Bus lock detection and handling

Authors: Fenghua Yu <fenghua.yu@intel.com>
Tony Luck <tony.luck@intel.com>

Problem

A split lock is any atomic operation whose operand crosses two cache lines. Since the operand spans two cache lines and the operation must be atomic, the system locks the bus while the CPU accesses the two cache lines.

A bus lock is acquired through either split locked access to writeback (WB) memory or any locked access to non-WB memory. This is typically thousands of cycles slower than an atomic operation within a cache line. It also disrupts performance on other cores and brings the whole system to its knees.

Detection

Intel processors may support either or both of the following hardware mechanisms to detect split locks and bus locks.

#AC exception for split lock detection

Beginning with the Tremont Atom CPU split lock operations may raise an Alignment Check (#AC) exception when a split lock operation is attemped.

#DB exception for bus lock detection

Some CPUs have the ability to notify the kernel by an #DB trap after a user instruction acquires a bus lock and is executed. This allows the kernel to terminate the application or to enforce throttling.

Software handling

The kernel #AC and #DB handlers handle bus lock based on the kernel parameter "split_lock_detect". Here is a summary of different options:

split_lock_detect=	#AC for split lock	#DB for bus lock
off	Do nothing	Do nothing
warn (default)	Kernel OOPs Warn once per task and disable future checking When both features are supported, warn in #AC	Warn once per task and and continues to run.
fatal	Kernel OOPs Send SIGBUS to user When both features are supported, fatal in #AC	Send SIGBUS to user.
ratelimit:N (0 < N <= 1000)	Do nothing	Limit bus lock rate to N bus locks per second system wide and warn on bus locks.

Usages

Detecting and handling bus lock may find usages in various areas:

It is critical for real time system designers who build consolidated real time systems. These systems run hard real time code on some cores and run "untrusted" user processes on other cores. The hard real time cannot afford to have any bus lock from the untrusted processes to hurt real time performance. To date the designers have been unable to deploy these solutions as they have no way to prevent the "untrusted" user code from generating split lock and bus lock to block the hard real time code to access memory during bus locking.

It's also useful for general computing to prevent guests or user applications from slowing down the overall system by executing instructions with bus lock.

Guidance

off

Disable checking for split lock and bus lock. This option can be useful if there are legacy applications that trigger these events at a low rate so that mitigation is not needed.

warn

A warning is emitted when a bus lock is detected which allows to identify the offending application. This is the default behavior.

fatal

In this case, the bus lock is not tolerated and the process is killed.

ratelimit

A system wide bus lock rate limit N is specified where 0 < N <= 1000. This allows a bus lock rate up to N bus locks per second. When the bus lock rate is exceeded then any task which is caught via the buslock #DB exception is throttled by enforced sleeps until the rate goes under the limit again.

This is an effective mitigation in cases where a minimal impact can be tolerated, but an eventual Denial of Service attack has to be prevented. It allows to identify the offending processes and analyze whether they are malicious or just badly written.

Selecting a rate limit of 1000 allows the bus to be locked for up to about seven million cycles each second (assuming 7000 cycles for each bus lock). On a 2 GHz processor that would be about 0.35% system slowdown.