WHITE PAPER

Intelligent Power Allocation

Maximize performance in the thermal envelope

Xin Wang March 2017

ARM DTO 0052A

Contents

Release Information

The following changes have been made to this White Paper.

Document History

Date	Issue	Confidentiality	Change
March 10, 2017	**A**	Non-Confidential	First release

Non-Confidential Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

http://www.arm.com

1 Introduction

The modern *System-on-Chip* (SoC) has higher thermal dissipation than its previous generations, because of the following factors:

- Increasing processor frequencies.
- Decreasing SoC package and device sizes.
- Higher levels of integration.
- Static power consumption trends with the most advanced SoC fabrication[1].

Faster frequencies mean faster switching, which means more power consumption and more heat dissipation. Smaller size means smaller thermal mass, which makes it more difficult to transfer heat quickly. As a result, power density has increased rapidly, and SoC temperature has now become a fundamental design bottleneck. Therefore, you need a thermal management solution to keep an SoC within a defined temperature range.

Most ARM-based systems do not have fans. Therefore, you must take special care to keep system temperatures within acceptable bounds.

To address this challenge, ARM has developed *Intelligent Power Allocation* (IPA) to keep an SoC within its thermal envelope by dynamically tuning the power allocation according to the current temperature.

IPA uses passive cooling to control the temperature of the hardware without using fans, by carefully regulating the power being supplied to the system.

The power consumed by the device depends on the running frequency and voltage. You can work out the power by using the following equation:

$$P = C * f * V^2 + Pstatic$$

By controlling the frequency and voltage, you can change the power consumption of the circuit, which implies the device temperature change as well. This is referred to as *Dynamic Voltage and Frequency Scaling* (DVFS) [2]. DVFS is a technique used by IPA to reduce or increase the core power and performance to match the expected core workload. It uses two major subsystems in the Linux kernel: CPUfreq and Devfreq.

These subsystems consist of governors and vendor-specific drivers, which manipulate the frequency and voltage. The CPUfreq framework is used for CPU cores, while the Devfreq framework is used for other devices, including GPU cores. This is the interface where vendors can add support to IPA for a new SoC and device drivers.

The section describes how operating systems behave in two situations:

- Running CPU-intensive tasks without any thermal constraints.
- Running CPU-intensive tasks and controlling the temperature to keep it under a predefined threshold.

When there are no thermal constraints, the operating system increases the frequency of a particular device when high computation power is needed. However, increased power consumption means increasing the temperature of that device.

The following figure shows how the CPU frequency changes over time in a situation when there are no thermal constraints. The CPU frequency reaches the highest level so the temperature of the SoC also rises.

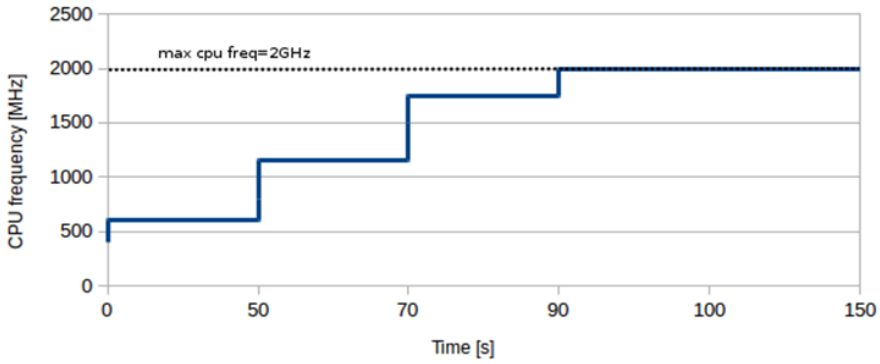

Figure 1 CPU frequency changes in time with no thermal constraints

In IPA, the operating system is aware of the thermal effects and can control it. The following figure shows how the CPU frequency changes over time when there is a temperature limit.

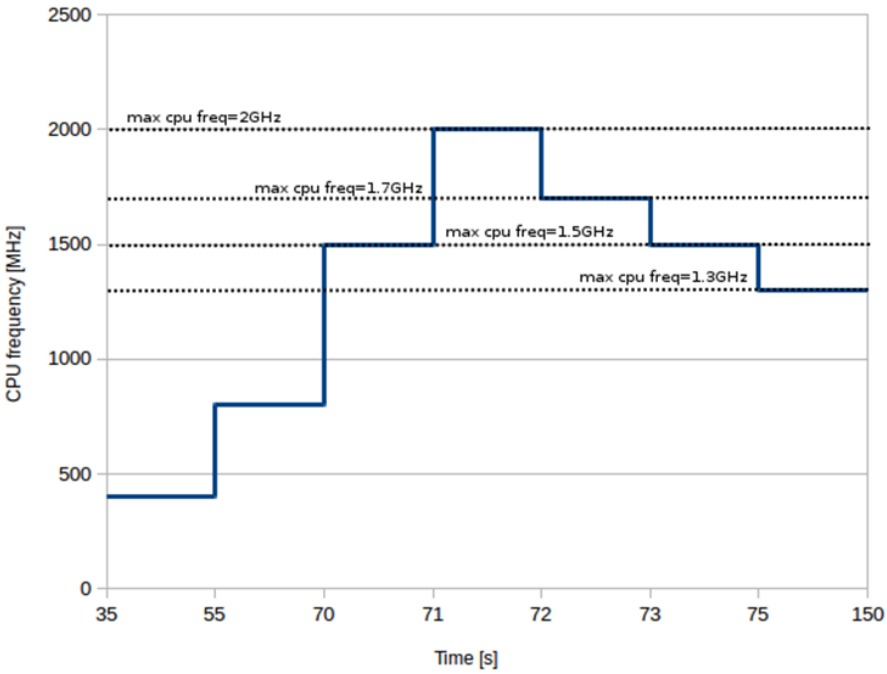

Figure 2 CPU frequency changes in time under thermal constraints

You can manipulate the maximum CPU frequency value, `max_cpu_freq`, to limit the temperature of the SoC. This new `max_cpu_freq` value is widely used by the system components, and no other higher value is allowed at that time. This is the proper behavior of the running system because it does not increase the temperature above some well-defined limit.

In the Linux kernel, the mechanism that is responsible for the temperature constraints is called *thermal framework*. The thermal subsystem consists of thermal zones, temperature sensors, cooling devices, and governors. IPA is the thermal governor, which maintains the temperature value for each thermal zone.

The thermal zone is an abstraction, which provides a container for some devices and the temperature sensor. It is initialized from the device tree, but devices can be added or removed to the zone anytime.

The temperature of a thermal zone is controlled by reducing the power consumption of the devices pinned to that zone. The interface to the physical device is provided by an abstraction called *cooling device*. Each cooling device defined in the system is described with some additional fields, such as contribution, link to the real device or software routine, static and dynamic power coefficient. It is referred to as *power model*.

For example, cooling devices can include, but are not limited to, the following:

- CPU clusters.
- GPUs.
- *Image Signal Processor* (ISP).
- Video Accelerator.

1.1 Power consumption scenarios

For an SoC, power consumption can be classified into the following types of scenarios:

- Short-term responsiveness
- Sustained workload
- Energy constrained use cases

A successful thermal management solution maximizes user experience for the first two power consumption scenarios: Short-term responsiveness and sustained workload.

1.1.1 Short-term responsiveness

In this scenario, power consumption can significantly exceed the sustainable maximum power of an SoC for a relatively short period to increase system responsiveness. For example, this scenario is common in use cases, such as web browsing, *User Interface* (UI) updates, and *High Dynamic Range* (HDR) image processing to improve user experience.

In a web browsing use case, there are bursts of requests for high performance from time to time. These requests can still be satisfied even when the power value crosses the 'sustainable power limit' threshold. The limit is the time period when the device consumes this power because the temperature rises quickly. The following figure shows the power consumption profiles of mobile web browsing on a Quad Cortex-A7 platform.

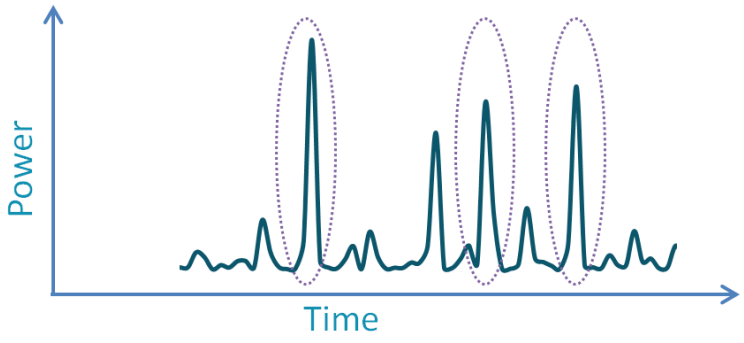

Figure 3 Power consumption profiles for web browsing

1.1.2 Sustained workload

In this scenario, the performance is maximized at or within the sustainable power envelope. This scenario adapts performance allocation to major power consuming entities. For example, this scenario is common in gaming, running long benchmarks, video processing, and multi-shot HDR.

Mobile users spend 32% of time on gaming. The following figure shows the power consumption profiles of mobile gaming on a Quad Cortex-A7 platform.

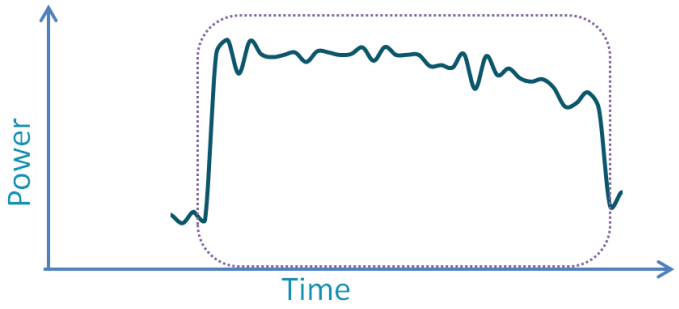

Figure 4 Power consumption profiles for gaming

1.1.3 Energy constrained use cases

This scenario consumes little power, and operates under the radar of IPA. For example, the scenario is common in low-power audio, message, and voice.

Mobile users spend 16% of time on audio, video, and other utility applications. The following figure shows the power consumption profiles of these applications on a Quad Cortex-A7 platform.

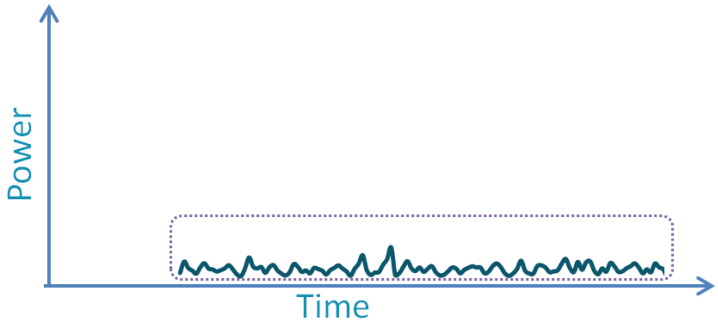

Figure 5 Power consumption profiles for audio, video, and other utility applications

1.2 Introduction to IPA

IPA is an algorithm that maximizes performance in a thermal envelope. It achieves this goal by combining thermal closed loop control and dividing power intelligently among components in the system.

The main principle behind IPA is based on a highly accurate closed loop *Proportional Integral Derivative* (PID) temperature controller and the power-division algorithm.

Based on the delta from the desired temperature, the thermal governor estimates available power budget. After that, this budget is intelligently allocated among cooling devices in the system to maximize performance while maintaining temperature control.

With power models, a power-allocation algorithm can map the allocated power to the voltage and frequency supplied to a cooling device, so that each cooling device consumes the allowed amount of power. This is a big improvement on existing Linux thermal governors, which do not have an accurate knowledge of the power used at different operating points.

The new distinction of the devices (called *actors*) helps to calculate more accurately the needed power for each of the actors. This is used by IPA to provide fast-reacting, reliable, and efficient algorithms to solve thermal and performance issues in a dynamic environment.

1.3 The status of IPA development

In 2013, ARM started developing a Linux OS implementation of IPA. This implementation shows key concepts and benefits that can be obtained in real workloads.

On March 3, 2015, the first full IPA patch was accepted into Linux, and was merged into Linux 4.2. All operating systems based on Linux (Version 4.2 or later) benefit from this, and no longer require patches to support IPA.

ARM intends to improve thermal management in the ecosystem, while maintaining the ability of our partners to usefully differentiate in this area.

2 IPA technology

2.1 PID controller

A *Proportional Integral Derivative* (PID) controller[3] is a control loop feedback controller commonly used in industrial control systems.

The PID controller is a closed-loop control system. It feeds the output of the system back to the input of the controller, so that it obtains an accurate or adaptive control. It evaluates the difference between the reference value and the output, and tries to minimize the difference to zero. Therefore, it can avoid various problems with the open loop control system, such as inability to respond to disturbances, and inability to adjust to different systems.

The PID controller is probably the most commonly used feedback control design. However, it cannot be used in complicated cases, especially for *Multiple Input and Multiple Output* (MIMO) systems.

PID stands for *Proportional*, *Integral*, and *Derivative*. These three terms describe the basic elements of a PID controller. Each of these elements performs a different task, and has a different effect on the functionality of a system. The following figure shows a block diagram of a basic PID controller.

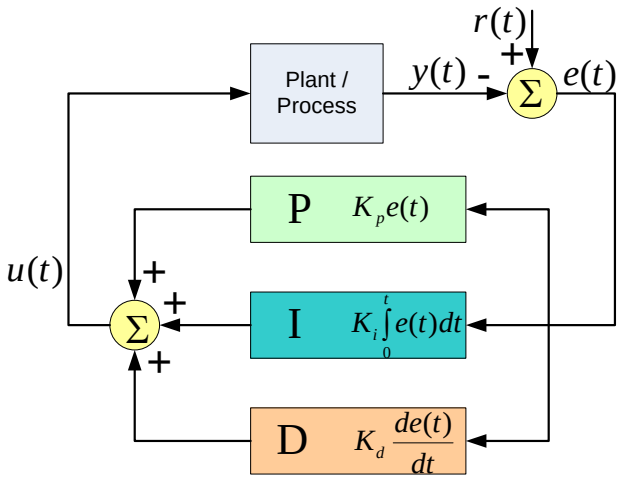

Figure 6 Basic PID Controller

In Figure 6:

- $u(t)$ is the control signal.
- $y(t)$is the actual output value.
- $r(t)$ is the reference value, also called the *setpoint*. It is the desired output value.
- $e(t)$ is the control error ($e(t) = r(t) - y(t)$). It shows the difference between the reference value and the actual output value.
- The term *Plant* refers to the object that is being controlled.

The control signal $u(t)$ is a sum of the following terms:

- The *P-term*: This term is proportional to the error. It acts on the present value of the error.
- The *I-term*: This term is proportional to the integral of the error. It represents the accumulation of past errors.

- The *D-term*: This term is proportional to the derivative of the error. It can be interpreted as a prediction of future errors, based on linear extrapolation according to the current rate of change.

The PID controller can be easily implemented and work well in most situations. You can tune the PID controller without knowing much about the control theory.

2.2 Power allocation algorithm

IPA is implemented as a thermal governor, which is referred to as *Power Allocator* in the Linux thermal framework.

The following figure shows a high-level view of how IPA works.

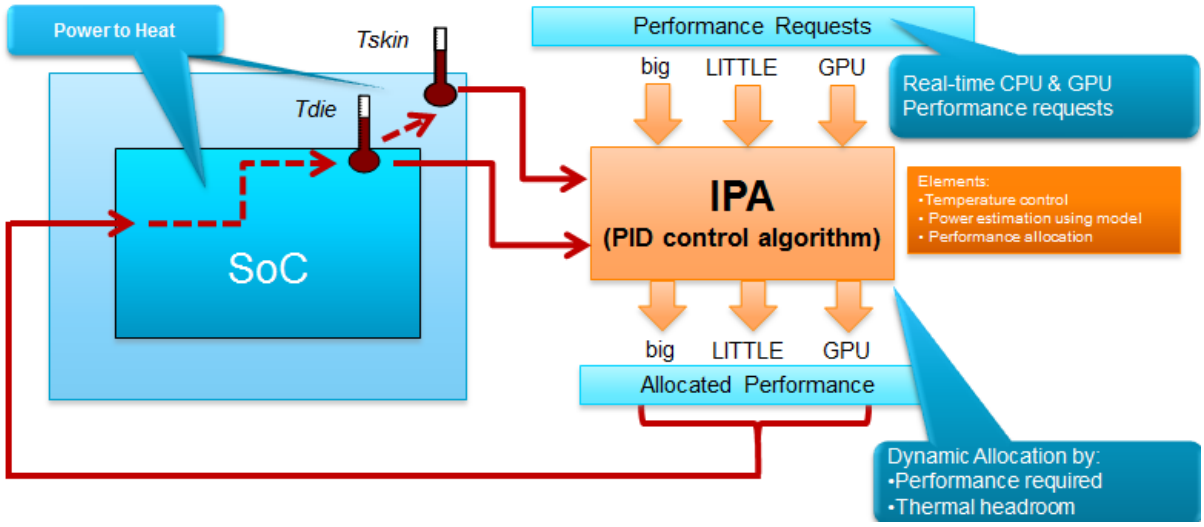

Figure 7 ARM Intelligent Power Allocation

You can choose different thermal sensors to monitor the SoC temperature. Currently, there must be only one sensor in the thermal zone, but the thermal zone can have multiple cooling devices. There can be different thermal zones as well. The cooling device is not directly connected to the sensor.

Compared with earlier solutions in the kernel, IPA provides more accurate power control, and allocates more power to cooling devices to achieve better performance while preserving the thermal envelope.

To maximize performance while maintaining temperature control, the governor dynamically allocates performance for cooling devices in the system, based on the following factors:

- Available power budget, a value computed according to the difference between a current temperature and the desired temperature.
- Performance requests from cooling devices.
- The power allocation policy, which allows you to define weights to bias performance towards a particular cooling device.

2.2.1 Thermal management approach

IPA uses a central Power Arbiter to keep a system within a thermal envelope and manage performance requests. The following figure shows the thermal management approach of IPA.

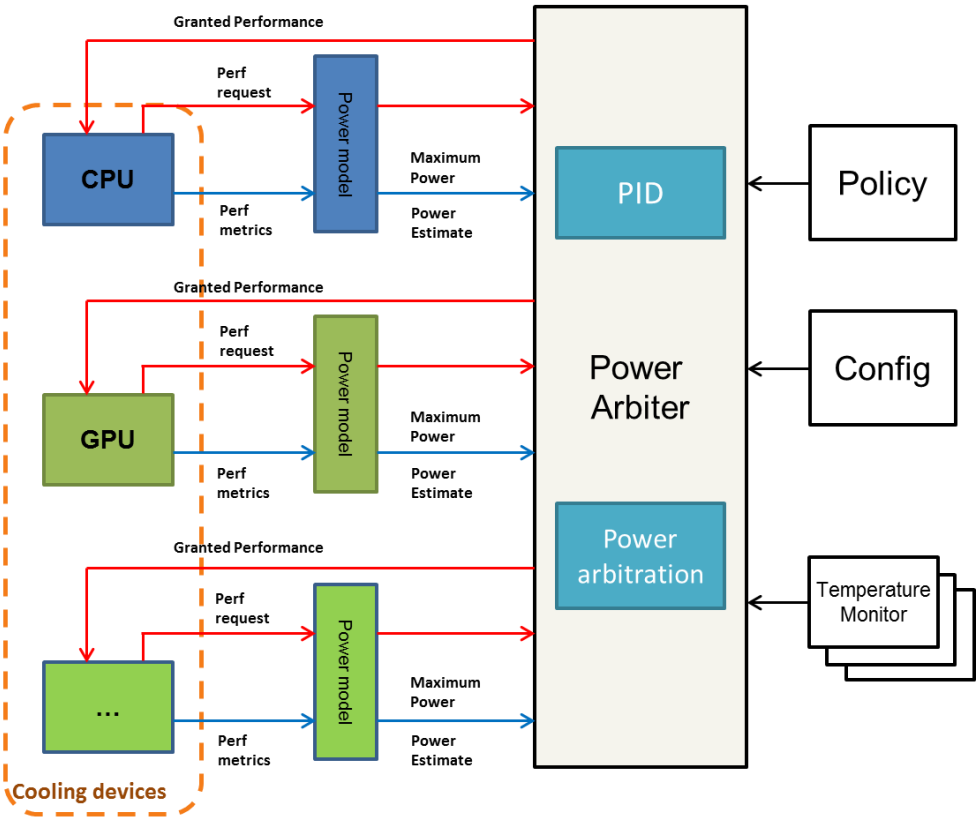

Figure 8 IPA thermal management approach

To keep the system within the thermal envelope, IPA uses the PID controller to dynamically allocate power budget, and allows a short term boosting by exploiting thermal headroom.

IPA manages performance requests from different cooling devices. Each cooling device can request different performance levels, and has a power model to estimate its power consumption and the impact of the performance request for the cooling device.

The Power Arbiter provides guaranteed minimum performance. You can configure the Power Arbiter with policies to allocate power among different cooling devices.

2.2.2 Power model

Each cooling device supports a power model. IPA uses the power model to determine the power allocation for the cooling device.

The power model sets frequency limits, applies other power policies, and represents the relative size of performance requests from the cooling devices. In other words, the power model represents the quality of performance allocation.

Power models can be divided into the following categories:

- Dynamic power models

Dynamic power models deal with dynamic dissipation and should reflect the impact of DVFS and actual running time.

- Static power models

  Static power models deal with static power dissipation, which essentially consists of the power used when the transistor is not in switching.

For details about dynamic dissipation and static dissipation, see the website[4].

Initially, you can implement only dynamic models for easy implementation, because dynamic power is usually the major component in ARM-based SoC designs. You can extend the implementation to include static power modeling if you anticipate a benefit in the target system.

An ideal dynamic power model must track the changes in power from program flow variation. However, DVFS affects power consumption more than program flow variation. Therefore, to build a simple power model for a CPU, the effect of the DVFS level and the running time are critical input factors to be considered.

Improving the accuracy of the power model can help improve the quality of power allocation. However, the actual power consumption still varies from what the power model predicts, regardless of how accurate the model is.

2.2.3 Power Allocator governor

The Power Allocator governor implements a PID controller with temperature as the control input and power as the control output, as shown in the following figure:

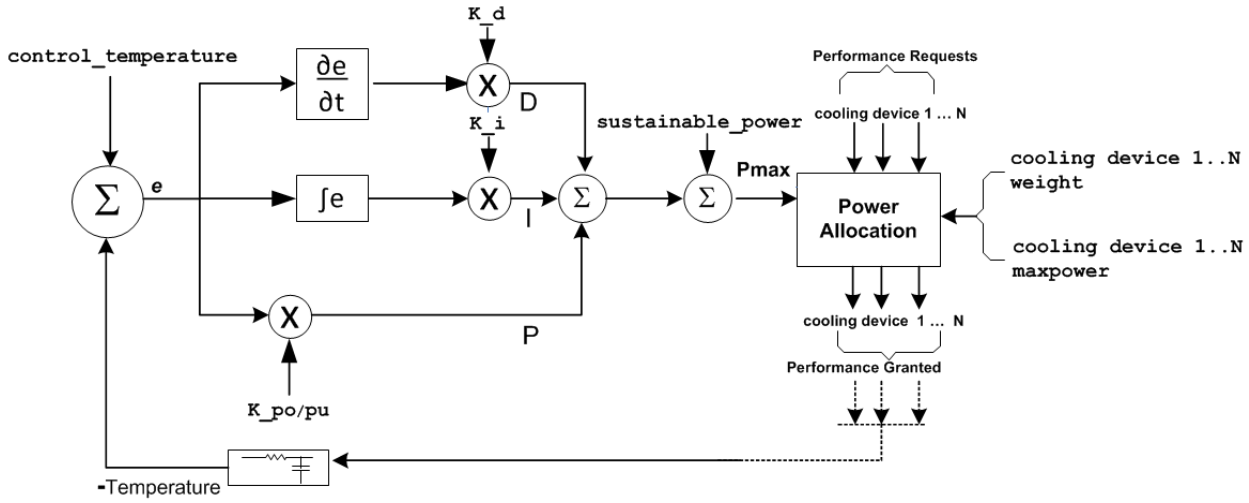

Figure 9 Power Allocator governor

$Pmax$ is the power budget, computed based on current temperature$Temperature$and desired temperature $control_{temperature}$.The value of $Pmax$ can be computed using the following formula:

$$Pmax = k_p * e + k_i * integral_{err} + k_d * diff_{err} + sustainable\_power$$

In this formula:

- $e = control_{temperature} - current_{temperature}$
- $integral_{err}$ is the sum of previous errors.
- $diff_{err} = e - previous_{error}$
- The proportional and integral terms offset the sustainable power.

  Sustainable power is the *Thermal Design Power* (TDP) of an SoC. For details, refer to the website at http://en.wikipedia.org/wiki/Thermal_design_power.

The Power Allocator Governor produces final output power, based on the following factors:

- Performance requests from cooling devices.
- The power budget $Pmax$.
- The power allocation policy.

2.2.4 Power allocation policy

Each cooling device is allocated with a share of the power budget, depending on the proportion of the device's requested power in the total requested power.

When each cooling device receives less than its maximum power, for example, if the power requested by a device accounts for 20% of the total requested power, this device is allocated with 20% of the total power budget.

When a cooling device receives more than its maximum power, its surplus power budget is re-divided up among other devices. Other cooling devices receive a share of the extra power that the cooling device requested. For a given cooling device, the share of extra power it receives depends on the proportion of this cooling device's extra power in total extra power.

In short, every device that can consume more power is granted by the same proportion of its available extra power. For example, there is a free extra amount of power 40W, ready to be consumed by some capable cooling devices. There are 3 devices with capable extra power:

- Device 0 can consume 20W.
- Device 1 can consume 80W.
- Device 3 can consume 60W.

The total extra power that the three devices can consume is 160W. Each of them gets extra power, which can be calculated using the following equation:

$$capable\ device\ power\ *\ (free\ extra\ power\ /\ sum\ of\ capable\ extra\ power\ of\ devices)\ =\ new\ extra\ power\ for\ this\ device.$$

Each of the three devices gets extra power calculated as follows:

- Device 0: 20 * (40/160) = 5W
- Device 1: 80 * (40/160) = 20W
- Device 2: 60 * (40/160) = 15W

The total extra power that the three devices receive is, 40W=5W+20W+15W. This number is correct because there is 40W of free extra power.

In summary, every device gets the same percentage of its own capable extra power, which has an impact on the whole system performance. Allocating extra power improves the whole system performance.

Power reallocation enhancement

ARM is proposing a new algorithm enhancement, which adds a cooling devices awareness feature to power allocation. You can find the patches at the following locations:

- http://linux-arm.org/git?p=linux-thermal.git;a=commit;h=1ce9f67b4ef1fb849ca94bd125db448d0e973c19.
- http://www.linux-arm.org/git?p=linux-thermal.git;a=summary.

The new algorithm creates two groups of actors:

- cpufreq type devices.
- devfreq type devices.

Every cooling device is pinned to only one of these groups. The CPU devices are pinned to cpufreq, while all other devices, such as GPU and ISP, are pinned to devfreq.

This assumption solves a problem when an unused device such as the GPU is granted with large part of the free extra power while the other CPU device only gets a small part if there is some extra power for sharing (providing that the CPU devices are close to their max power limit and the GPU is far from its max power limit).

This is an undesirable situation when there is an intensive CPU calculation (i.e. on the big cluster), but the other CPU (the little cluster) cannot offload the overloaded CPU because its frequency is limited. The little CPU cluster cannot get a large part of the free available power because the GPU is taking most of it.

The total free available power is maintained by the group. The calculation is divided into 2 phases:

- Phase 1:

  The extra power in each group is shared among the group members.

- Phase 2:

  If there is still some extra power left because the group actors cannot consume it, it is shared fairly among all of the capable actors in the system. Therefore, it is the same idea as the previous algorithm in this phase.

In summary, this new algorithm is similar to the previous algorithm, but it is more accurate in granting power to devices. This has an impact on the whole system performance. Every device that can offload the overloaded device in the same group gets priority before other capable devices in consuming free extra power.

Also, there is a ‘weights’ setting used in power allocation. The weights variable is designed to enable you to bias the power allocation among cooling devices. Weights indicate the relative power efficiency of different cooling devices. Use a high weight to indicate high power efficiency, and power will get preferentially distributed to higher efficiency components.

If each cooling device has a weight of one, they are considered equal. Weights are useful, for example in a big.LITTLE system. For more information about the weights variable, see 3.2 Porting IPA to your platform.

3 IPA porting

This section describes how to port IPA to your platform.

3.1 Porting preparation

Linaro has integrated IPA into *Linaro Stable Kernel* (LSK), which can be found at https://git.linaro.org/landing-teams/working/arm/kernel-release.git.

For example, you can reference the `lsk-3.10-armlt-juno`, `lsk-3.18-armlt`, or `lsk-4.4-armlt` tag:

- For `lsk-3.10-armlt-juno`, choose `lsk-3.10-armlt-juno-20150804` or later tags.
- For `lsk-3.18-armlt`, choose `lsk-3.18-armlt-20151111` or later tags.
- For `lsk-4.4-armlt`, choose `lsk-4.4-armlt-20160824-experimental` or later tags.

For Linux kernel mainline branch, visit the website at http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/ and get it from Linux Version 4.2 or later releases.

Important files in the source are:

- `drivers/thermal/power_allocator.c`

  This file contains the implementation of the Power Allocator governor. For details about Power Allocator, see the document `Documentation/thermal/power_allocator.txt` in your kernel repository.

- `drivers/thermal/scpi-thermal.c`

  This file is the thermal driver used by the ARM Juno platform.

The GPU devfreq code for IPA and the Mali GPU driver are in the same folder `drivers/gpu/arm/Midgard/.`

Note: The source code might change, depending on the particular tag you access.

The GPU thermal driver is implemented by the file `/drivers/gpu/arm/midgard/backend/gpu/mali_kbase_devfreq.c`.

3.2 Porting IPA to your platform

To port IPA to your platform, perform the following steps:

1. Enable IPA in the Kernel build by enabling the following configuration options in the kernel build:

   - `CONFIG_THERMAL_GOV_POWER_ALLOCATOR`, which enables the Power Allocator governor.
   - `CONFIG_SCPI_THERMAL`, which enables the Juno platform thermal integration.
   - `CONFIG_DEVFREQ_THERMAL`, which enables thermal management for devfreq devices used by the Mali thermal driver.

- `CONFIG_MALI_DEVFREQ`, which enables the devfreq device for Mali.

2. Develop your own thermal driver and GPU driver for IPA.

   Reference the file `driver/thermal/scpi-thermal.c`, and perform the following steps to develop a specific thermal driver:

   Implement the sensor driver callback interfaces, which include the interfaces to get the sensor information and temperature.

   Provide your platform code with information about the characteristics of the static power and dynamic power.

   Specifically, provide the following information for the cooling device register function:

   - Dynamic power coefficient.
   - Static power callback function.

   a) Search all possible cooling devices that have the property `#cooling-cells` from the device tree data.

      A CPU cooling device uses the function `of_cpufreq_power_cooling_register` to register and bind it with a specific thermal zone device.

      A GPU cooling device uses the function `of_devfreq_cooling_register_power` to register and bind it with a specific thermal zone device.

   b) Use the function `thermal_zone_of_sensor_register` or `devm_thermal_zone_of_sensor_register` to register sensors for a specific thermal zone device.

3. Configure the device tree.

   For details about the device tree node, see the file `Documentation/devicetree/bindings/thermal/thermal.txt`.

   To design the device node for thermal devices, you can refer to the file `arch/arm64/boot/dts/arm/juno.dts`.

   The device tree defines:

   - Tripping Points value:

     Two Trip Points are defined, one for the control temperature and one for the Switch-on temperature, in the Flattened Device Tree (FDT).

   - Weight for different cooling devices:

The weight of different cooling devices represents the effectiveness of those devices. In a big.LITTLE system, big cores have less weight than the LITTLE cores to achieve higher power efficiency.

4 IPA processing in operation

IPA dynamically allocates power to each cooling device, based on its load. For example, the cooling devices of a system are CPUs (big and LITTLE) and GPU. The loads determine how power is divided among the CPUs (big and LITTLE) and GPU.

After you set the control temperature, IPA sets the limit of maximum frequencies for the CPUs and GPU. For each cooling device, its running frequency cannot exceed that limit. CPUfreq and devfreq subsystems are notified about these new limits, and should take an action if needed.

The following figure shows how the running frequencies change over time, depending on the loads of the cooling devices.

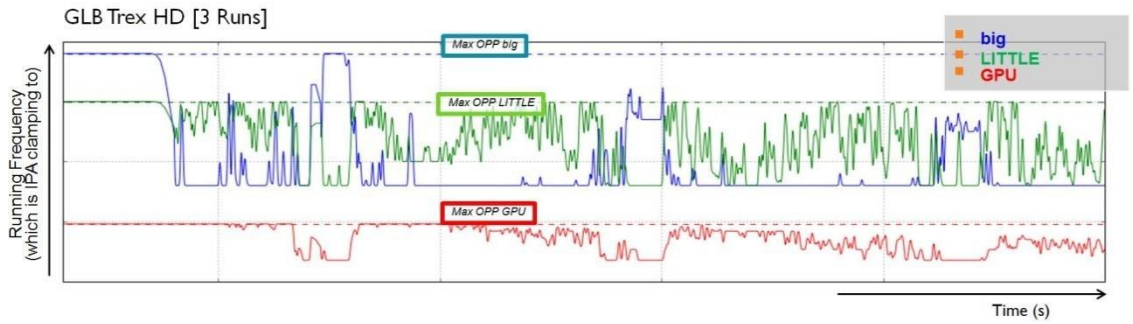

Figure 10 Running frequencies and loads

At the beginning of execution, the device temperature is below the switch-on trip point of IPA. When IPA is below the switch-on trip point, it is switched off and does not report running frequencies. That is why the running frequency is reported as the maximum at the beginning of the graph.

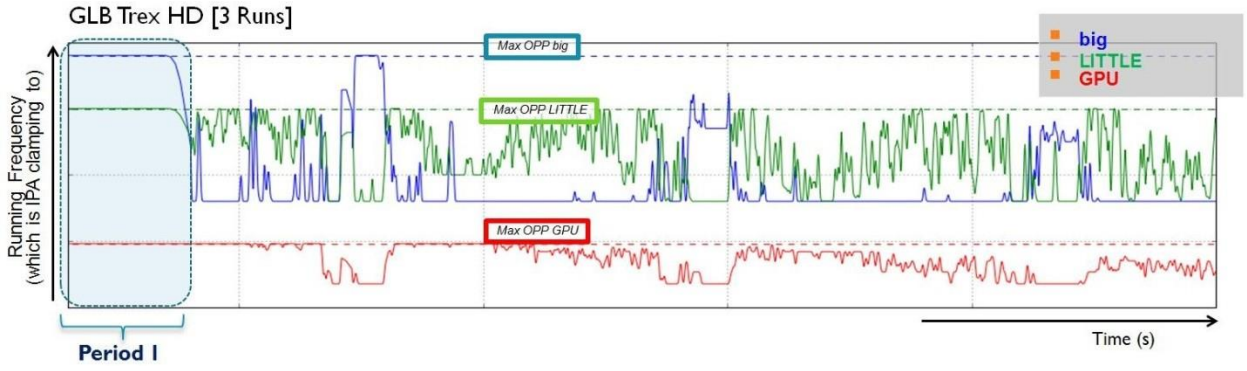

Figure 11 Running frequencies in Period 1

When the GPU load is high, most of the power is allocated to the GPU. As shown in the following figure, the running frequency of the GPU is high, while the running frequencies of the CPUs are low in Period 2 and Period 4.

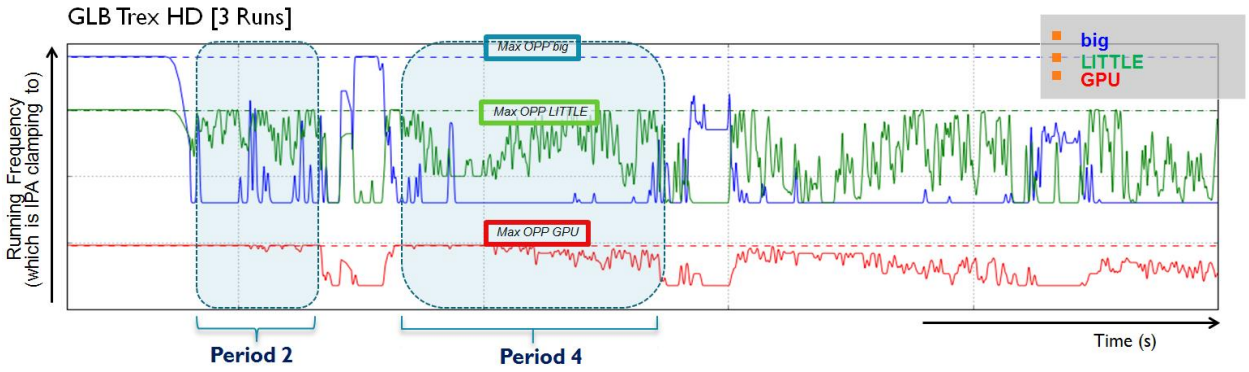

Figure 12 Running frequencies in Period 2 and Period 4

When the GPU load is low and CPUs load is high, most of the power is allocated to the CPUs. As shown in the following figure, the running frequency of the GPU is low, while the running frequencies of the CPUs are high in Period 3.

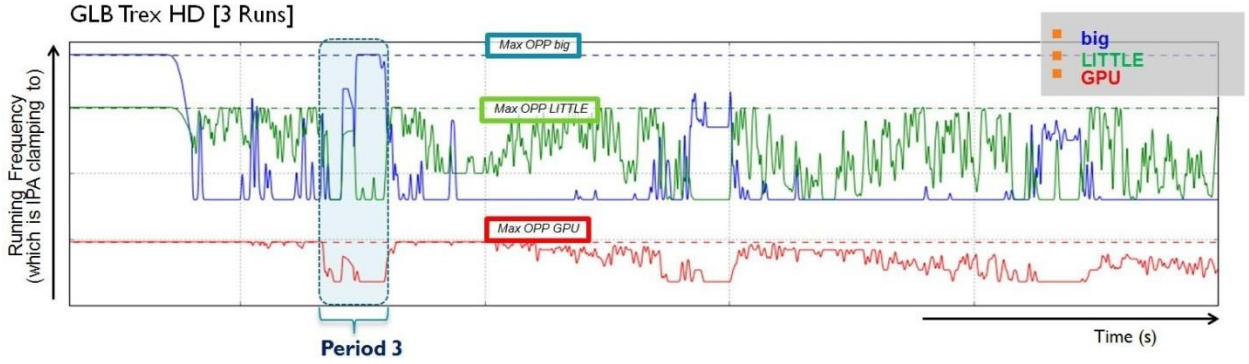

Figure 13 Running frequencies in Period 3

When the device gets hot, IPA reduces the available power for the cooling device to maintain temperature control. As shown in the following figure, the running frequency of each cooling device is lowered to maintain temperature control in Period 5.

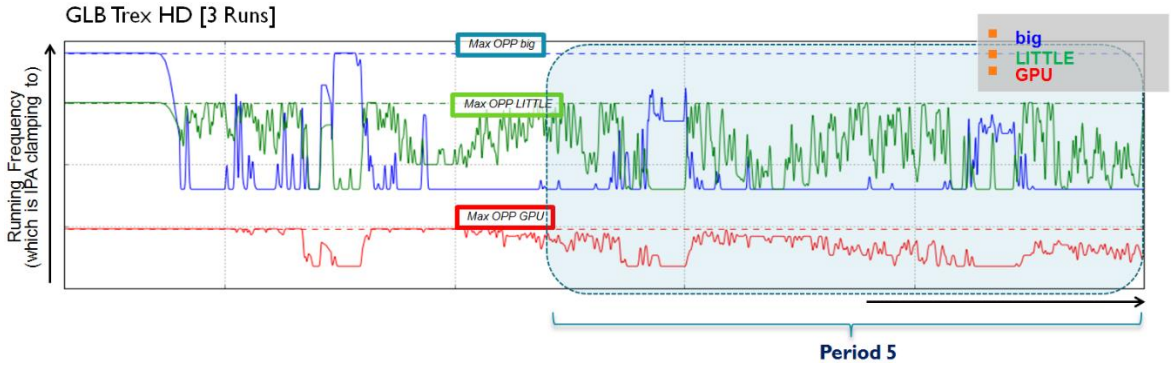

Figure 14 Running frequencies in Period 5

5 IPA tuning tools overview

The IPA tuning requires the following tools:

- Workload Automation Framework.
- IPython Notebook.
- Trappy.
- Bart.

5.1 Workload Automation Framework

Workload Automation (WA) is a framework developed by ARM. You can use it to execute workloads and collect measurements on Android and Linux devices. WA makes measurements reproducible and standardized for IPA tuning. It provides a method for collecting measurements from a device in a repeatable way.

WA is highly extensible. Most of its core functionality is implemented through plug-ins. It is easy to write new plug-ins to support new device types, workloads, instrumentation, or output processing.

WA is open-source software, and can be accessed from github at https://github.com/ARM-software/workload-automation.

5.2 IPython Notebook

The IPython Notebook is an interactive computational environment, in which you can combine code execution, rich text, mathematics, plots, and rich media.

To install this tool, use the `apt-get` command to install the packages `ipython-notebook` and `python-pandas`.

The package `python-pandas` is used for Trappy, and must be installed before installing Trappy.

5.3 Trappy

Trappy stands for *TRace Analysis and Plotting in Python*, and is a visualization tool to help analyze thermal Ftrace data generated on a device.

You can get this tool from https://github.com/ARM-software/trappy.

5.4 Bart

Bart stands for *Behavioral Analysis and Regression Toolkit*, and it is a tool based on Trappy. The primary purpose of this tool is to assert behaviors by using the Ftrace output for the kernel.

You can get this tool from https://github.com/ARM-software/Bart.

6 Results and conclusion

A popular thermal governor in Linux is Step Wise. To evaluate the benefits of the Power Allocator governor, experiments are conducted to compare Power Allocator with Step Wise, in terms of temperature control and performance.

Experiment information is as follows:

- One Odroid XU3 board, which has an Exynos 5422 SoC with the hardware configuration:
  - Four Cortex-A15 cores.
  - Four Cortex-A7 cores.
  - One Mali T628 GPU.
- Three cooling devices:
  - The Cortex-A15 cluster.
  - The Cortex-A7 cluster.
  - The GPU.
- Governors configuration:
  - For Power Allocator, the control temperature is set to 77 ºC.
  - For Step Wise:
    - The Cortex-A7 and GPU cooling devices are bounded to trip points at 77ºC.
    - The Cortex-A15 cooling device is bounded to the trip point at 74 ºC.

6.1 Temperature graphs

Figures 15 - 17 show how the temperatures change over time in three benchmark tests: Antutu, Egypt offscreen, and Geekbench. As shown in the figures, the temperatures controlled by Power Allocator are closer to the control temperature than the temperatures controlled by Step Wise.

Compared with Step Wise, Power Allocator provides more accurate power control, and allocates more power to cooling devices to achieve better performance.

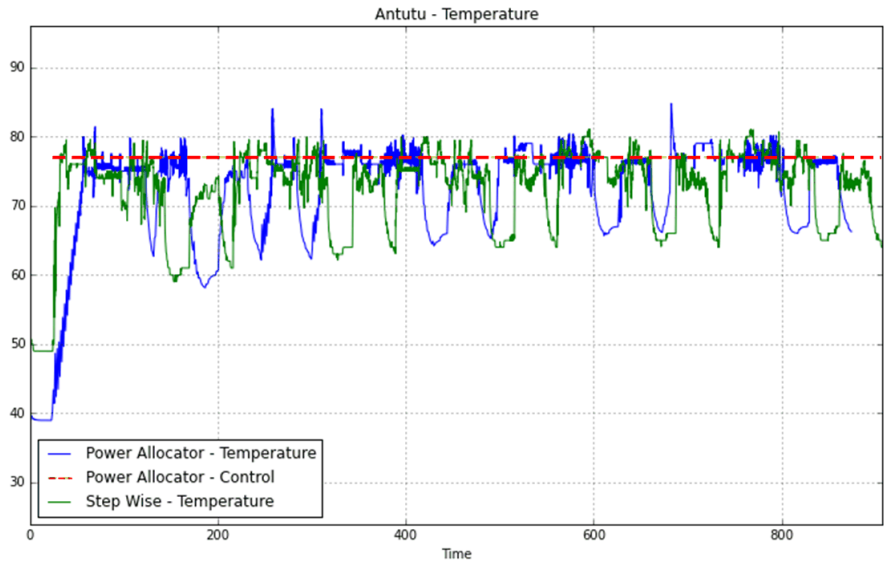

Figure 15 Antutu temperature

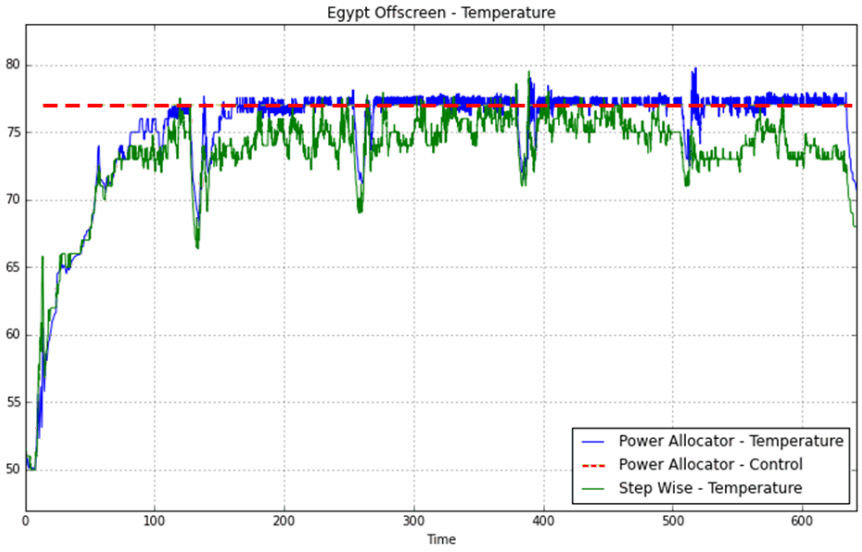

Figure 16 Egypt offscreen temperature

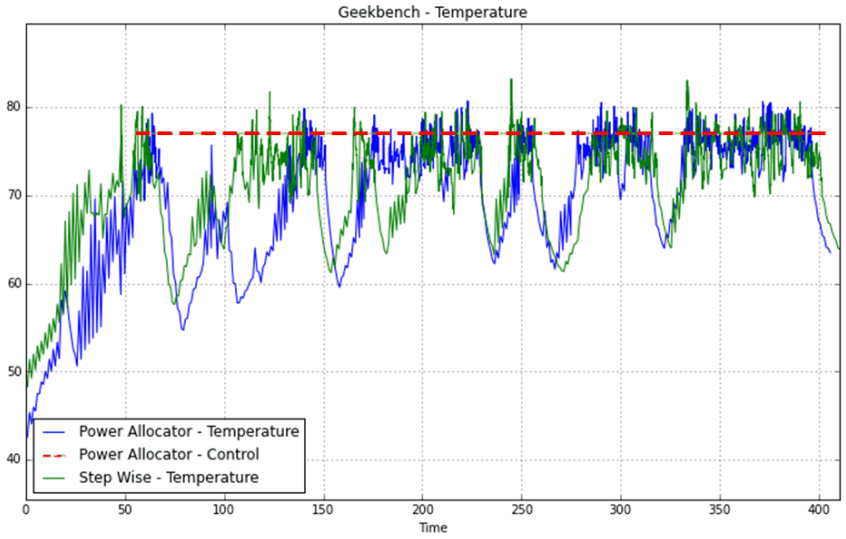

Figure 17 Geekbench temperature

6.2 Results comparison

Each of the Antutu, Egypt (offscreen), and Geekbench benchmarks is run five times and the results are used to compare Power Allocator with Step Wise, in terms of performance.

The following table shows the results comparison between Power Allocator and Step Wise for different benchmarks. The higher the number is, the better the performance.

Table 1 Benchmarks execution results comparison

Benchmarks	Power Allocator	Step Wise
Antutu	36381	36060
	36381	33801
	32900	32102
	30112	30666
	31120	30835
Egypt_offscreen	75	73
	65	60
	55	55
	55	53
	52	44
Geekbench	983	931
	998	948
	994	931
	970	876
	960	870

6.3 Deviation analysis

Histograms are used to show the probability distribution of the temperature. Compared with the histograms for Step Wise, the histograms for Power Allocator are more tightly packed with thinner tails. It indicates that Power Allocator has a better temperature control and less deviation than Step Wise.

The central tendency of the histogram for Power Allocator is closer to the control temperature than that for Step Wise. In other words, Power Allocator controls the temperature in such a way that it does not overshoot. In addition, Power Allocator optimally allocates power to maximize performance in the provided thermal budget.

Figures 18 - 20 show the temperature deviation comparison between Power Allocator and Step Wise in different benchmark tests.

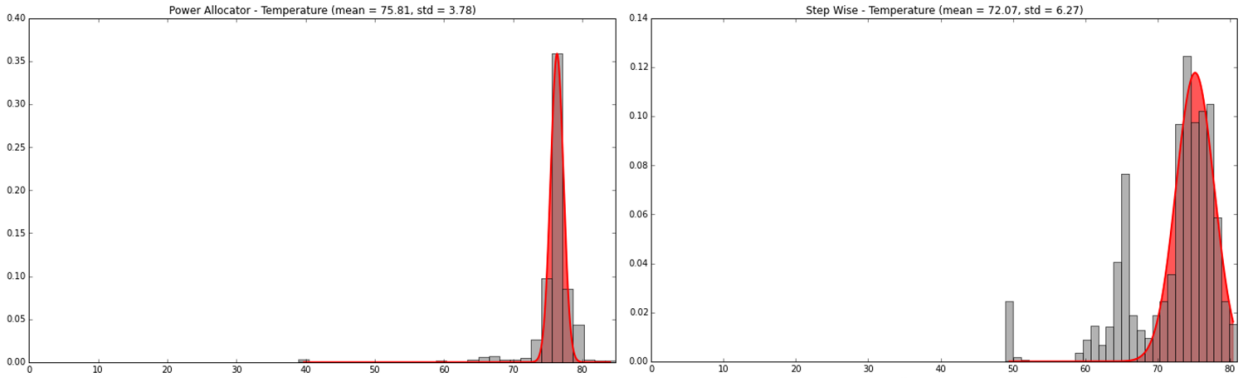

Figure 18 Antutu Temperature deviation comparison

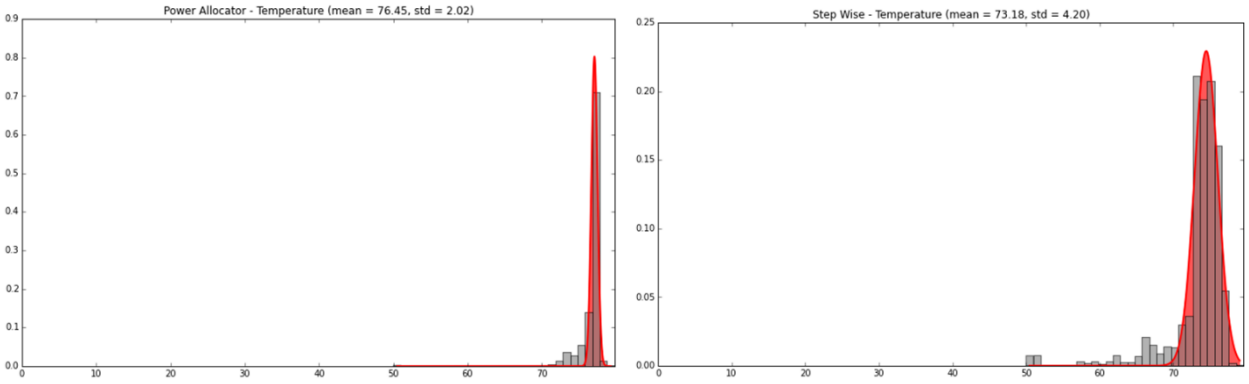

Figure 19 Egypt Offscreen temperature deviation comparison

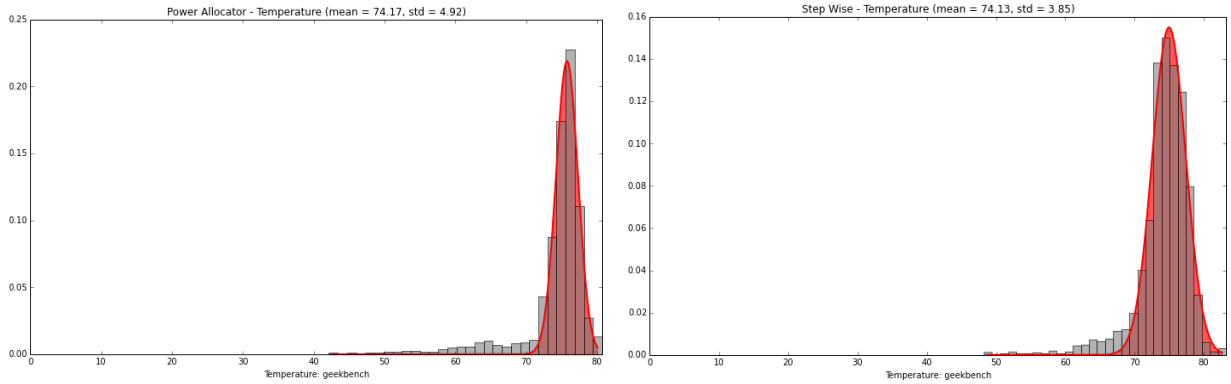

Figure 20 Geekbench temperature deviation comparison

ARM Intelligent Power Allocation shows solid thermal control and better performance than existing Linux thermal framework.

7 References

This document refers to the following publications from third parties:

- [1] Device scaling limits of Si MOSFETs and their application dependencies.

  http://ieeexplore.ieee.org/document/915374/.

- [2] Dynamic Voltage Scaling.

  https://en.wikipedia.org/wiki/Dynamic_voltage_scaling.

- [3] PID controller

  https://en.wikipedia.org/wiki/PID_controller.

- [4] CMOS

  https://en.wikipedia.org/wiki/CMOS.