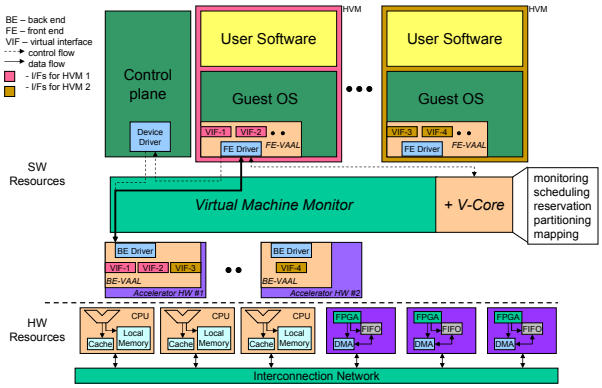
Virtualizing Heterogeneous Many-core Platforms

Greg Diamos, Ada Gavrilovska, Vishakha Gupta, Sanjay Kumar, Himanshu Raj, Karsten Schwan, Sudhakar Yalamanchili
CERCS, Georgia Institute of Technology
{*ada, vishakha, ksanjay, rhim, schwan, sudha*}*@cc.gatech.edu*

Introduction. The relentless progress of Moore's Law has periodically inspired major innovations – both in hardware and software – at specific points in time to keep performance growth on pace with transistor density. Industry has reached another such point as it encounters intellectual and engineering challenges in the form of power dissipation, processor-memory performance gap, limits to instruction level parallelism, slower frequency growth, and rising non-recurring engineering costs. As a consequence, when we consider how the large number of transistors that will be supplied at future technology nodes will be used to sustain performance growth, there are some inevitable trends, including i) replication of cores, ii) the use of high volume custom accelerators due to the fact that these devices have small footprint and dramatically less power consumption, and iii) innovations in memory hierarchies. The preceding collectively inspire the development of *heterogeneous many-core platforms (HVM)* – large scale, heterogeneous systems comprised of homogeneous general purpose cores intermingled with customized heterogeneous cores – accelerators, and using diverse memory and cache hierarchies. Such will be the case both on chip as well as in rack scale and multi-rack scale systems.

Virtualization for HVMs. Central to our approach is the ability to virtualize resources, specifically, accelerators. Towards this end, we propose several key concepts to support streaming applications composed in the manner suitable for execution on heterogeneous target machines. We encapsulate these concepts into the V-Core/VAAL run-time system. A software-centric view of the overall execution model of the system is illustrated in Figure . The system hardware is virtualized through a single virtual machine monitor (VMM) that manages a physical machine composed of multi-core processors and accelerators and can host multiple heterogeneous virtual machines (HVMs) executing in possibly distinct operating system environments.

The first key concept proposed here is that of *self virtualized accelerators (SVAs)*, a generalization of the self virtualized I/O devices described in [1], where multiple virtual accelerators can share a single physical accelerator. A virtual accelerator is accessed through a virtual interface (VIF) that consists of input queues and output queues for communication, VIF state information, and operations on the state and queues. When a HVM component is mapped to an accelerator, the input and output streams will map to a VIF for that accelerator. The implementation for a VIF has two components – one that is



implemented on the processor (the front-end portion) and a corresponding component that is realized in the accelerator (the back-end component). A single accelerator may host back-end VIFs for multiple HVMs that share the accelerator. A single front-end driver exists in each HVM for each physical accelerator and multiplexes all front-end VIFs for that physical accelerator. Similarly, a single back-end driver multiplexes all of the back-end VIFs for each accelerator. We have experimentally demonstrated the viability of the approach for I/O accelerators, represented via the Intel IXP network processors. Our current developments target self-virtualizing instances of FPGAs, NVidia GPUs and the Cell processor. Finally, the run-time implements functions to manage VIFs, including scheduling VIFs on an accelerator under appropriate time- or space-sharing constraints, depending on the capabilities of the target device. Importantly, the design of this layer diverges from traditional VM implementations in that the applications in the host VM have direct access to the accelerator bypassing the traditional driver domains that virtualize devices. These functions and the drivers collectively form the *virtualized accelerator abstraction layer (VAAL)*.

A second key concept is the extension of modern VMMs to support the execution of multiple HVMs onto a set of physical processors and accelerators. This is captured in the *virtualized core (V-Core) extension to the VMM*. V-Core implements several new capabilities. These include (1) reservation of physical resources, as specified by the HVM, (2) ability to dynamically create and manage VIFs, and (3) mechanisms to communicate QoS or service requirements, and co-schedule the virtual processors and accelerators in a HVM on the hardware, as well as maintaining QoS across HVMs.

1. H. Raj et al., *Scalable I/O Virtualization via Self-Virtualizing Devices*, GIT-CERCS-06-02, Georgia Tech