January 10, 2019

Direct I/O solution for Containerized HPUX

Maheshwara Aithal
Hewlett Packard Enterprise

Dinesh Thanumoorthy
Hewlett Packard Enterprise

Harish K
Hewlett Packard Enterprise

Direct I/O solution for Containerized HPUX

Abstract

This disclosure relates to the field of hardware emulation solution called c-UX (code named Kiran) which runs HPUX in emulated (Itanium hardware emulation on x86) mode as a futuristic solution for the margin rich UNIX business. The value of containerized HPUX is that it allows customers using legacy HPUX applications to continue running on x86 hardware. c-UX design relies on instruction level emulation which has inherent performance issues. Especially, compute intensive workloads are prone to performance issues while running in emulated environment. However, I/O workloads on such emulated systems can make use of direct device access or device assignment when configured for the highest possible I/O performance. This technique provides the most efficient way to do I/O, compared to other approaches such as device emulation, which imposes a high number of exits from guest context, with the benefits of significantly reduced latency, higher bandwidth, and direct use of bare-metal device drivers. The proposal presents an innovative approach to realize Direct I/O mechanism (a.k.a PCI passthrough) on emulated HPUX environment by leveraging Virtual Function I/O framework in Linux. Disclosed is an approach of accelerating I/O performance in c-UX application by allowing the emulated HPUX Operating System direct access to parts of the I/O subsystem of the host and handle various aspects of the communication like DMA and interrupts. It also throws light on the network I/O performance improvement that is achieved on c-UX, using this method.

Problem statement

The c-UX or Containerized-UX is a solution that can emulate Itanium System on x86-64. It is a user space application that emulates all hardware components of Itanium required to run HPUX OS. Currently, we're experiencing slower rates of I/O on c-UX guest and there is a need to boost the I/O performance by driving maximum throughput on I/O interface to achieve optimum results and get the full value of emulation. Also, we're in need of a common, light-weight, secure, user space I/O framework to avoid using specialized kernel drivers that have to go through the full development cycle and be maintained out of tree. The PCI passthrough technique would solve all these problems but the challenge is to implement a comprehending solution for emulated HPUX on Linux environment.

Competitive approaches

Other techniques include para virtualization or VirtIO model, device emulation model etc. These models need specialized kernel drivers that have to either go through the full development cycle, be maintained out of tree, or make use of the UIO framework, which has no notion of IOMMU protection, limited interrupt support, and requires root privileges to access things like PCI configuration space. The VFIO kernel infrastructure intends to provide a more feature full user space driver environment within a highly secure and programmable IOMMU context, promising near line-speed.

Proposed Solution

Direct I/O mechanism significantly improves the performance by eliminating the device emulation overhead. This method allows the guest HPUX operating system on c-UX to directly access the I/O devices present in the system. From a device and host perspective, this simply turns the VM into a user space driver. Since exits from guest context to the hypervisor are not needed in order to emulate aspects of the device, some applications running on the servers, particularly in the mission critical and high performance computing field, benefit from relatively low-overhead access from user space. This solution takes advantage of the VFIO framework in Linux to emulate certain parts of the device and to manage the other aspects of device handling like MMIO access, DMA and interrupts by bypassing the host kernel.

VFIO (Virtual Function I/O) is an open source device agnostic framework, for exposing direct device access to user space, in a secure, IOMMU protected environment. An IOMMU is a hardware block, comparable to the MMU of the CPU, which intercepts any memory access from the device and translates the memory addresses, from virtual addresses to physical addresses. Initially, when VFIO is loaded on the host Linux system, a bus specific module (the VFIO bus driver) binds to the device, usually discoverable by some kind of bus specific mechanism, such as the PCI configuration space. Then, using the standard VFIO APIs in c-UX, device’s MMIO and DMA regions are mapped to guest memory securely, allowing the HPUX guest to claim the interface, while the x86_64 host can handle and forward incoming interrupts. VFIO also allows to pool the devices that needs to be assigned to the same HPUX guest, thereby enabling the devices to share the same IOMMU TLB.

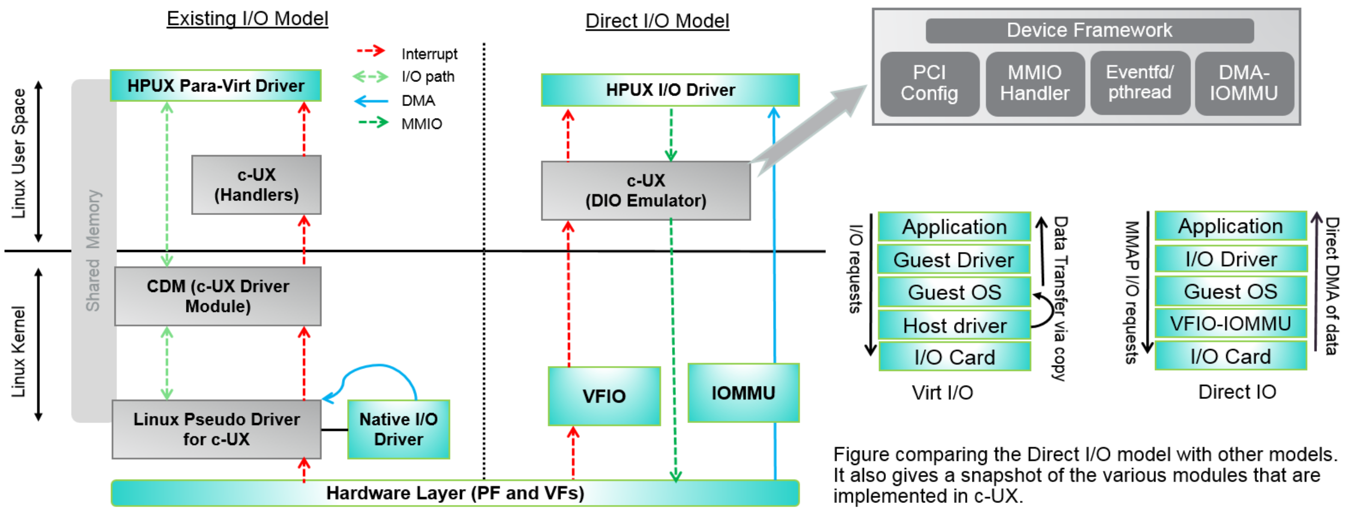

Fig1 An overview of DIO emulator framework components with a snapshot of datapath

<u>Handling Configuration/MMIO space access:</u> In c-UX, a DIO emulator framework is developed that is furnished with device specific information like host hardware path etc. which is eventually parsed and used to populate the device on guest. VFIO abstracts PCI devices as different regions like PCI configuration space, MMIO and I/O port BAR spaces, MSI-X or IRQ spaces etc. We've used the VFIO APIs to query such information and emulate the device so that it can be discovered on guest PCI environment. During the guest boot up, c-UX intercepts and emulates attempts by the guest OS to access PCI config space and MMIO space of the device. Emulation of the device registers is achieved by hooking the interface specific I/O handlers directly to the existing facilities of c-UX. The host BAR address space of the device will be mapped to the guest address space using the standard HPUX WSIO functionality. This way, using the VFIO standard interface, the PCI devices behind an IOMMU is unmapped from the host operating system, and subsequently map to c-UX virtual address space. The unmodified HPUX network driver will claim the exposed device on guest, followed by the initialization, configuration and port link up.

<u>Handling DMA:</u> In this case, the IOMMU is programmed with the Linux host user virtual address of the entire c-UX application, to map the HPUX guest memory with the host physical memory space, enabling the device to do DMA directly to the guest space, eliminating the buffer copy overhead. By configuring the IOMMU this way, the network device will operate under the illusion that they are directly accessing the physical memory of the Linux host, while they have actually been mapped to the memory space of HPUX running on c-UX. The IOMMU will prevent the hardware components behind IOMMU from accessing memory beyond the mappings that have been assigned in the page tables thereby protecting the c-UX system from accesses to memory from the device. Multiple devices can be shared to the same guest by pooling them together by creating VFIO containers, which handles the DMA mapping/unmapping and not the device.

<u>Handling Interrupts:</u> MSI-X is the most efficient way to spread interrupts from one device among multiple cores. VFIO provides standard interfaces to configure and interact with the MSI-X/IRQ signals of the device. The device signals the host driver using interrupts. And in this design, the guest is interrupted using eventfd, a file descriptor for event notification. Eventfd creates an eventfd object that can be used as an event wait/notify mechanism by the emulator running in the user space and by the kernel to notify guest operating system of events. During the MSI-X initialization on guest, the HPUX vector details are gathered by intercepting the MMIO writes using the device MMIO handlers on c-UX. For every vector on guest, a corresponding MSI-X vector is allotted on the device via VFIO, and each vector is assigned a handler (pthread in this case) on the host, along with an eventfd for notification. When an IRQ is being triggered, the eventfd wakes up the handler, which will notify the HPUX guest with the corresponding IRQ details that are already stored in the handler.

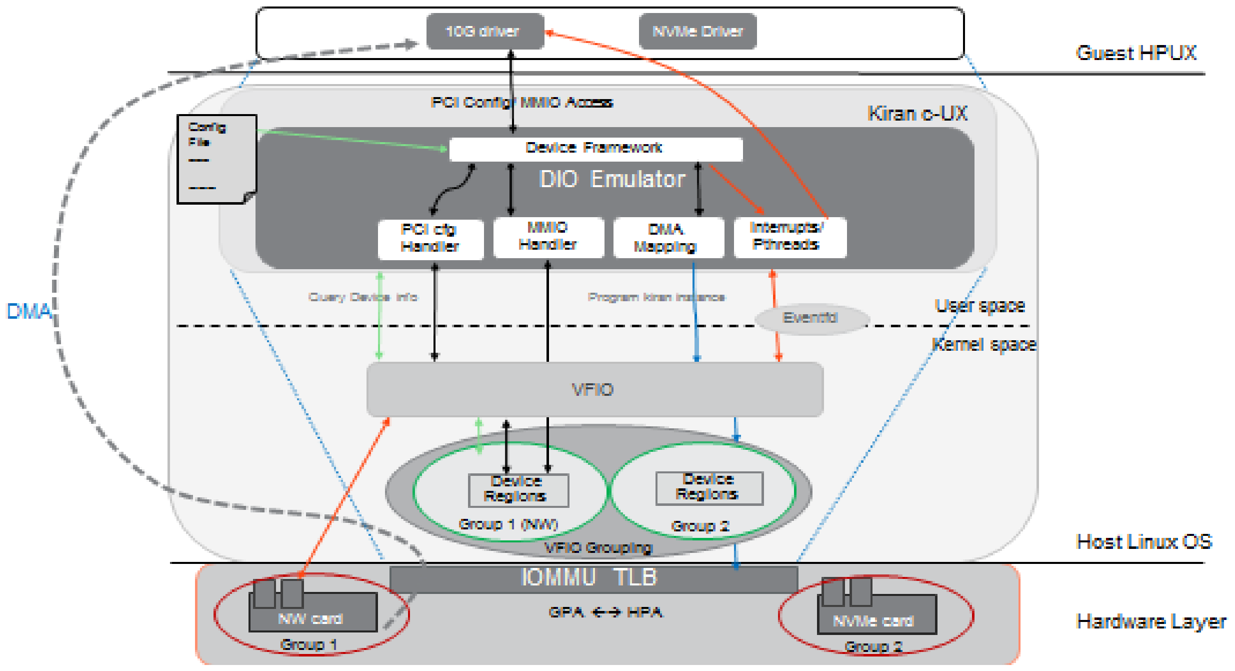

Fig.2 Describing interaction between various modules via DIO emulator on a Linux system

Performance: The DIO model delivers the best performance when implemented within multi-core processor environments by efficient distribution of I/O workloads across CPU cores. Load balancing of interrupts using MSI-X enables more efficient response times and application performance. By avoiding the Rx side of the copy with DMA, this method results in better CPU utilization on virtualized servers. I/O emulation overhead applies for PCI configuration access only which is viewed as control path access and not in the performance path, thereby exhibiting near native performance.