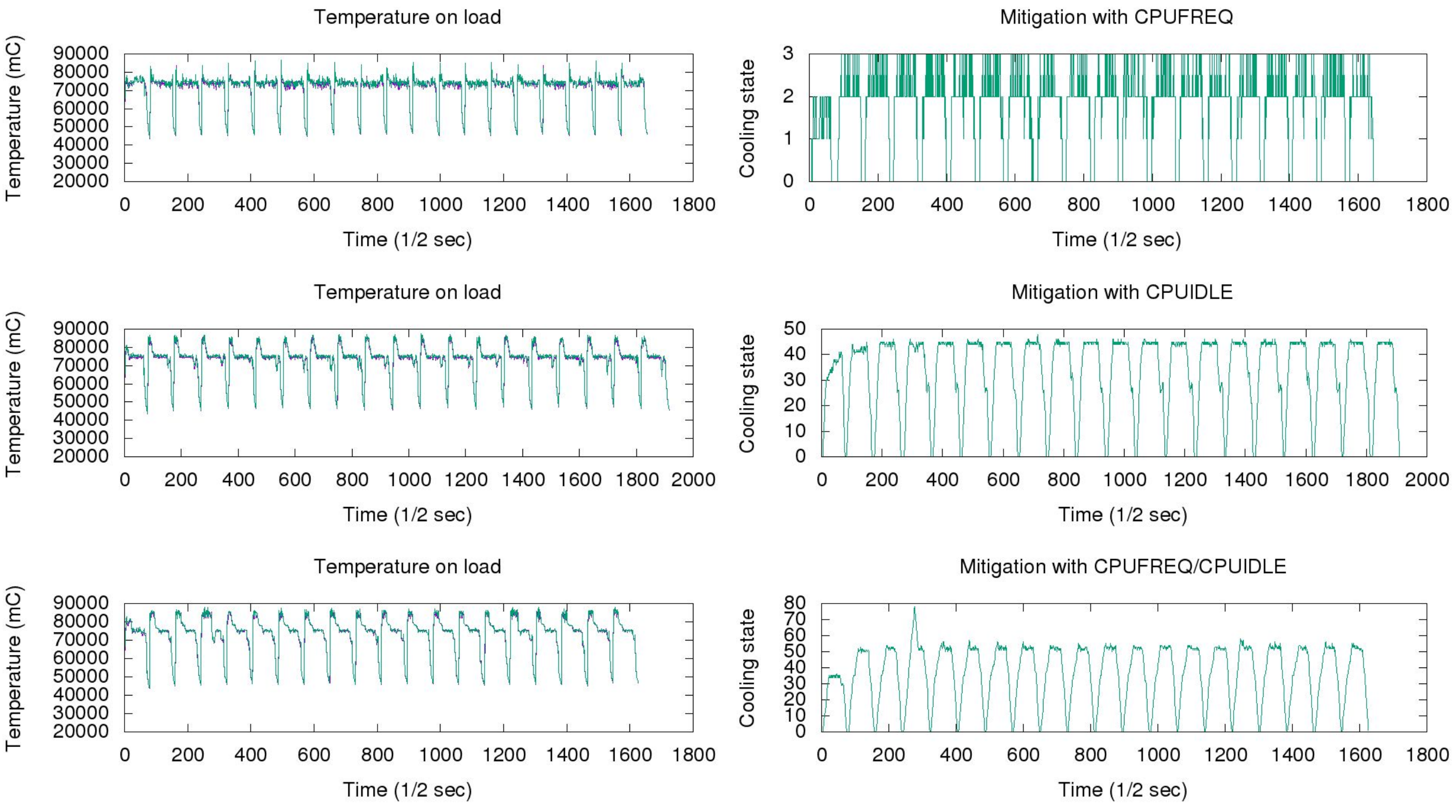
Compute capacity / Frequency

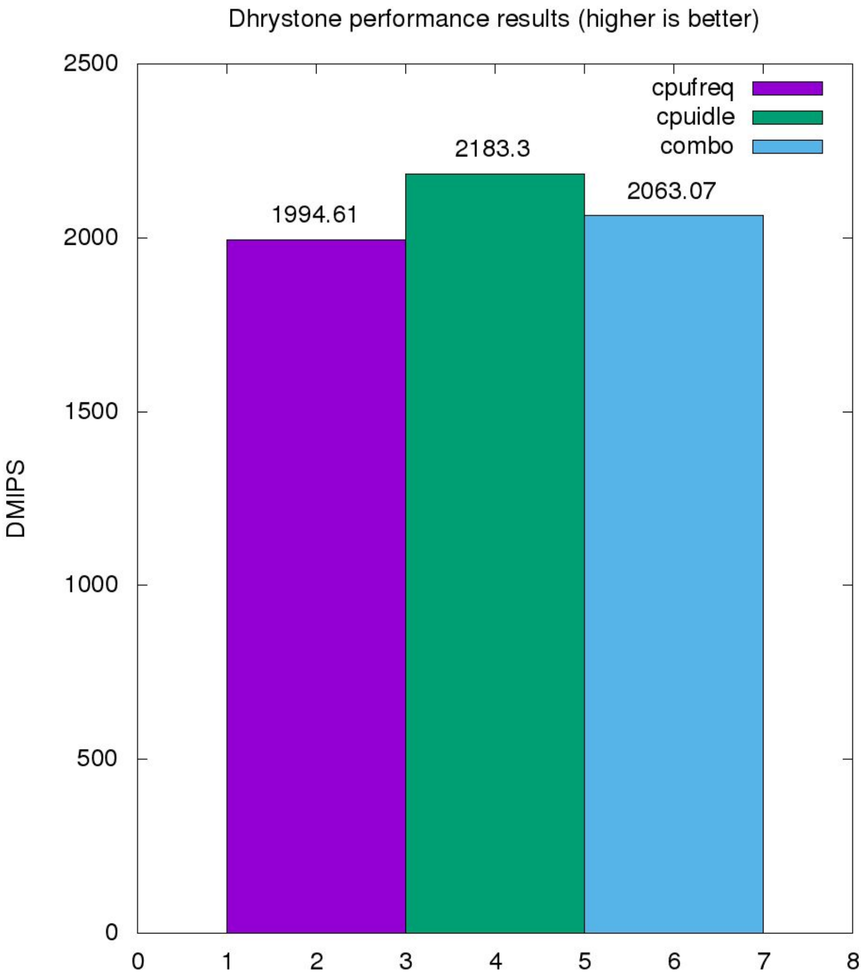
Frequency (MHz)	Capacity
2362	1024
2112	916
1805	783
1421	616
903	391

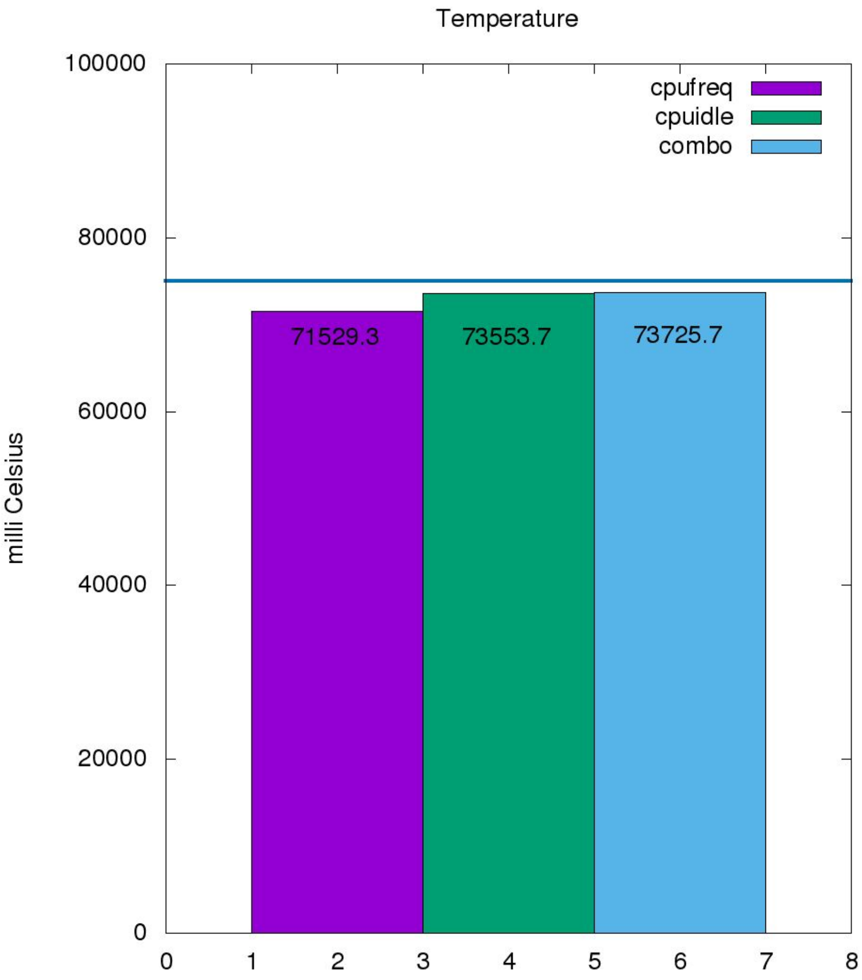
716

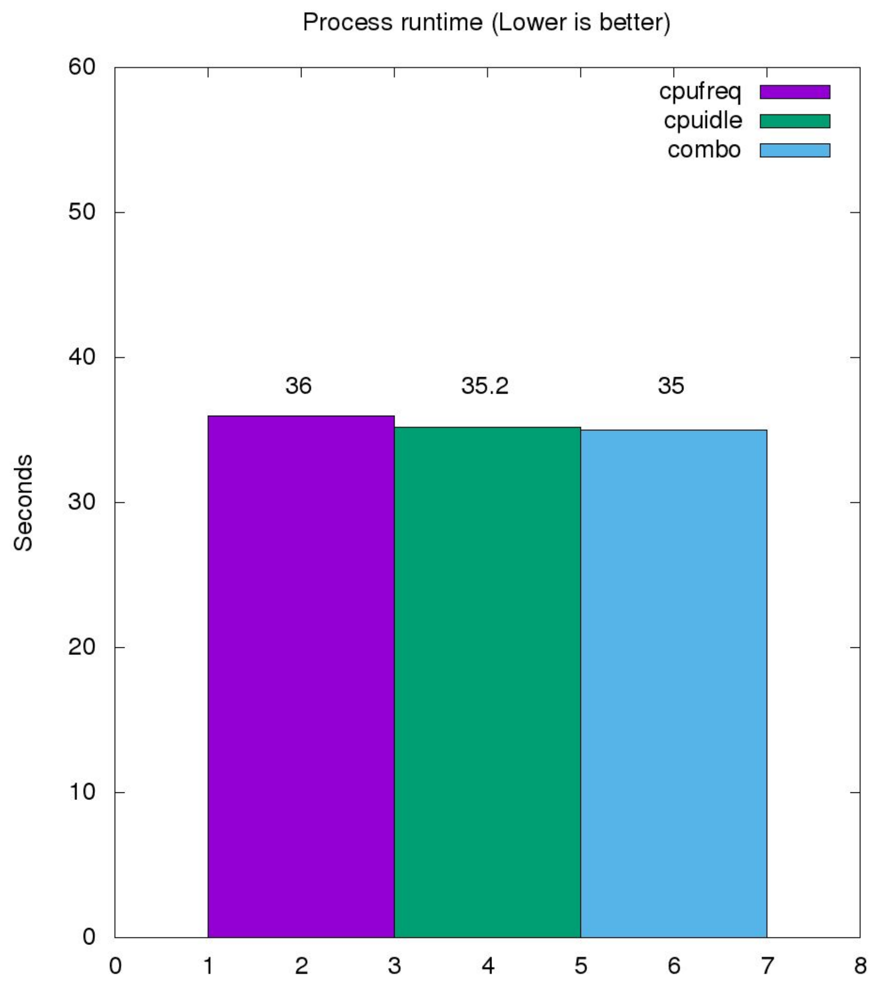

Capacity vs power
2400 MHz
2200 MHz
1800 MHz
1400 MHz
900MHz
Power mW
Capacity
0
1000
2000
3000
4000
5000
6000
0
200
400
600
800
1000

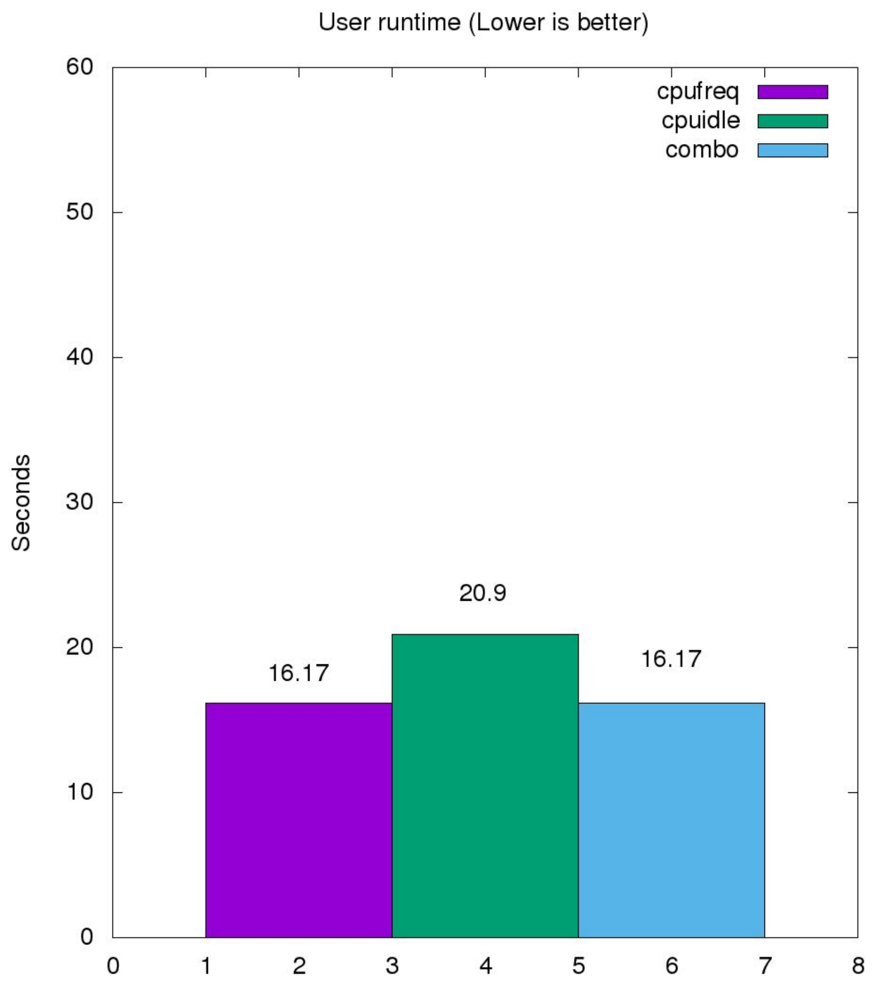
Mixing mitigations




Dhrystone performance results (higher is better)
DMIPS
2500
2000
1500
1000
500
0
0 1 2 3 4 5 6 7 8
cpufreq
cpuidle
combo
1994.61
2183.3
2063.07


Temperature
milli Celsius
100000
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8
cpufreq
cpuidle
combo
71529.3
73553.7
73725.7


Process runtime (Lower is better)
Seconds
cpufreq
cpuidle
combo
36
35.2
35
0
10
20
30
40
50
60
0
1
2
3
4
5
6
7
8


User runtime (Lower is better)
Seconds
cpufreq
cpuidle
combo
16.17
20.9
16.17
0
10
20
30
40
50
60
0
1
2
3
4
5
6
7
8

Conclusion

- We have slightly better performance and mitigation results than cpufreq
- In addition we drop the static leakage

Improvements

- Tested on board with slow idle states (3.2ms)
- Idle state numbers may be wrong
- Make sure it is compatible with the thermal pressure incoming patchset
- Mathematical proof
- Investigate if there is still room for more performance

Appendix

Introduction

- Mobile is an effervescent ecosystem, competing by innovation

- New CPU intensive user experience proposed :
  - Augmented reality
  - Virtual reality
  - High resolution movies
  - Image recognition
  - ...
- More performances ⇒ more power ⇒ higher temperature
- How to prevent high temperature without dropping performances

Thermal behavior of recent boards

- Recent hardware has high performances, up to 2.4GHz
  - The higher the frequency, the higher the voltage which is squared
- The temperature transitions are very fast
  - Monitoring the temperature is also CPU intensive
  - We consume power to save power
- The heat capacity of the SoC is quickly reached

Thermal runaway phenomena

Temperature threshold where the semiconductor resistance decreases, thus letting more current to pass and consequently heating more.

- Critical situation, silicon can die if there is no proper protection (kernel, firmware, hardware)
- Even if the dynamic consumption is zero, static leakage can raise this phenomena
- Solution ⇒ power down the component

A brief overview - Passive cooling devices

- devfreq: Use DVFS for devices other than the CPUs
- clock: reduce clock for any device
- cpu: Use DVFS for the CPUs (ARM)
- power clamp: (Intel specific) inject idle periods to reduce the power

Observation

- Mitigation but at the cost of dropping a lot of compute capacity:
  - Cpufreq stats is oscillating between 1.4GHz and 1.8GHz
  - Compute capacity is dropped between 30% and 40%

- Heat capacity is not restored

- Static leakage power still there

A new cooling device

- We want:
  - Drop performances
  - Drop static leakage
  - Restore the thermal capacity
- Instead of reducing the frequency, force the CPU to do nothing
- Synchronize the CPUs together to be idle
- Shutdown the components in order to let them to cool down

Powercap idle injection

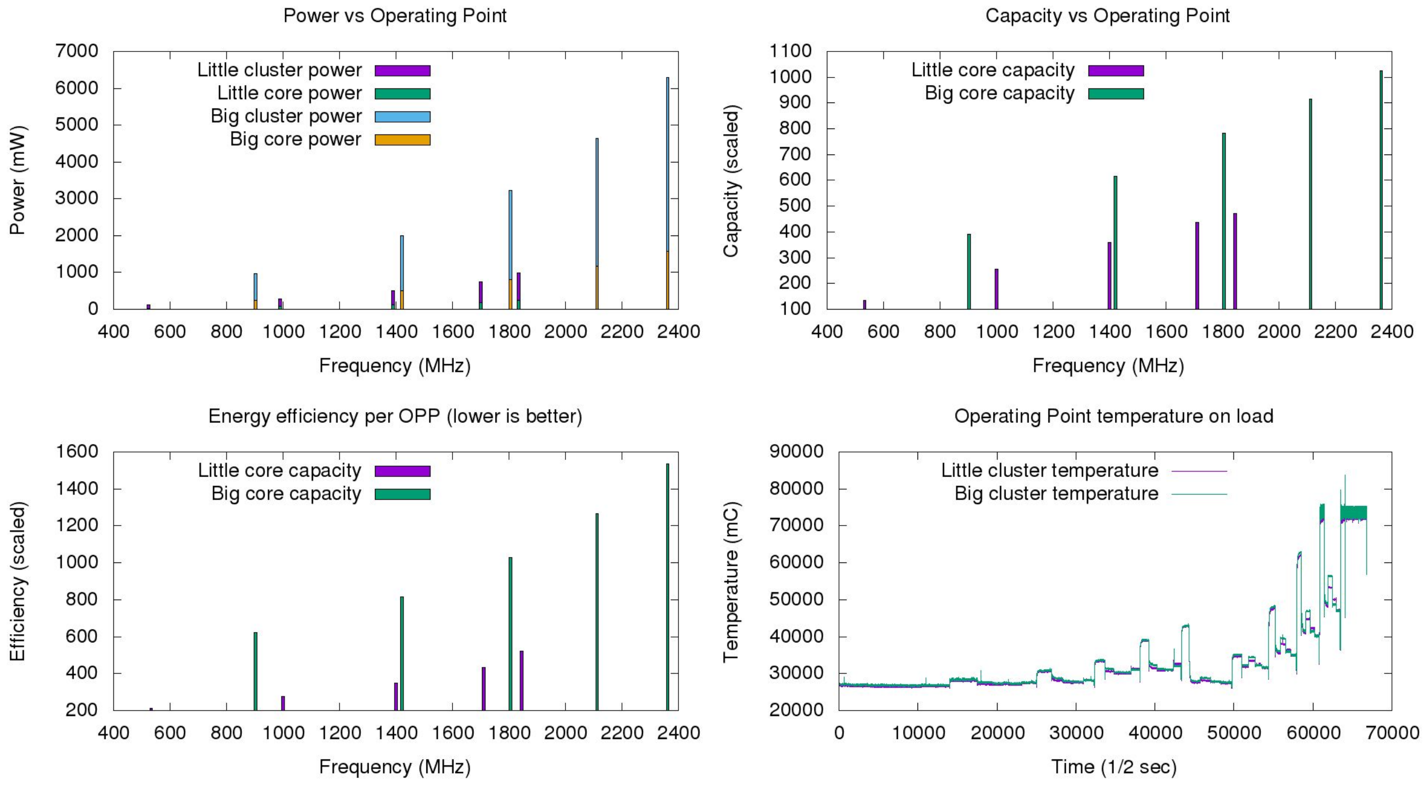
- Framework for idle injection merged upstream
  - drivers/powercap/idle_inject.c
  - uses smpboot kthread API

- Simple API
  - [un]register a set of CPUs
  - Set idle and run durations
  - Start / Stop

- Rely on play_idle

  ⇨ deepest enabled idle state (usually cluster off state)

Compute capacity / Energy / Frequency



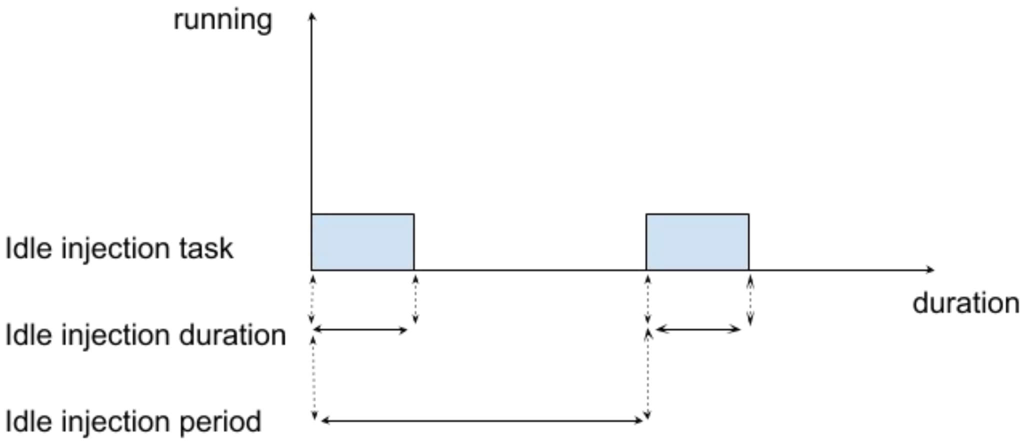
Compute capacity / Energy / Frequency

- The micro architecture has an impact on the consumption
- The energy consumption increase exponentially for high compute capacity
- In turn the mitigation reduce the compute capacity

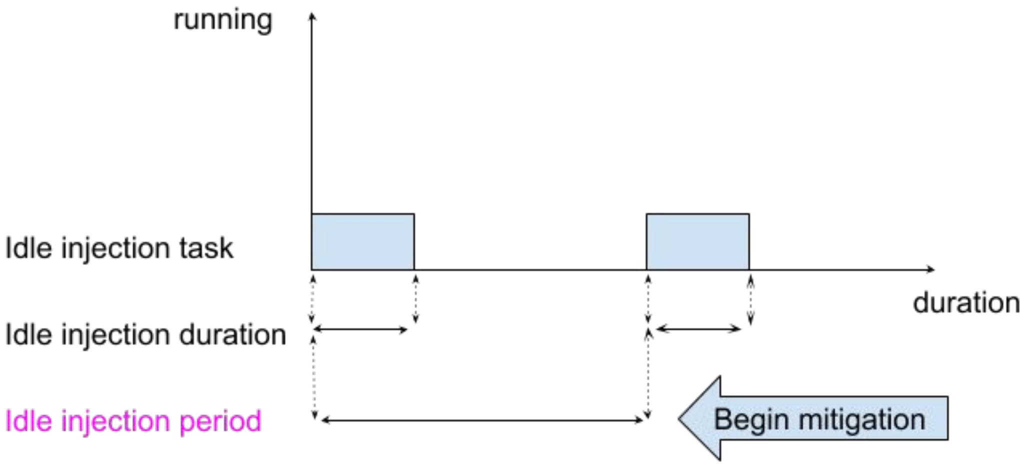
CPUidle cooling device

- Relies on the powercap idle injection
- Provides 100 states
- Temperature increases ⇒ more idle cycles
- Temperature decreases ⇒ less idle cycles
- No mitigation ⇒ no idle cycles
- Maximum mitigation ⇒ 100% idle

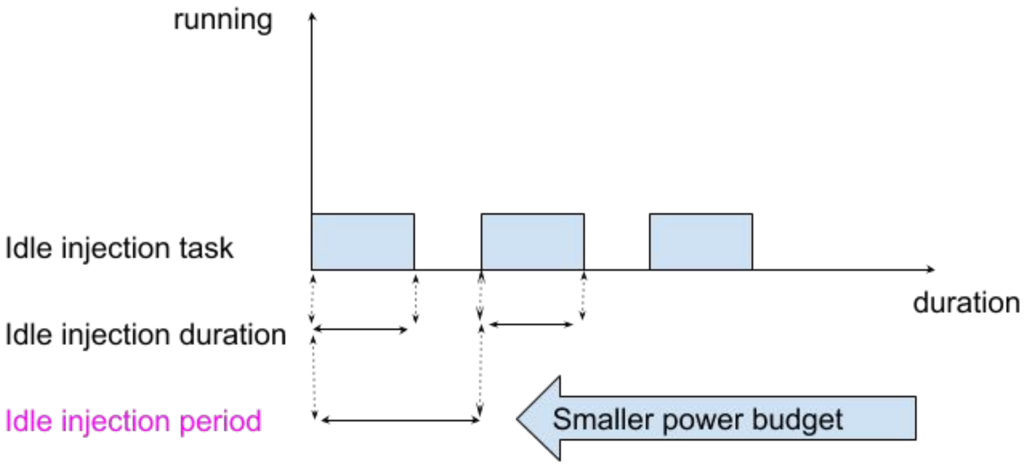
Idle injection (1)



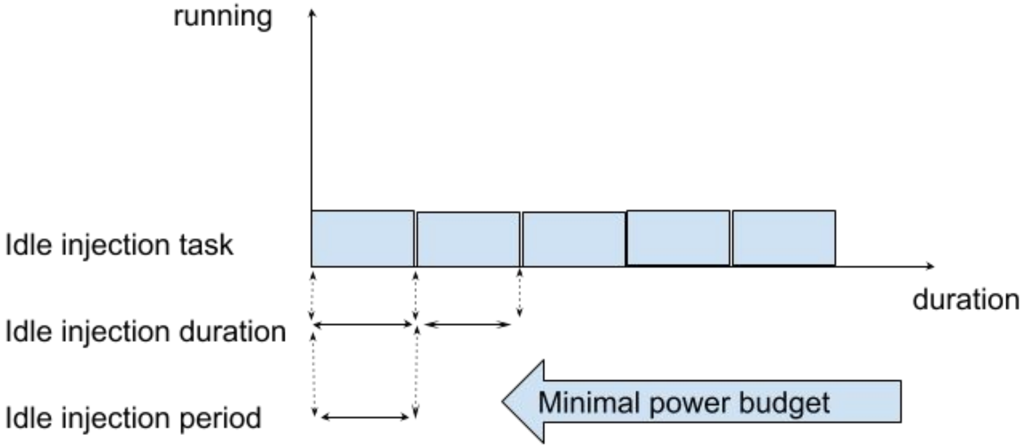
Idle injection (2)



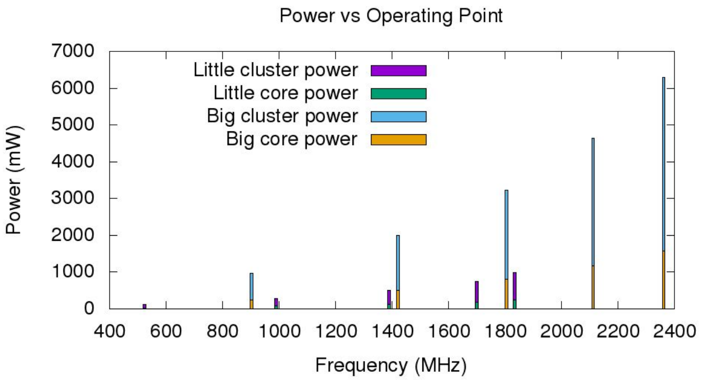
Idle injection (3)

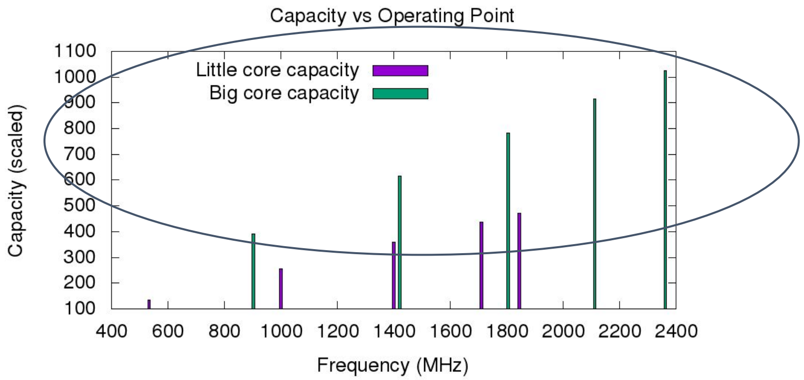


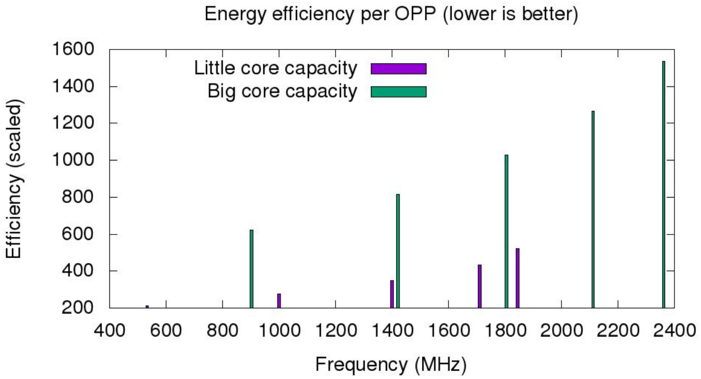
Idle injection (4)

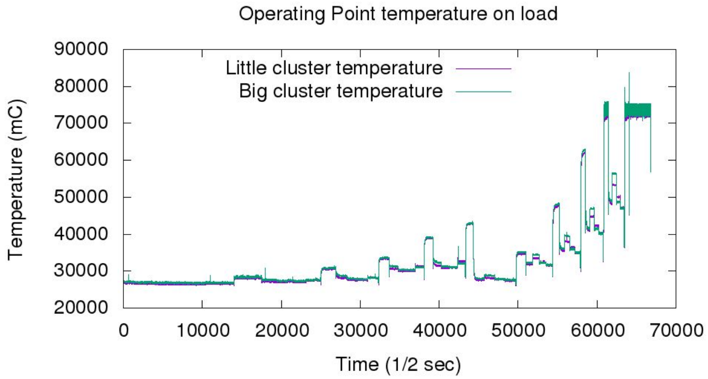


Compute capacity / Energy / Frequency

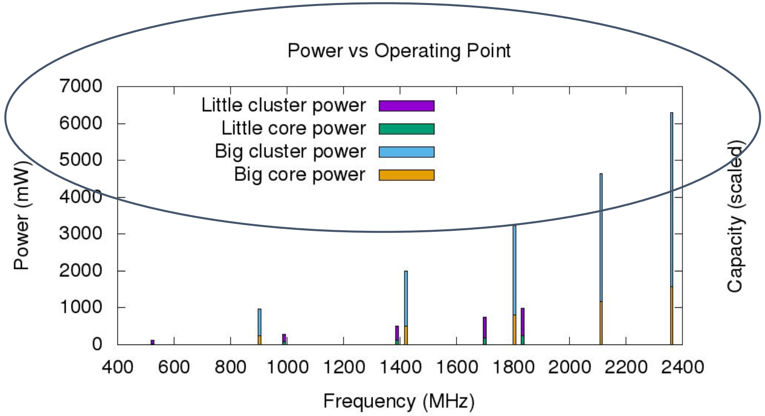


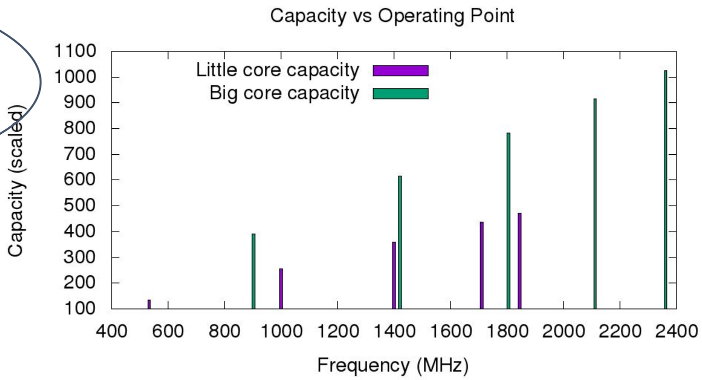


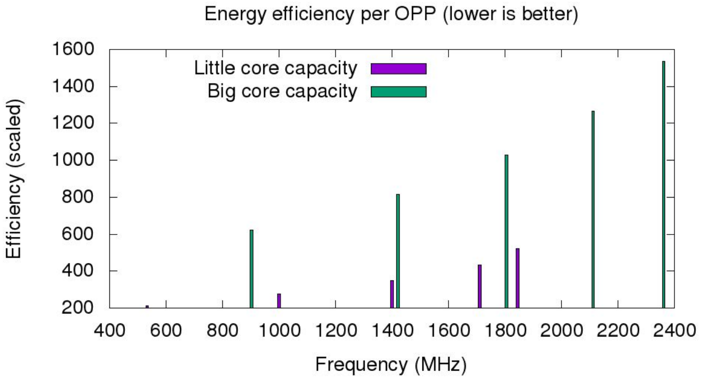


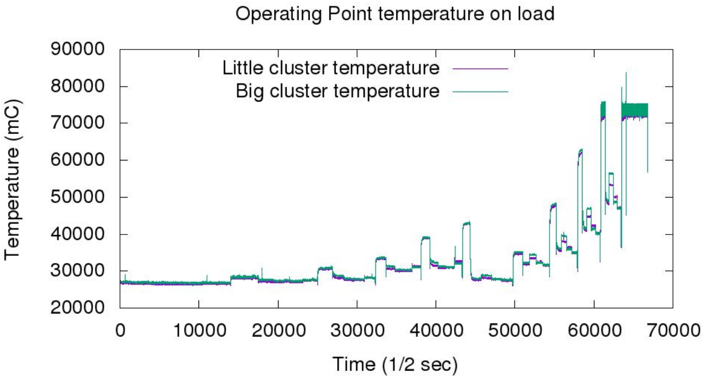


Compute capacity / Energy / Frequency





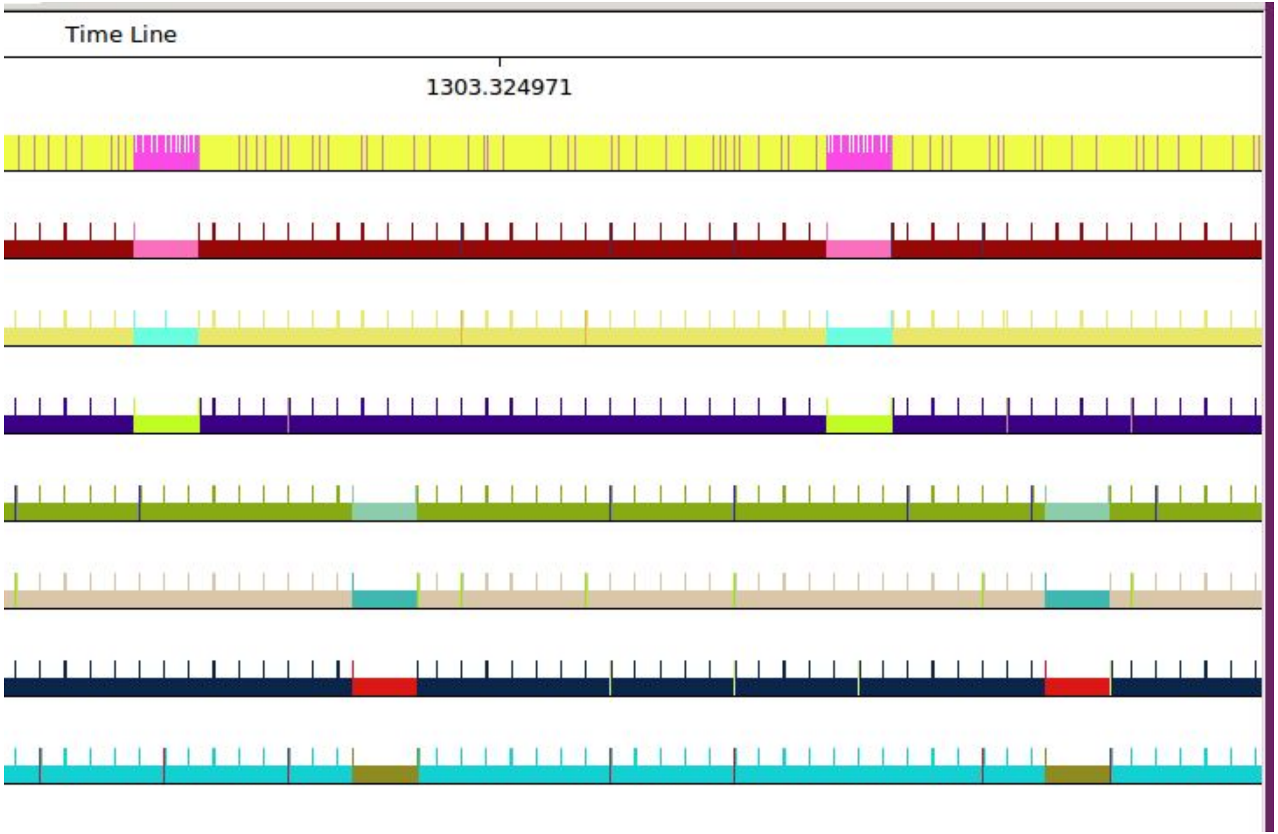




Compute idle injection cycles

- Ratio = (1024 - 916) * 100 / 1024 = 10.55 %
  - If more than 10.55 % of idle injection is needed
    - 0% idle injection
    - OPP_max - 1
- Ratio = (916 - 783) * 100 / 1024 = 13 %
  - If more than 13% of idle injection is needed
    - 0% idle injection
    - OPP_max - 2
- Etc ...

The implementation



Conclusion - idle injection

- Idle injection is useful to:
    - Mitigate temperature
    - Drop static leakage
- Idle injection adds latency
- Should be considered as an alternative if cpufreq is not available on the platform
- Could be used as a secondary cooling device

High OPP costs

- Highest OPP (2.36 GHz) with 30% idle injection consumes:
  - 6288 mW x 0.70 = 4400 mW
- OPP (2.1 GHz) consumes : 4648 mW
- OPP (1.8 GHz) consumes : 3220 mW

We can have the same compute capacity for less power