CERIAS

The Center for Education and Research in Information Assurance and Security

WIP: Secure High-Performance Interrupts For Secure High-Performance Processors

Berk Aydogmus
baydogmu@purdue.edu

Kazem Taram
kazem@purdue.edu

Abstract

The advent of User level Interrupts sparked an advance in the workloads that benefit from low latency notification systems. User level schedulers, and high throughput devices are the two main customers of such systems. The lack of OS management in interruption, can however cause security exploits. The approach we propose for User Interrupt handling eliminates the latency side-channel identified in prior works, and reduces the overhead of User Interrupts by a significant margin.

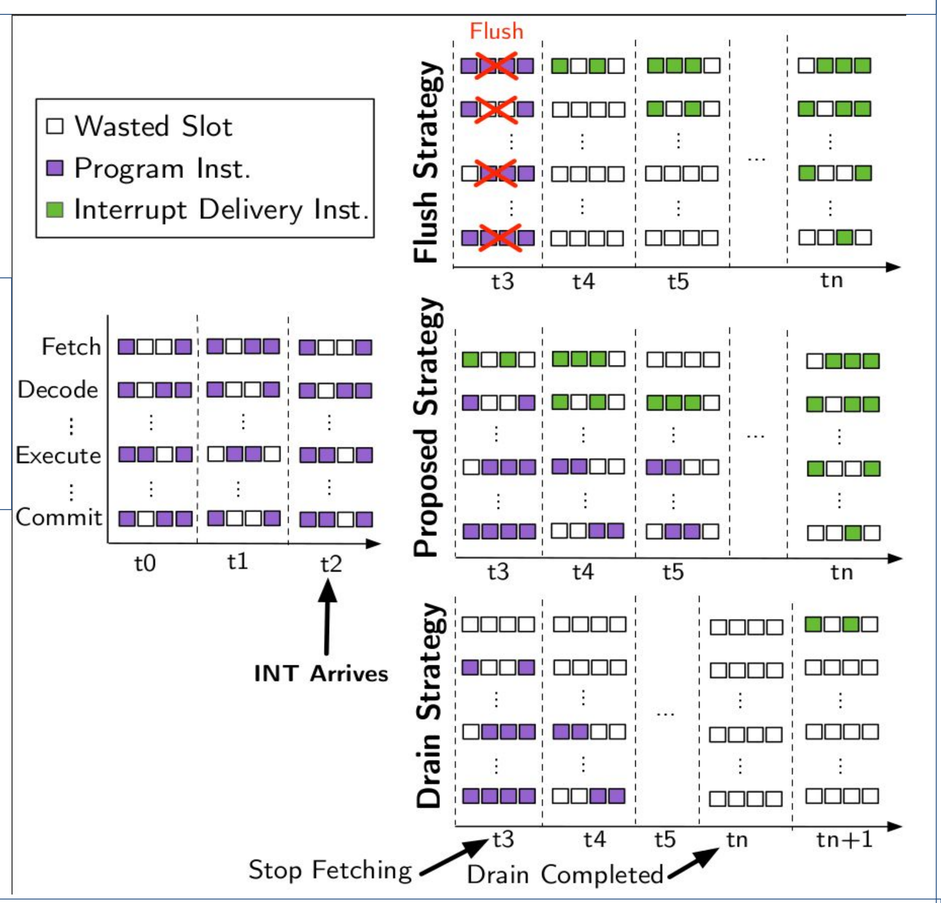
Moreover, we investigate the potential integration of hardware timers to further enhance user-level preemptive schedulers. While current user level schedulers dedicate a core to generate interrupts on dedicated time intervals, a hardware timer generates interrupts in core, which reduces the constant overhead of interrupt communication bookkeeping, and the cycles spent spinning for the next interrupt interval.

Security Concerns

Prior work has shown that interrupt handling systems are prone to latency side-channels, arising from the need to wait for the instruction that is in flight to start processing. As an example in the best case scenario of having to wait for a nop instruction we are able to send %28 of interrupts generated from a core, while for the scenario of having to wait for a longer latency interrupt we can reduce that rate to 8% This, we hypothesize gives an attacker with knowledge of interrupt intervals or the ability to send an interrupt, the chance to fingerprint the instructions and determine the path a program takes. Furthermore because interrupts redirect the flow of execution on demand, a speculative exploit might be possible.

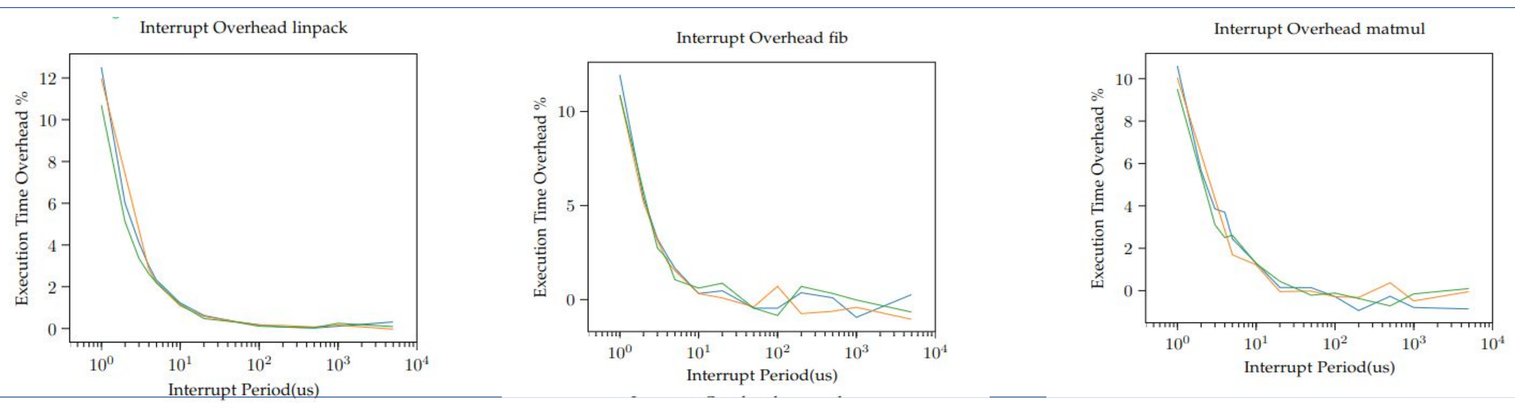
Methodology

There are two established ways of handling interrupts in modern processors: *Flush strategy* will empty the entire pipeline as soon as the interrupt arrives, while *Drain strategy* will block the fetch stage and wait until all the instructions already in the pipeline are finished. In both cases, the interrupt handling starts from an empty pipeline. This is a complete waste of pipeline capacity. We propose an architecture in which the pipeline has the ability to seamlessly continue the execution even when interrupts are arriving.



Performance Results

Our overheads for the proposed method and other methods modeled in gem5 simulator yield these results for a CPU model close to Ice Lake server CPUs.



On a more recent processor core (Sapphire Rapids), we get more significant decreases in overhead. In the extreme case, we get an overhead reduction of 4.4% compared to Flush strategy. Further tests indicate that most of the overhead is caused by the interrupt delivery bookkeeping code, implying in cases where there is no need for intercore communication, the **overhead can be brought close to zero**. In fact, when we test this by performing a minimal interrupt delivery (no handler), we achieve an overhead of %0.02. This sets the groundwork for proposing specialized notification mechanism that have near zero overhead, **potentially revolutionizing a wide range of use cases from high-performance networking (as alternative to polling) to more efficient preemption and scheduling to more efficient synchronization with on-chip accelerators.**

PURDUE UNIVERSITY

CERIAS