July 30, 2018

A para-virtualization technique to improve the performance of cross-platform emulation

Sanish Suresh
Hewlett Packard Enterprise

Vrashi Ponnappa Puchimanda Ramacha
Hewlett Packard Enterprise

Santosh Abraham
Hewlett Packard Enterprise

Sherin Thyil George
Hewlett Packard Enterprise

A para-virtualization technique to improve the performance of cross-platform emulation

Abstract

A cross platform system emulator is a hyper-visor which can emulate a source ISA (Instruction Set Architecture) platform on a different target ISA platform. Since the source instruction cannot be executed directly on the target system, each CPU instruction of the source platform should be converted to one or more target instructions. This incurs additional overhead, thus reducing the overall performance. In this paper we propose a solution to improve the performance of cross platform emulators that use a binary translator to translate a guest ISA to a target ISA. The solution para-virtualizes hot guest functions by invoking a more efficient host native code instead of translated guest code.

Overview

This disclosure relates to the field of cross-platform emulation, wherein a hypervisor emulates a hardware platform which has a different Instruction Set Architecture (ISA) specification than the platform that hosts the hypervisor. Such an emulation (Figure 1) is performed by translating each of the source ISA instructions (instructions of the program running within the hypervisor) to a set of target ISA instructions (instructions defined in the host's ISA). The translated code blocks are cached for reuse when the same source code block is encountered again. This entire process involves major overhead during run-time and execution of un-optimized CPU instructions, thus penalizing the performance of the program (guest) running inside the hypervisor. As an example, consider the implementation of the emulating Intel Itanium architecture on an x86 architecture. An efficient way to implement a memory copy on Intel Itanium is by using software pipelining, whereas on an x86 architecture, the memory copy can be efficiently implemented using the hardware-provided repetitive 'mov' instruction or using XMM registers. However, the translator component of the emulator cannot determine that the software pipeline loop corresponds to a memory copy operation, thus resulting in a highly inefficient version of memory copy to be emulated.

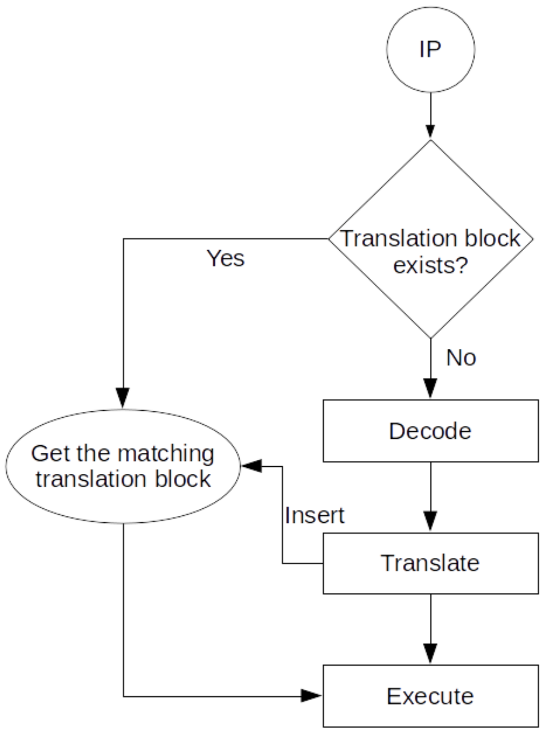

Figure 1

In this paper, a method to bypass the above mentioned translation of the ISA in a cross-platform emulator is disclosed. This method avoids the overhead mentioned above and also generates a target-aware optimized code. The result is a significant boost to the performance of the guest running inside the hypervisor.

Description

In this disclosure, a framework is implemented that allows the guest to notify the emulator about a set of hot functions that needs to be para-virtualized and the address where the set of hot functions is loaded in the guest address space. The hot functions are those functions/code blocks which are frequently executed in a performance path. The emulator will map the guest hot functions to host functions that can exhibit the same functionality in a highly optimized way. Then the emulator’s instruction translator creates a translation block for the first instruction residing at the host function address. But unlike earlier where this translation block contained a set of host ISA instructions, this translation block instead enables a transfer of execution to the mapped host function (Figure 2). The translation block is generated and kept in the cache for all such hot functions. Thus, once the CPU Instruction Pointer (IP) points to one of the hot functions, instead of executing independently translated host ISA instructions, the CPU executes a highly optimized code compiled for the host ISA which provides the same functionality as that of the guest function that is being translated. The host functions are written in such a way that they operate on the guest address space and also have the required checks to emulate faults, traps etc. on the guest. Ideal candidates to be para-virtualized in this way are Kernel APIs and library APIs that are frequently invoked.

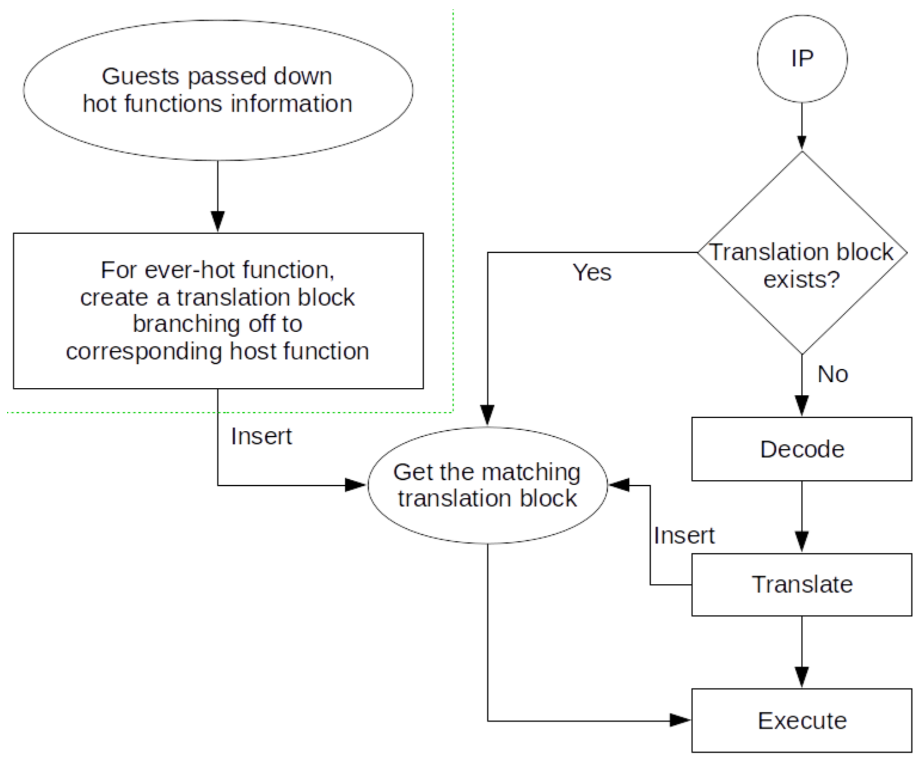
Guests passed down hot functions information
For ever-hot function, create a translation block branching off to corresponding host function
Insert
IP
Translation block exists?
Yes
No
Get the matching translation block
Decode
Insert
Translate
Execute

Figure 2

To enable this functionality, the emulator should know the guest address where the candidate functions are loaded. A new stand-alone guest dynamically loadable kernel module is introduced which passes all the required information of the functions which are to be para-virtualized down to the emulator. For the purpose of this communication, a new firmware call is implemented. The emulator invokes the para-virtualized host functions whenever the IP points to these candidate function addresses. To avoid the overhead of comparing the IP against candidate functions addresses each time to decide whether to generate a translated block or to invoke corresponding para-virtualized functions, our solution creates special translation blocks as soon as the function details are known to the emulator. With this design, whenever the IP matches one of the candidate function addresses, the corresponding target host function gets invoked in a seamless manner without incurring any additional overhead. If a debugger in a guest requests debugging of the hot function by setting breakpoint, a watch point, or a single-step into the hot function, then para-virtualization will be disabled for that function and enabled later at a safe point.

Disclosed by Sanish Suresh, Vrashi Ponnappa Puchimanda Ramacha, Santosh Abraham, Sherin Thyil George - Hewlett Packard Enterprise