High-vCPU Count Guest Enablement

David Woodhouse <dwmw@amazon.co.uk> 2020-10-29

Background

Instance types with >255 vCPUs present a new challenge. Intel's XAPIC design only supports 8 bits of destination APIC ID for targeting interrupts. The X2APIC design expands this to 32 bits for Inter-Processor Interrupts (IPIs) but not intrinsically for external interrupts.

External interrupts are received as Message Signaled Interrupts (MSI), where the device requesting attention performs a memory write to special physical address range owned by the APIC. Even with X2APIC, the Compatibility Format MSI messages only have space for 8 bits of destination ID.

Intel's design is that systems with any APIC ID above 255 shall use Interrupt Remapping, a feature of the IOMMU. With Interrupt Remapping, the Compatibility Format MSI message which directly encodes the destination CPU is no longer used. Instead, the Remappable Format is used which is merely an index into the IOMMU's remapping table — which *can* contain the full 32 bits of destination ID.

Interrupt Remapping was added as an optional supported feature to Intel's VT-d IOMMU specification. Unfortunately, the initial VT-d specification did not make DMA translation an "optional" feature. Thus, there is no architectural way to expose an IOMMU which supports only Interrupt Remapping and not DMA translation.

DMA translation is problematic to expose to guests. It is extremely security-sensitive, as an error in configuring DMA translations could expose memory belonging to the host or other guests. It also requires space to shadow the guest's translation tables, turning the guest's IOVA→GPA translations into IOVA→HPA tables for the IOMMU to use directly. In the pathological case, that could require up to 8GiB of page tables in the host, unless we implement a non-architectural solution such as silently killing the guest if it exceeds a heuristic limit. We do not want to have to implement a full IOMMU. Certainly not before VT-d Scalable Mode potentially makes it possible to use hardware nesting of translations.

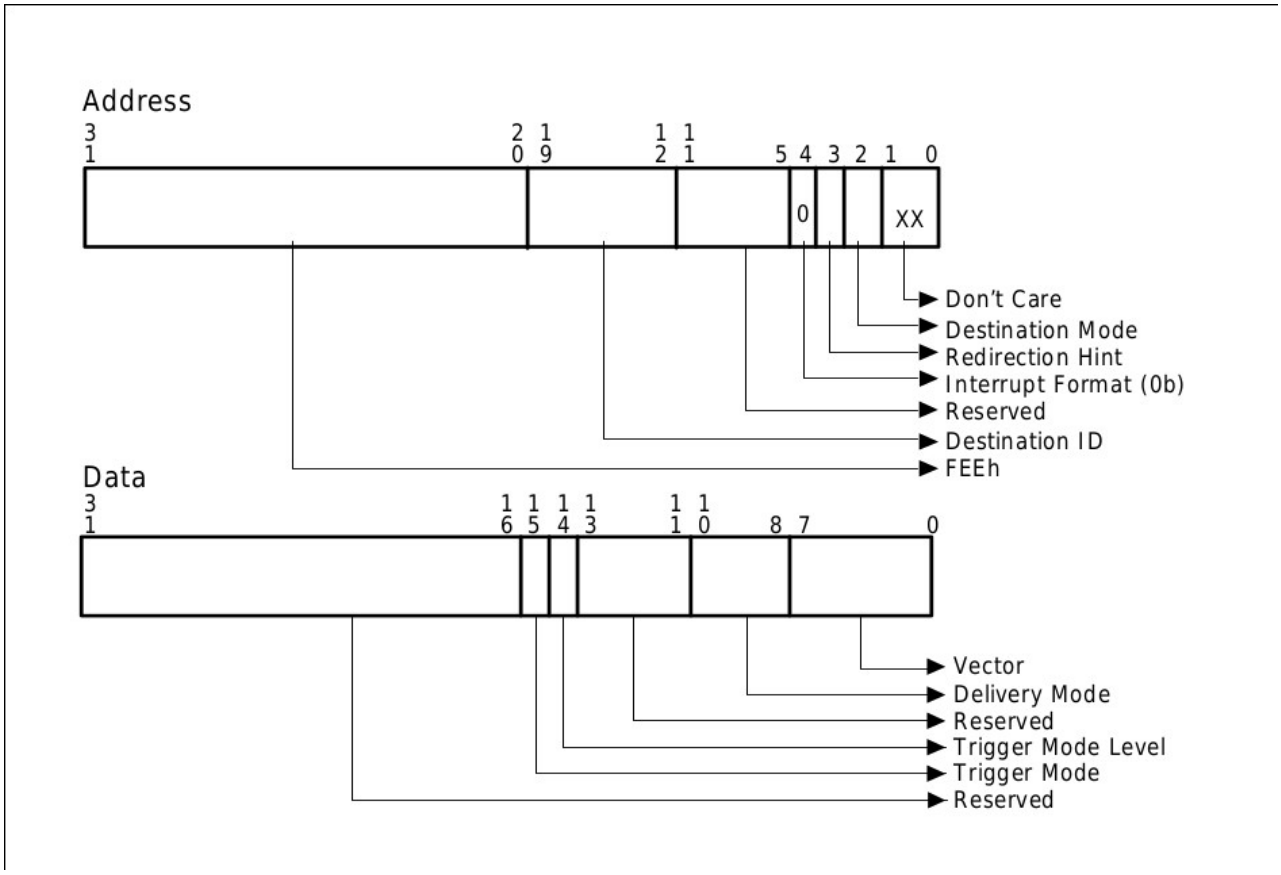
Option 1: Native 15-bit MSI support

The simplest option is to forget about Interrupt Remapping completely. There *are* extra bits available in the Compatibility Format MSI messages, and they are even supported by existing Intel hardware.

The I/OAPIC is a device for turning legacy line-based interrupts to MSIs. It contains a Redirection Table Entry (RTE) for each pin, and the RTE on real hardware already has an 8-bit 'Extended Destination ID' field, which translates to the previously "Reserved" bits 11-4 of the resulting MSI message.

Intel uses the lowest of the Extended Destination ID bits to signal a Remappable Format interrupt, but the remaining 7 bits are not architecturally used unless the Remappable Format bit is set. A hypervisor can advertise an enlightenment which uses those 7 bits as an additional 7 bits of APIC ID, offering support for up to 32768 vCPUs without the need for Interrupt Remapping.

It would be theoretically possible to use even more bits of the MSI message, giving a full 32 bits of APIC ID, but that would not be addressable through the I/OAPIC without an extension to the RTE format which would be even more complicated. The beauty of the 15-bit approach is that it can be implemented without even having to touch the I/OAPIC code.



The patches for Linux to support 15-bit mode are submitted and will be pulled in to the tip.git tree as soon as Linus has pushed out the 5.10-rc1 kernel. At that point the 15-bit support will be in linux-next ready to be merged into 5.11.

To encourage guest support for this extension, it is best for it not to be seen as vendor-specific. A patch for Qemu to support the feature has been submitted upstream (where it will need to wait for the new KVM feature bit to become officially defined as the Linux patches are merged). Microsoft were planning to implement a similar extension in Hyper-V, but have now changed their plans to be precisely compatible with the format defined here. We should look at doing it in Xen also.

NOTE: Without Interrupt Remapping, Linux uses the X2APIC in 'flat' mode, directing IPIs only at a single processor at a time. In 'cluster' mode it could multicast an IPI to 16 CPUs at a time using the full 32 bits. It is possible to use cluster mode for IPIs and still use flat mode for external IRQs, since external IRQs are unicast anyway. It would be worth benchmarking the `send_IPI_mask()` functionality to see if that's worthwhile.

Option 2: IOMMU without DMA remapping

Although there is no architectural support for an IOMMU to report that it lacks the capability to perform DMA remapping, there are a set of bits which indicate the supported levels of page tables — 3-level page tables, 4-level page tables, and/or 5-level page tables. An IOMMU which sets none of those bits cannot be used for DMA translation.

A patch to support `dma-translation=off` in the emulated Intel IOMMU in Qemu has been submitted.

It has been verified that Windows 10 *(ISO downloaded from Microsoft and tested with the modified Qemu)* boots and enables Interrupt Remapping successfully, while not attempting to enable DMA translation. Booted without the `dma-translation=off` option, the same image does enable DMA translation.

Historically, Linux would refuse to initialise such an IOMMU, and not even use it for Interrupt Remapping. A patch to rectify that has been merged into Linux for the 5.10 release and we intend to backport it to stable branches too.

Option 3: Full IOMMU support including DMA translation

This was our final fallback plan, involving a lot of additional work and security exposure. It was a worst case in case Windows could not support Interrupt Remapping without DMA translation, but thankfully that seems not to be the case as observed in testing of Option 2 above. Therefore, further consideration of this solution can continue only in my nightmares.