Solution Brief

Power Management - Dynamic Receive Side to Increase Sleep State Residency

Executive Summary

Intel provides several CPU power management features that help Communication Service Providers (CoSP) to save their energy consumption when the cores of a processor are not in use. One such technology is C-states, which gives the user the opportunity to put those cores to sleep when not needed, for example, before any workloads have been deployed on a system.

In this pre-deployment scenario, broadcast, multicast, or unicast network traffic may be directed to the network interface ports, which may cause interrupts to be sent to all the cores of the processor. This will wake them out of any sleep state they are in. Having these cores in a running state unnecessarily increases the power consumption of the processor.

This guide describes how to mitigate the waking of many cores from sleep states when network traffic arrives and optimizes the power consumption by keeping most of the CPU cores in low-power C-state. This is achieved by directing all the interrupts to a minimum number of cores, which can result in power savings of up to 12%[1].

This document is part of the Network Transformation Experience Kits.

Introduction

As CoSPs drive towards their energy savings goals, it is important to identify every opportunity to save power in their networks. Intel has a suite of power management technologies to support these energy-saving methods.

C-states, which we use as part of this solution, are sleep states that the individual cores of the processor can go into when idle. They range from C0 to C1 and C1E, to C6. C0 is a running state, C1 and C1E are shallow sleep states, and C6 is the deepest sleep state which provides the most energy savings. The number of available C-states differs from generation to generation of CPU. In this guide, we will focus on the Core C6 (deep sleep) state.

Servers running standard Linux networking stacks may be handling low rates of routing and general network traffic which, by default, will distribute interrupts across all cores. This interrupt balancing can inadvertently prevent cores from entering Core C6 because they are handling interrupts instead of sleeping. This guide will focus on methods to tune the Linux operating system to direct network adapter interrupts to the least number of cores, allowing the remaining cores to sleep. Methodologies covered in this guide include dynamic management of Receive-Side Scaling (RSS) and provisioning the most appropriate power management configuration.

It is suggested that the reader work with their ecosystem partners to implement a solution based on the methodologies outlined.

[1] Workloads and configurations. Results may vary.

Solution Description

Prerequisites

Before tuning, we need to configure the system with the prerequisite settings for the test use-case. The only power management technology enabled on the test server is C-States. All other power management technologies are disabled. This is to isolate the benefit of the configuration suggested in this paper. Other technologies such as P-states, uncore frequency scaling, and Turbo Boost, are disabled.

Methods for Tuning Network Adapter Interrupts

Two techniques are used and described in the following sections:

- Static queue configuration: In this configuration, the user manually lowers the number of queues using `$ethtool`. This reduces the number of cores that handle interrupts from incoming network queues, allowing the remaining cores to enter sleep states.
- Dynamic queue configuration: In this configuration, the user sets queues based on the incoming traffic rate, ideally using software that dynamically monitors the rate and enables the appropriate number of queues. This reduces the number of queues when they are not needed and enables additional queues as the traffic grows, keeping the number of unused cores at an optimal level.

Before discussing either approach, we first need to enable C6 for all cores. This can be achieved using the power.py script. Refer to Appendix A.

The first approach proposed is static queue configuration using RSS, where the system is configured using `$ethtool` to set the number of network adapter queues, which are then distributed over the available cores. A lot of systems typically maximize the number of queues used in the network interface and distribute packets evenly across these queues, which results in interrupts being sent to cores that handle the given queue. This results in cores regularly servicing short-lived lightweight tasks, preventing them from entering a sleep state or causing them to be woken from sleep states, reducing C-State residency.

The static queue configuration approach reduces the number of cores that are available to handle interrupts. In this example, we can reduce the number of cores used to 1 by issuing the following command:

```
$ethtool -X $ifname equal 1
```

This has the effect of keeping all the interrupts from that network interface on a single core, allowing all the remaining cores to maximize their C6 residency.

This static queue configuration approach, using a single core, proves the concept. However, this approach requires additional steps in order to scale up when there is an increase in traffic.

This is where dynamic queue configuration should come into play. This is achieved by dynamically adjusting the number of enabled queues on affected interfaces based on the incoming workload. This is achieved using the following command:

```
$ethtool -X $ifname equal x
```

where, `$ifname` should be substituted with the targeted interface name, and x is the number of queues to enable up to a maximum of the number of cores on the local NUMA socket.

Increasing the number of queues has the effect of also increasing the number of cores available to service the interrupts from those queues. Please ensure that cores handling interrupts are on the same NUMA node as the network interface. This can be achieved by observing the contents of /proc/interrupts and matching those interrupts with the cores handling them. As traffic increases, additional queues should be added to the pool of available queues and the IRQ affinity should be modified to process increasing traffic. The opposite is true when the traffic reduces again.

Adaptive interrupt moderation (a feature available on many network adapters, which allows network cards to accumulate several packets before an interrupt is sent to the core) should be turned off. This results in a reduced number of interrupts and allows for increased residency across the cores. This is achieved by the following command:

```
ethtool -C $ifname adaptive-rx off
```

To achieve dynamic queue configuration, the main execution loop of a potential solution would consist of:

- Calculation of the current packet per second (PPS) rate
- Logic control flow designed to increase or decrease the number of receive queues based on the PPS rate

To do this efficiently, a ratio of packets per queue needs to be worked out that suits the system best. Then, based on PPS you can calculate how many receive queues are needed. Based on the ratio of PPS and the number of required queues it is now possible to increase the number of queues using the following command:

```
ethtool -X $ifname equal $new_num_active_qs && cpupower -c $CPUS idleset -D 3
```

Where, `$new_num_active_qs` should be substituted with the new number of queues required and `$CPUS` should target

affinitive CPUs. To decrease the receive queues when queues become redundant, the following command is used:

```
ethtool -X $ifname equal $new_num_active_qs && cpupower -c $CPU idleset -enable-all
```

Where, `$new_num_active_qs` and `$CPU` are set to the number of required queues and affinitive CPUs, respectively.

Benefits of Solution

The main benefit of this solution is power savings associated with being able to achieve a deep sleep state (C6) and not being woken up by unnecessary traffic. Figure 1 shows power consumption on one socket when broadcast traffic is sent to the network adapter ports, as it can be observed on average, approximately 11 Watts of power is saved between the unoptimized baseline, 2 static examples, and dynamic RSS approach. The x-axis is a timeline with a multiplier of five seconds, y-axis represents package power consumption in watts[2].

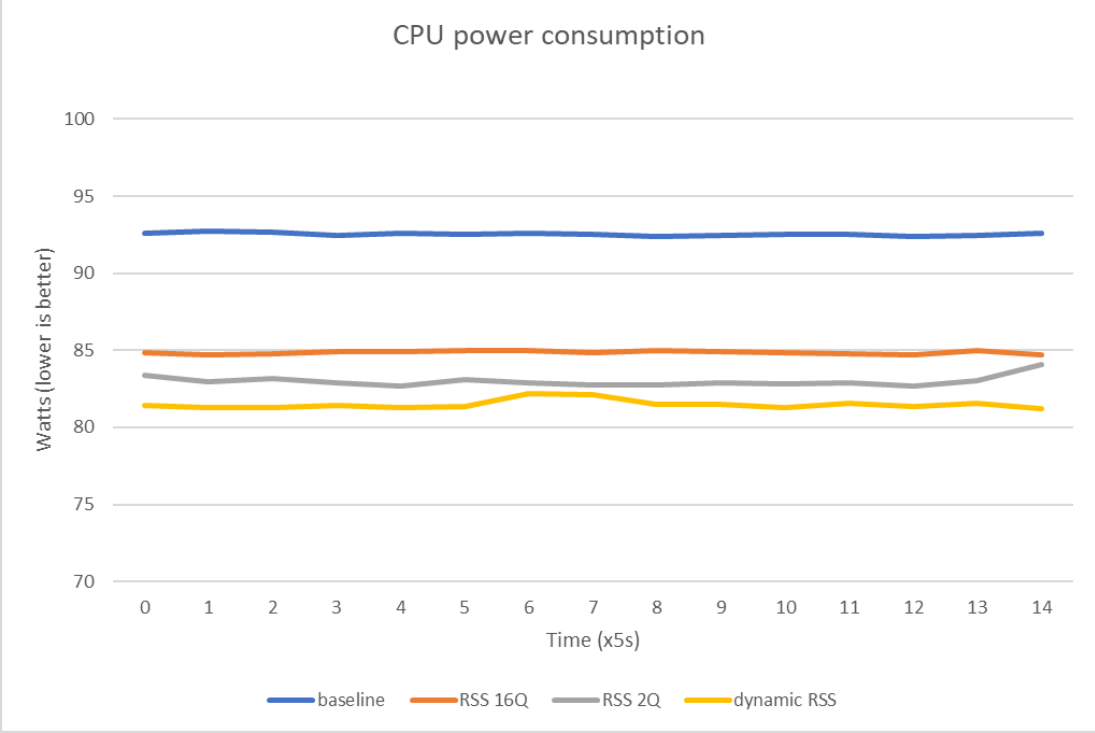

Figure 1. Power consumption while all cores are receiving traffic

Figure 2, shows the C6 residency when all cores are in the default unoptimized state, accepting interrupts from the network interfaces. Each core spends less than 0.02% of its time in a deep sleep state leading to higher power consumption than the optimized state.

[2] Workloads and configurations. Results may vary.

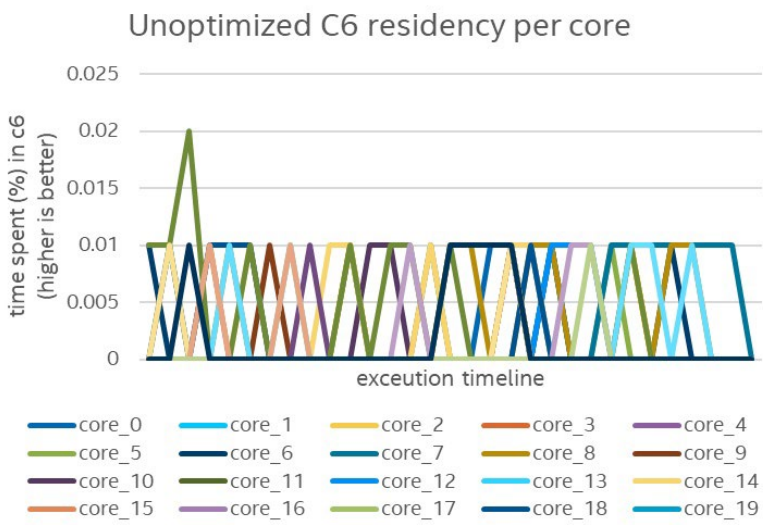

Figure 2. C-States residency per core without optimizations

A significant improvement is shown in Figure 3 where dynamic RSS is enabled on the system. It can be observed that most cores spend their time between 80-100% in deep sleep state (C6).

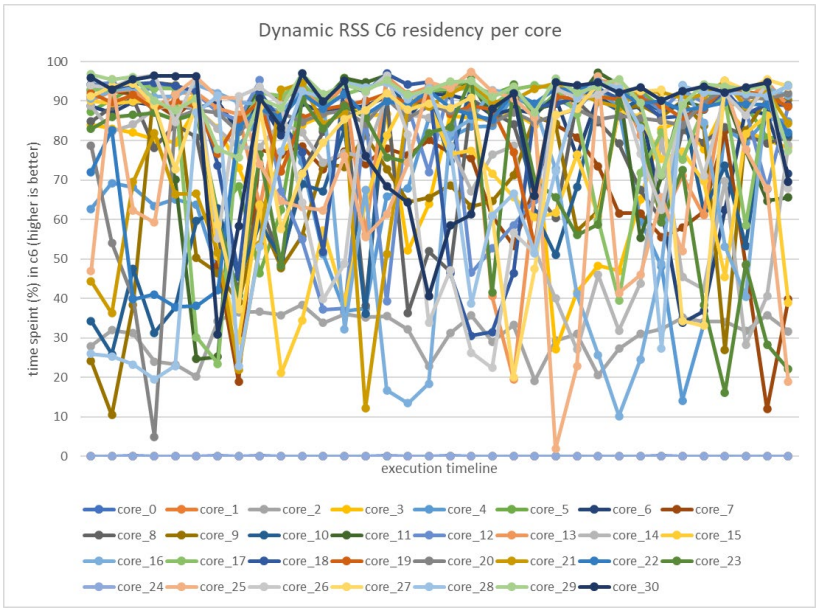

Figure 3. C6 sleep state residency per core when dynamic RSS is enabled

The metrics for C6 residency (as well as many other metrics) can be collected using the `$turbostat` command, which is available in linux-tools-common package. This data was collected on Intel® Xeon® Gold 6438N processor and 100GbE Intel® Ethernet Network Adapter E810.

Summary

This guide describes two techniques: the first is a static configuration to reduce the number of cores handling interrupts to a single core, and the second is a dynamic configuration to adjust the number of cores based on the incoming traffic rates.

A significant improvement in C6 residency was observed by reducing the number of cores handling RSS queue interrupts. The residency baseline was <1%, whereas after implementing the methodologies described above, the residency on some cores was improved to above 80%, resulting in up to 11 Watts of power savings in our test environment[3].

[3] Workloads and configurations. Results may vary.

Appendix A

In this guide, ethtool and turbostat tools were used. Ethtool is a Linux tool for the control and display of network controller device settings and associated device drivers. The tool itself is very extensive and its full capabilities are outside of the scope of this guide. However, if the tool is not present on the system it can be found and installed using the following command:

```
apt-get install ethtool
```

Turbostat is a tool that can be used for reading processor telemetry such as frequency, idle power states, temperature, power and topology. It can be found in linux-tools-generic package and installed using the following command:

```
apt-get install linux-tools-$(uname -r)-generic
```

To alter c-state settings on the system power.py script available here is used:

https://github.com/intel/CommsPowerManagement

This extensive script can be used to display and change many of the Intel CPU features such as available c-states and current settings.

Terminology

Table 1. Terminology

Abbreviation	Description
CoSP	Communication Service Providers
IRQ	Interrupt request
NUMA	Non uniform memory access
PPS	Packets per second
RSS	Receive Side Scaling

References

Table 2. References

Reference	Source
Dynamic Interface Power Management	https://netdevconf.info/0x15/session.html?Dynamic-Interface-Power-Management-(PowerMAN)
Dynamic Interface Power Management (PowerMAN)	https://netdevconf.info/0x15/papers/4/powerman_final.pdf

Document Revision History

Revision	Date	Description
001	February 2024	Initial release.

0224/DN/WIT/PDF 814415 -001US