Transparent Dynamic Management of Reconfigurable Accelerators in Virtualization Technology

Tian Xia, Jean-Christophe Prévotet, Fabienne Nouvel

Transparent Dynamic Management of Reconfigurable Accelerators in Virtualization Technology

Tian Xia, Jean-Christophe Prevotet and Fabienne Nouvel
INSA, IETR, UMR 6164, F-35708 RENNES
Email: {tian.xia, jean-christophe.prevotet, fabienne.nouvel}@insa-rennes.fr

Abstract

This paper is intended to provide an abstract and transparent layer for virtual machines (VM) to access reconfigurable resources. The underlying infrastructure of partial reconfiguration management is hidden from the VMs. DPR accelerators are presented as virtual devices, which are universally mapped in each VM space as ordinary peripherals. The framework automatically allocates DPR resources dynamically according to a preemptive allocation mechanism. The evaluation of DPR managemen-t overheads demonstrates that our mechanism is implemented with low latency.

1. Introduction

Today, the concept of CPU-FPGA hybrid processor has become more and more popula. In this approach CPU and FPGA domains are tightly connected by dedicated interconnections, which makes it possible to enhance the traditional CPU virtualization with the dynamic partial reconfiguration (DPR) technology on FPGA. However, the exploitation of DPR-enhanced virtualization also brings up new challenges. In virtualization, guest OSs are executing in strongly-isolated environments [1]. When DPR accelerators are shared by multiple virtual machines, the allocation of DPR resources and consistency of hardware tasks are critical problems to be solved.

There are some researches exploring DPR technologies with CPU computing, mostly in context of multi-process OSs [2] or computing servers [3]. In this paper, we propose a dedicated management framework to provide efficient DPR resource sharing in virtual machine system.

2 Virtualization with DPR Management

Ker-ONE is a lightweight kernel that provides para-virtualization on small-scaled embedded ARM systems. Ker-ONE co-hosts multiple guest OSs in isolated virtual machines (VM) in the user level, which are managed by a virtual machine monitor (VMM). Ker-ONE provides only basic functions such as scheduling, inter-process communication (IPC) and memory management, and ends up

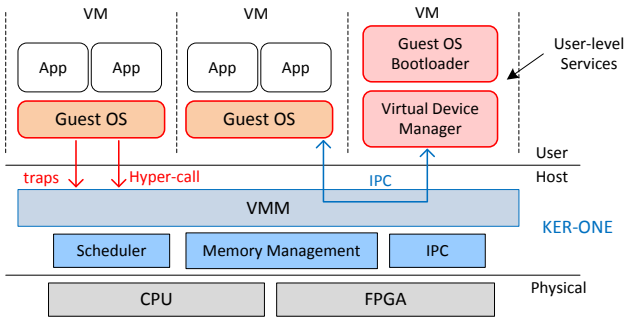

Figure 1. The architecture of KER-ONE virtual machine system

with with a small trust computing base (TCB).The architecture of the KER-ONE virtual machine system is depicted in Figure 1. The features of Ker-ONE include:

- Isolated memory space for each VM. The allocation of physical resources can be performed by manipulating VM page tables to allow or forbid VM's access to specific memory spaces.
- Preemptive priority-based round-robin scheduler. In our case, the dedicated DPR resource manager service is given a higher priority than guest OSs, and will preempt other running components as soon as it's scheduled.
- IRQ-based simple *IPC Channel*, which allows VMs to send messages to other components.

In our system, DPR accelerators are implemented in pre-determined partial reconfiguration regions (PRR) in the FPGA fabric. We proposed a standard interface to all DPR accelerators, so that one PRR can be in multiplexed usage for different accelerators. Between the software and the DPR accelerators, we have introduced an intermediate layer composed by PR interfaces (IF). These IFs are mapped into the address space of VMs as different virtual devices. Each IF is exclusively associated to a specific virtual device in a specific VM. For a certain virtual device, it is mapped at the same address in all VMs, but is implemented with different IFs in lower layer. These IFs are in charge of connecting VMs with the DPR accelerators that actually performs the desired algorithm, so that VM is able to use the virtual devices as normal peripherals.

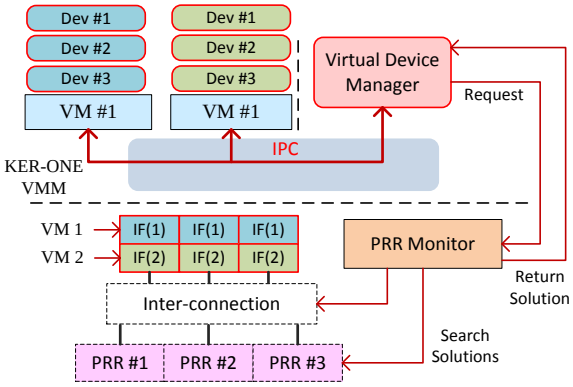

Figure 2. Overview of the DPR management framework in KER-ONE.

An IF has two states, *connected* to a certain PRR or *unconnected*. When an IF is *connected*, it is considered that the corresponding virtual device is implemented in the PRR and that it is ready to be used. Being in the *unconnected* state means that the target accelerator is unavailable and the IF registers are mapped as read-only pages. When a VM attempts to use this virtual device by writing to its interface, this action will trap an page-fault exception and can be detected by the VMM. Since VMs' usage of DPR accelerators are independent, our system introduces additional management mechanisms to dynamically handle VMs' request for DPR resources.

In Figure 2 the proposed management mechanism is described. The *Virtual Device Manager* is a particular software service in an independent virtual machine domain, which aims at allocating DPR resources to VMs. In the static part of the FPGA, a *PRR Monitor* is created and is in charge of interconnections between IFs and PRRs, and dynamically monitoring DPR accelerators and search for available solutions of DPR resource requests.

Each time that a VM accesses an unconnected IF, VMM immediately schedules *Virtual Device Manager* to handle this DPR resource request as *Request (vm_id, dev_id, prio)*, which is composed of the VM ID, the virtual device ID and a request priority (i.e. calling VM priority). This request is posted to the *PPR Monitor* on the FPGA side to search for an appropriate allocation solution, which include different methods:

- ***Assign*** *(prr_id)*: directly allocates PRR (i.e. *prr_id*) to VM. If device *dev_id* is not implemented in this PRR, a *Reconfig* flag will also be added.
- ***Preempt*** *(prr_id)*: no PRR can be directly allocated, but PRR (i.e. *prr_id*) can be preempted and reallocated. If device *dev_id* is not implemented in this PRR, a *Reconfig* flag will also be added.
- ***Unavailable***: this state means that currently no PRR is available. This unsolved request is then added to the searching list which *PPR Monitor* keeps searching for solutions. New solutions will be sent to *Virtual Device Manager* immediately.

Virtual Device Manager then performs the allocation according to the returned solution which may be performed in several steps: (1) disconnect the target PRR from other IFs, and change the memory page as read-only in the corresponding VM; (2) connect it to the requesting VM and update the memory page as read/write; (3) resumes the requesting VM to the interrupt location and continues running. In non-immediate solutions (i.e. *Reconfig* and *Preempt*), the allocation process can't be completed in one-shot execution, since it needs to wait for the completion of reconfiguration or preemption. In this case, *Virtual Device Manager* acknowledges VM software by releasing IPC messages, and leaves the *PPR Monitor* tracking these solutions on the FPGA side, which delivers interrupts to *Virtual Device Manager* when reconfiguration/prteemption is over.

Table 1. Overheads of DPR allocation

Methods	Overheads (μs)
$\{Assign\}$	$3.03\mu s$
$\{Assign,\ Reconfig.\}$	$6.76\mu s + T_{RFCG}$
$\{Preempt\}$	$5.10\mu s + T_{preempt}$
$\{Preempt,\ Reconfig.\ \}$	$9.96\mu s + T_{preempt} + T_{RFCG}$

3 Evaluation

Our evaluation focuses on the DPR allocation latency, i.e. the delay that occurs before the accelerator is properly allocated and ready to start. This latency comes from several sources: page-table faults handling, IPCs, VM scheduling and *Virtual Device Manager* execution. According to the experiment measurements, overall overheads for allocation path can be estimated as in Table 1. It can be clearly noticed that direct allocations can be efficiently performed with $3\mu s$ latency, whereas in other solutions the system overhead remains still quite low and the major costs come from preemption and reconfiguration, which are inevitable in most DPR systems.

4 Conclusion

In this paper we have introduced a framework capable of DPR resource management in a virtual machine system. DPR accelerators are mapped as ordinary devices in each VM. Through dedicated memory management, our framework automatically detects the request for DPR resources and allocates them dynamically with low overheads.

References

[1] G. Heiser, "The role of virtualization in embedded systems," in *Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pp. 11–16, ACM, 2008.

[2] A. Agne, M. Happe, A. Keller, E. Lubbers, B. Plattner, M. Platzner, and C. Plessl, "Reconos: An operating system approach for reconfigurable computing," *Micro, IEEE*, vol. 34, no. 1, pp. 60–71, 2014.

[3] O. Knodel and R. G. Spallek, "Rc3e: Provision and management of reconfigurable hardware accelerators in a cloud environment," *arXiv preprint arXiv:1508.06843*, 2015.