RISC-V QoS Strategy

Draft by ved@rivosinc.com for discussion at the SoC HC and QoS SIG

Introduction

Quality of Service (QoS) is the minimal end-to-end performance that is guaranteed in advance by a service level agreement (SLA) to an application. The performance may be measured in the form of metrics such as instructions per cycle (IPC), latency of servicing work, etc.

Various factors such as the available cache capacity, memory bandwidth, interconnect bandwidth, CPU cycles, system memory, etc. affect the performance in a computing system that runs multiple applications concurrently. Further when there is arbitration required for shared resources, the prioritization of the applications requests against other competing requests may also affect the performance of the application.

When multiple applications are running concurrently on modern processors with large core counts, multiple cache hierarchies, and multiple memory controllers, the performance of an application becomes less deterministic or even non-deterministic as the performance depends on the behavior of all the other applications in the machine that contend for the shared resources leading to interference. In many deployment scenarios such as public cloud servers the application owner may not be in control of the type and placement of other applications in the platform.

System software can control some of these resources available to the application such as the number of hardware threads made available for execution, the amount of system memory allocated to the applications, the number of CPU cycles provided for execution, etc. System software needs additional tools to control interference to an application and thereby reduce the variability in performance experienced by one application due to other applications' cache capacity usage, memory bandwidth usage, and interconnect bandwidth usage through a resource allocation extension.

Effective use of the resource allocation extension requires hardware to provide a resource monitoring extension by which the resource requirements of an application needed to meet a certain performance goal can be characterized.

A typical use model involves profiling the resource usage of the application using the resource monitoring extensions and to establish resource allocations for the application using the resource allocation extensions.

Literature review

1. Per-Thread Cycle Accounting in Multicore Processors studied the problem of interference among threads caused by contention effects due to competition for shared resources such as caches, memory bandwidth, etc. The paper presented a per-thread cycle-accounting architecture and had following key conclusions
    a. Resource sharing leads to interference among co-executing threads in multicore processors because of contention effects: co-executing threads compete for the shared

resources, such as caches, off-chip bandwidth, memory banks, etc., which leads to unpredictable per-thread performance.

2. Heracles: Improving Resource Efficiency at Scale - the paper presents a heuristic feedback-based system to manage isolation mechanisms to enable latency-critical workloads to be colocated with batch jobs. The paper studied the impact of shared resources on a set of real-world workloads at scale and established the impact of interference caused by sharing LLC, DRAM bandwidth, power, and network bandwidth. The paper had following key conclusions:
    a. Dynamically managing multiple hardware and software isolation mechanisms, such as CPU, memory, and network isolation, to ensure that the latency-sensitive job meets latency targets while maximizing the resources given to best-effort tasks enabled server utilizations of 90% without latency violations. The controller made use of QoS technologies such as Intel Resource Director Technology (RDT).
3. CAT @ scale - a presentation from Google on experience deploying cache isolation using Intel Resource Director Technology (RDT) in a mixed workload environment. Following key takeaways were shared:
    a. With larger machines, isolation between workloads is more important than ever.
    b. RDT extensions work really great at scale:
    c. Claimed Google loves cgroups and containers and rolled out cgroup based CAT support to the fleet.
4. Achieve stable high performance DPDK Application on modern CPU - a presentation from DPDK conference presented impact of shared LLC and memory bandwidth on networking applications and showed use of Intel Resource Director Technology (RDT) being used to recover most of the performance. Following key takeaway were shared
    a. Shared resource partition can reduce the competition and achieve stable high performance
5. Performance evaluation of cache allocation technology for NFV noisy neighbor mitigation
6. An Integrated Instrumentation and Insights Framework for Holistic 5G Slice Assurance
7. Resource Tuning for Energy Efficient Slicing
8. Resource Allocation: Intel Resource Director Technology (RDT) - LinuxCon 2016

Key takeaways from the literature review

1. Multiple studies showed resource sharing leads to unpredictable per-thread performance when co-executing threads compete for the shared resources, such as caches, off-chip bandwidth, memory banks, etc., which leads to.
2. The resources that are most contended for and thus lead to most interference effects are shared caches and memory bandwidth.

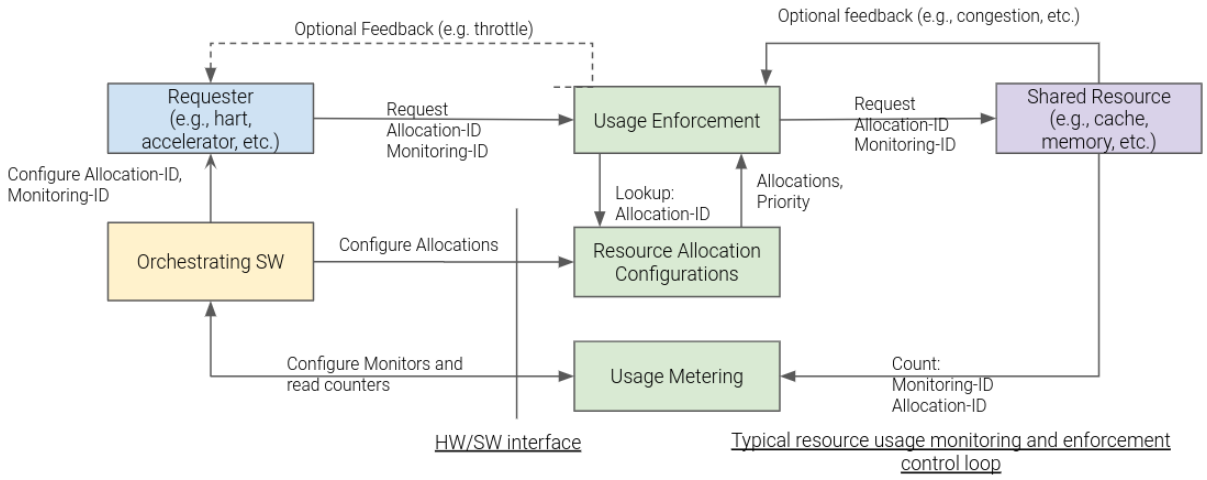
Survey of existing technologies

A gap analysis against existing technologies was done - Quality of Service Gap Analysis. Key conclusions as follows:

1. Associate an ID with an application to lookup capacity allocations and monitoring usage
2. Support partitioning/allocation of L2 cache, L3 cache, and memory bandwidth.
3. Support differentiating code vs. data usage.

Typical control system for QoS enforcement

The QoS technologies enable system software to implement a feedback driven control loop to achieve the SLA objectives of the applications contending for the shared resources. A typical feedback control loop is as follows:



Strategy discussion

There are two key gaps identified:

1. QoS ID (QID): Standard mechanism for associating a resource control ID (RCID) and a resource monitoring ID (RMID) with a thread of execution. These IDs would be part of the thread context and expected to be context switched by the OS/VMM scheduler.
2. QoS Register Interface (QRI)): Standard register interface for configuring resource allocations into resource controllers and to enable resource usage monitoring in such controllers. QRI enables a uniform view of configuring and controlling the resource controllers and eases product integration and software enabling/support.

RVI members developing products have two choices for each component, such as cache controllers, memory controllers, etc. that are required for the product:

1. Implement a custom QoS register interface
2. License other architectures QoS register interface
3. Buy other architecture QoS enabled IP

RVI member companies developing IPs, such as cache controllers, memory controllers, etc. for the RVI ecosystem have two choices:

1. Implement a custom QoS register interface
2. License other architectures QoS register interface

All of these choices are valid and must be supported.

Custom QoS register interfaces place a burden on the member companies to have to develop the architecture and enable/support it uniquely in the software ecosystem. This may further be a hindrance to the development of a rich RVI ecosystem of IP providers if the architecture is fragmented by custom

implementations making product integration difficult and enabling/supporting in the RVI software ecosystem harder. These may be alleviated to some extent by custom vendor specific drivers.

Implementing a custom CSR or other method for QID however places burden on the product integration and upstream software enabling/support. A QID mechanism is needed irrespective of the QRI choices made by product integrators or IP providers. The QID mechanism, needing to be more closely integrated into the OS/VMM schedulers, may be harder to enable as a custom extension.

The following 4 step strategy is proposed to address the identified gaps in RVI technology portfolio:

1. Pursue a fast track privileged QID extension.
    a. This may be a single CSR to configure QID such as RCID and a MCID into a RISC-V hart.
2. Pursue a IOMMU QID extension to associate a QID with a device context.
3. Form a TG to define a QRI specification with following charter:
    a. The QRI TG shall develop a specification to standardize a memory-mapped register interface for **cache controllers** and **memory controllers** to:
        i. Configure capacity allocations and resource arbitration priorities
        ii. Monitor capacity usage
    b. The QRI TG shall not prescribe:
        i. The level of QoS in the system
        ii. Methods for usage metering or enforcing the configured allocations
        iii. Behavior of components in the system that choose not to implement QRI
4. QoS SIG to explore the need for additional QoS technologies such as power, hyperthread, etc. and define the strategy for addressing identified gaps.