Virtualization and Security Aware Multi-core Architecture for Clouds

Mitsu Mehta, Bhanu Prakash Vempati, Tejas N, Ishwar Raut, Laskhmi J, S K Nandy
(Affiliation): CAD Lab, Indian Institute of Science (IISc.), Bangalore, India
Email: {nmitsu, bhanuprakash, tejas, ishwar}@cadl.iisc.ernet.in, jlakshmi@serc.iisc.in, nandy@cds.iisc.ac.in

Abstract

Cloud computing is fast becoming a pervasive technology as it has the potential to eliminate requirements for setting up high-cost computing infrastructure for Internet-based application services that form the crux of many industry workloads. Owing to these agility and efficiency benefits, the cloud is becoming the delivery model of choice for a growing number of enterprise services. Most cloud deployments are over virtualized infrastructure systems as these have already been tested for their resource allocation and management strategies. In a cloud computing setup, the entire data resides over a set of networked resources, enabling it to be accessed through virtual machines. Privacy and security challenges are among the most significant issues in cloud computing. Competing organizations may host their services over the same set of resources. The user-centric security model of old is unable to meet these challenges and a new trust model is needed. Most current virtualization techniques in cloud systems depend on resource management through a centralized software virtual machine management layer. This layer is rendered complex by the fact that the hardware is usually agnostic to the existence of multiple virtual entities competing for the same resources and the onus is on the software virtualization layer to ensure the sanctity of every transaction over such hardware resources. The other challenge stems from the time-sharing of resources which makes isolation and protection of shared resources more complex. These complexities make it difficult to apply formal methods of verification to ensure a trust base in a virtualized system. We propose a multi- core architecture that lends itself to spatial partitioning to avoid time-sharing of resources and a hardware resource allocation entity that creates independent spatial partitioning to avoid time-sharing of resources and a hardwareresource allocation entity that creates independent spatial partitions over which a complete virtual machine can run without interference from the host or other virtual machines. This paper explores a security aware multi-core architecture that utilizes spatial partitioning as opposed to time-sharing, to mitigate overheads in providing protected and direct access to the resources allocated to a virtual machine. The proposed architecture uses a Network-on-chip (NoC) as an interconnection solution for better scalability and performance. This NoC has been designed to provide a distributed framework for maintaining and enforcing resource boundaries for virtual machines and controlling access to the resources. The NoC provides isolation of resources

through spatial separation, consequently allowing a virtual machine to access its resources without the intervention of a central controlling entity.

Keywords

Virtualization; IaaS; Multi-core; Security

1. Introduction

Cloud computing is afflicted by some of the challenges of parallel and distributed computing. It also faces many major challenges of its own. The specific problems for different cloud delivery models may vary, but the common cause is the way utility computing is achieved using resource sharing and resource virtualization. This requires formulating a trust model that is significantly different from the user-centric model that has been the norm in non-virtualized systems. Gaining the trust of a large user base is critical for the future of cloud computing. It is difficult to envision current public cloud systems as a suitable environment for all applications. Highly sensitive applications related to critical infrastructure management, health-care applications and many real-time applications are most often hosted by private clouds or a hybrid cloud model.

The Infrastructure as a Service (IaaS) model is by far the most challenging to defend against attacks. Indeed, an IaaS user has much more freedom than the other two cloud delivery models. An additional source of concern is that the considerable cloud resources could be used to initiate attacks against the network and the computing infrastructure. In this paper, we shall be addressing the security and performance challenges faced by this model, especially from the perspective of system virtualization, as it a critical design option for this model [32, 33].

Virtualization solutions are gaining popularity as they present opportunities to better manage and more efficiently utilize computing resources. Current systems use emulation, para-virtualization or hardware-assisted virtualization with hosted or virtual machine monitor based mechanisms to facilitate resource sharing. These systems predominantly use temporal multiplexing of resources across virtual machines. With multi-core architectures becoming ubiquitous in virtualization platforms, the tasks of ensuring temporal isolation and sufficient execution time determinism gain a whole new level of complexity. Numerous solutions using well designed virtual machine monitors or host operating systems have been proposed. A large number of these systems suffer from considerable overheads that stem from the need for a centralized controlling virtualization layer to be invoked for every sensitive operation in order to provide temporal isolation along with certain performance guarantees. Moreover, the virtualization layer usually operates without specific information about applications or processes running in individual virtual machines. This disconnect of information between the application and the virtualization layer and subsequently the hardware requires extremely complex mechanisms at the virtualization layer to ensure proper sharing and isolation especially in the case of I/O virtualization.

The economics of server consolidation has caused a resurgent interest in machine virtualization [1, 2, 4, 5, 7, 9, 10]. System virtualization allows creation of virtual replicas of the physical system, over which independent virtual machines can be created, complete with their own, individual operating systems, software and applications. In principle, the two main drivers for using virtualization in systems are virtual machine performance and isolation on a consolidated server.

A number of popular virtualization solutions have been proposed using emulation, para-virtualization or hardware-assisted virtualization employing hosted or virtual machine monitor based mechanisms. All these techniques predominantly use time multiplexing for sharing resources across multiple virtual machines. However, because of this, these techniques present a single point of failure as the host operating system (OS) or virtual machine monitor is invoked every time a privileged operation is required by a virtual machine. Virtualization of Input/Output (I/O) resources has especially proved to be a significant challenge. Most virtual machine monitors prevent direct access to I/O devices to ensure proper sharing and protection of resources. The overheads in such schemes often result in significant performance and security penalties. Time-sharing of resources also considerably increases the complexity of the virtualization layer as the secondary task of resource usage management demands more participation from the virtualization layer as compared to the primary task of resource allocation.

The trusted computing base (TCB) of a virtual environment includes not only the hardware and the virtualization layer or hypervisor but also the management OS. You can allow migration and recovery by saving the entire state of a virtual machine (VM) to a file. However, this feature presents a problem when creating strategies to bring the servers belonging to an organization to a desirable and stable state. For example, an infected VM can be inactive when the systems are cleaned up. Then it can wake up later and infect other systems. Thus, benefits accrued from employing beneficial cloud computing technologies can often be overshadowed by the new issues they create.

The next major challenge is related to resource management on a cloud. Any systematic resource management strategy requires the existence of controllers to implement several classes of policies: admission control, capacity allocation, load balancing, energy optimization and the provision of quality of service (QoS) guarantees. In virtualized systems, these controllers form the crux of the hypervisor or the virtualization layer. To implement these policies, the virtualization layer needs to be aware of requirements of applications running on virtual machines. In most cases, the information shared by the guest OS is partial and application performance and security might get affected in the absence of accurate information at the virtualization layer.

It seems reasonable to expect that such a complex system would have to incorporate self-management principles. But self-management and self-organization raise the bar for the implementation of logging and auditing procedures critical to the security and trust in a provider of cloud computing services. In several cases, it also becomes difficult to identify the source of a security breach.

Most virtualization techniques rely on the indirection or abstraction provided by the virtualization layer to provide benefits portability, ease of migration, the ability to address a multitude of devices as a single virtual device, and many other benefits. However, this indirection comes at a cost. In most cases, there is a disconnect between the applications running in virtual machines and the physical hardware on which they have to execute. The hardware is agnostic even to the existence of multiple virtual machines and this is by design. With such a multi-level indirection in place, the onus is on the virtualization layer to incorporate techniques to provide appropriate isolation and performance guarantees to individual VMs. The primary function of creating a well-bounded set of physical resources and assigning them to a VM to suit its requests is vastly overshadowed with the software and hardware complexities in scheduling multiple VMs over the same physical hardware where the hardware is mostly unaware of the existence of multiple entities competing for the resources. Hence every sensitive operation has to traverse through the virtualization layer to ensure non-interference with the operation of other co-existing VMs. This results in overheads that affect the performance of applications running in VMs. It also makes the virtualization layer complex enough that malicious entities have the advantage of a larger attack surface. This is particularly challenging in the cloud space where establishing a suitable trust base can significantly impact the future of the cloud system. The next section describes these challenges in more detail.

In this paper, we propose an architecture designed to explore the benefits of spatial partitioning over time-sharing of host system resources. The proposed multi-core system architecture is designed with a view to support the physical separation and isolation of resources assigned to a virtual machine. Spatial partitioning of resources allows the creation of well-defined boundaries of isolation and performance benefits are accrued through allocation of dedicated resources to a virtual machine. The choice of a network-on-chip as a programmable interconnect has been influenced by the better scalability and performance offered by NoCs [14] and the distributed nature of NoCs that lends itself to support the creation of spatial resource partitions in a more fine-grained and reconfigurable manner [15].

The rest of the paper is organized as follows. In Section 2, we discuss performance and security issues in prevalent virtualization systems that would benefit from the spatial partitioning approach of the proposed architecture. In Section 3, we describe the proposed architecture and an implementation model of the architecture. Section 4 describes an emulation of the design on an FPGA platform along with test-benches to check how the design handles system isolation for a VM. This is followed by related work in Section 5 and finally by the conclusions in Section 6.

2. Motivation

This section analyzes application performance and security implications when hosted in prevalent virtualization solutions. We look at performance evaluation as it is a significant factor in indicating non-interference amongst VMs, which in turn contributes to the security challenges. Through the following experiments, we wish to show

how time-sharing of resources in a virtualized system causes a VM to impact the performance of other VMs that co-exist on such systems. A malicious VM can take advantage of such issues to launch denial-of-service attacks on other VMs and the entire system in general. This aims to highlight the importance of setting proper isolation boundaries around resources assigned to a VM. When the physical hardware is cognizant of these boundaries and plays an important role in preserving them, the responsibilities of the virtualization layer are minimized and the process of establishing a trust base is simplified.

The indirection introduced by the virtualization layer is beneficial in several ways. For example, decoupling a logical device from its physical implementation offers various advantages such as migration, replication, load balancing, multiplexing a physical device to improve hardware utilization, and composition of multiple devices into a single virtual device. However, as observed later in this section, this level of indirection also leads to more complex mechanisms that the virtualization layer has to employ to preserve the isolation and security for an individual VM.

2.1. Performance Evaluation of an existing Virtualization Solution

On virtualized servers, applications co-hosted for workload consolidation experience performance interference due to sharing of resources like memory, disk, CPU, and Network-interface card (NIC). This section describes experiments that were conducted to explore the effect of software virtualization mechanisms based on resource time-sharing constructs on application performance. The study explored the effect of time-sharing of system resources such as CPU, Memory and Disk on the performance of a VM. We have used KVM [19, 20] as the virtualization solution for the following experiments. Additional experiments conducted on other virtualization platforms [21, 22, 23] also show significant effects on performance due to the presence of the virtualization layer.

Experimental Setup

The experiments were conducted on a virtualized environment with KVM as the hypervisor. The host machine had a quad-core Intel core i7 Processor, 8GB RAM and 1TB SATA2 disk(7200 rpm, 16MB cache). A dedicated partition on the disk was used for each VM to minimize interference of host operating system. All the VMs were configured identically with 30GB virtual disk space, 1GB RAM and 1 vCPU (virtual CPU) pinned to an exclusive core so as to avoid any CPU scheduling effects. The non-virtualized counterpart was booted with 1 physical CPU and RAM restricted to 1GB. To examine interference effects, each individual resource was subjected to resource-specific benchmarks, in four scenarios, namely NVM (non-virtualized mode), 1VM (one VM mode), 2VM (two VMs mode) and 3VM (three VMs mode). Under NVM, the benchmark was exercised when hosted on a non-virtualized system. In 1VM case, the benchmark was exercised inside one VM that was instantiated on the virtualized platform. For the 2VM case, two independent VMs were instantiated on a virtualized platform with the resource under testing being shared between the two VMs, based on the isolation provided by the virtualization platform. Each of the VMs in this test exercised the same benchmark and the impact of resource sharing was measured by the performance effect on the benchmark metric. The case of three VMs is similar to that of 2VMs, except that three independent VMs were instantiated on the same platform. The benchmarks used to study the resource sharing effect include "nbench" for CPU interference, STREAM for memory interference and IOZONE for disk I/O.

Impact on Performance

The following graphs show the effect on the performance of each resource as the number of virtual machines sharing the resources is increased. In **Figure 1**, the CPU performance graph shows the normalized results with respect to the non-virtualized case. In **Figure 3**, disk performance is reported in terms of average throughput measured in KB/sec, and memory performance in terms of average memory bandwidth measured per VM in MB/sec as shown in **Figure 2**.

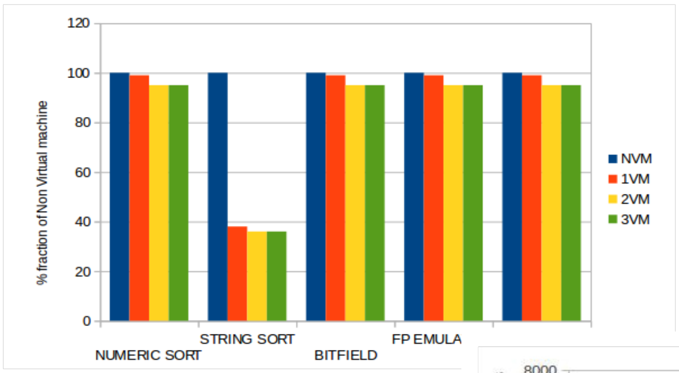


Figure 1. Relative CPU performance for different number of VMs.

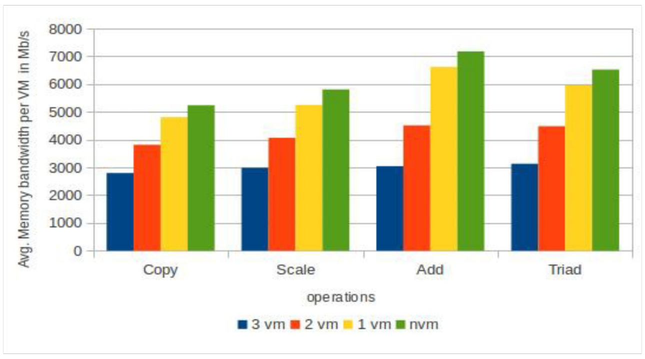


Figure 2. Average memory bandwidth per VM for different number of VMs.

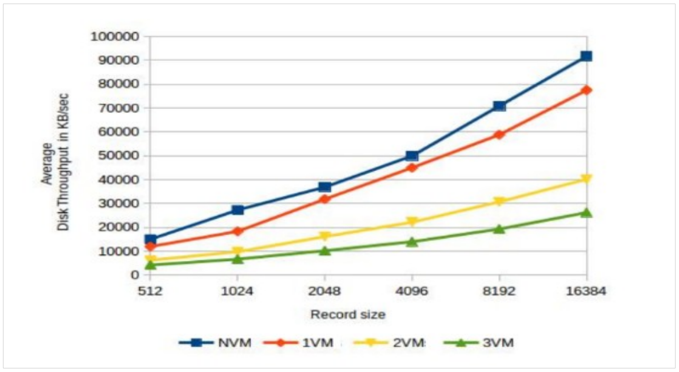


Figure 3. Average disk throughput with varying record sizes for different number of VMs.

CPU performance is not significantly impacted by the presence of additional VMs except in the case of string sort. This can be explained as string comparison does not follow spatial locality and multiple levels of address translation add to cache miss penalties. Memory and I/O operations are significantly affected in the presence of multiple virtual machines. The overheads incurred by the software virtualization layer can be seen by comparing with the non-virtualized (non-VM) case. We draw the analogy of VM provisioning in our proposed architecture to the case of the non-virtualized result. This is intuitive since we adopt space-sharing as the principle behind resource sharing in our solution. Application performance would therefore benefit significantly by spatial partitioning.

2.2. Security Concerns in prevalent Virtualization Systems

Most of the security challenges in prevalent virtualization system stem from a centralized and complex virtualization layer that is tasked with managing time-shared resources. The hardware is not cognizant of the presence of multiple virtual entities vying for the same set of resources. Such definitions are maintained primarily by a software virtualization layer and every access to time-shared resources has to be essentially controlled by this virtualization layer. Time-sharing of resources does not lend itself to defining neat isolation boundaries around hardware resources. Logical boundaries and controls have to be maintained and exercised by the software virtualization layer to facilitate the notion of a dedicated physical system for each VM.

For example, in the case of an I/O device access, the request has to be processed by the entire guest OS stack responsible for I/O requests and at the end of it, a new request is generated that traps to the virtualization layer. This request now has to traverse the I/O stack of the virtualization layer before it can be directed to the appropriate physical device. The virtualization layer schedules requests from multiple VMs onto a physical device usually via a device driver managed by the virtualization layer or a privileged VM with direct access to the physical device. The device itself is not aware of the multiple ownership of these requests. When the device finishes processing the request, the two stacks must now be traversed again in the reverse order. The I/O device raises a physical completion interrupt that traps to the virtualization layer, which then determines which VM has to be notified and in turn raises a virtual interrupt for the virtual device associated with that VM.

It becomes imperative for the virtualization layer to provide some measure of performance isolation or quality-of-service controls to satisfy requirements of mixed criticality applications running on multiple VMs time-shared over the same set of resources [23].

A list of some common security and isolation problems that afflict current virtualization systems and possible solutions to mitigate such problems are listed in **Table 1**. This list was arrived at after a survey of different workloads on consolidated server platforms, the vulnerabilities inherent in such systems and concerns reported by users of these systems. The list includes some well-known problems that affect almost all types of workloads.

Table 1: Security issues in prevalent virtualization systems and possible solutions

S. No.	Problem	Proposed Solution
1.	Malicious VM: One VM trying to access resources of another VM	Define clear physical boundaries for a VM and isolate it within those boundaries
2.	Denial-of-Service (DoS)	Well-defined and enforced limitation on resource allocation to each VM
3.	Non availability or malicious use of device access paths	Complete path isolation for device access paths associated with each VM
4.	Attacks on the virtualization layer	Make the attack surface of the virtualization layer as small as possible. Reduce or eliminate intervention by the virtualization layer for sensitive operations
5.	Packet Sniffing between VMs	Physical dedicated channel for each VM (split NiC) with path isolation
6.	Different security levels for each VM	Ensure complete non-interference from other VMs so different security levels can be established

It is possible to conclude that most of the problems stem from the lack of well-defined hardware boundaries for system resources assigned to a virtual machine. In a system using time-sharing, significant overheads appear as the resource boundary definitions reside in the software virtualization layer and often don't reflect at all in the underlying hardware. The virtualization layer has to perform several additional tasks to preserve resource separation and isolation for virtual machines and intervention is required for every sensitive operation.

The proposed architecture described in the following section aims to address the above performance and security issues by creating and enforcing well-defined resource boundaries through spatial partitioning. The hardware components of the system create and enforce resource boundaries. Enforcing and monitoring these boundaries is done in a distributed manner using the components of the NoC and the need for intervention from a central controlling entity is obviated for memory and I/O accesses. While multi-core architectures are gaining prevalence and virtualization promises to be the way forward for improved resource utilization, emerging security concerns and the requirement for application performance guarantees indicate the need for exploring whether spatial resource partitioning mechanisms can deliver better trade-offs.

3. Virtualization and Security Aware Multi-core Architecture

A virtualization layer typically performs the following important tasks:

1. Starting and stopping virtual machines – allowing virtual machine to boot on a set of virtual resources and view those virtual resources as a physical system. The virtual resources are mapped to the physical resource by the virtualization software and relinquished once the VM is stopped or exited.
2. Scheduling virtual machines on processors – Processors are time-shared among the VMs and hence scheduling becomes the responsibility of the virtualization layer.
3. Memory management – The virtualization layer allows the guest OS to assume it is accessing a physical memory. In reality, this is a virtual space and requires a second level of address translation to map to the actual physical memory space on the host system. Second level address translation adds significant performance penalties and has been somewhat mitigated by hardware assistance using Intel's extended page tables (EPT) or AMD's nested page tables (NPT).
4. Emulation and arbitration of accesses to I/O devices – To support the illusion that the VM is running over a physical machine, an emulated device is exposed to the guest OS on the VM. The virtualization layer is responsible for translating this emulated device onto the actual physical device. Hence, each I/O access necessitates an intervention from the virtualization layer.

In most existing systems, the virtualization layer is quite complex as it has to perform all the above activities in a software layer. This exposes a much larger attack surface and makes it a single point of failure.

In the proposed architecture, we split the functionality of this complex virtualization layer into resource allocation and resource management activities. Resource allocation is facilitated by a hardware entity called a system configurator whose only task is to maintain a list of available physical resources, carve out a subset of those resources as a system tile and allocate to a VM at the time of VM creation. When the VM exits, the resources allocated to the VM are folded back into the list of available resources. Clear boundaries are established on the resources allocated to a VM. The set of resources typically includes CPUs, memory segments, disk and network devices. Once assigned to a VM, these allocated resources can be directly controlled by a VM just like it would control the resources in a physical system. Each system tile also includes the components of the NoC that bind this subset of resources and provide the necessary means of communication. Resource usage monitoring at each system tile is accomplished using these NoC components associated with that system tile. They are responsible for enforcing and honoring the resource boundaries for the VM.

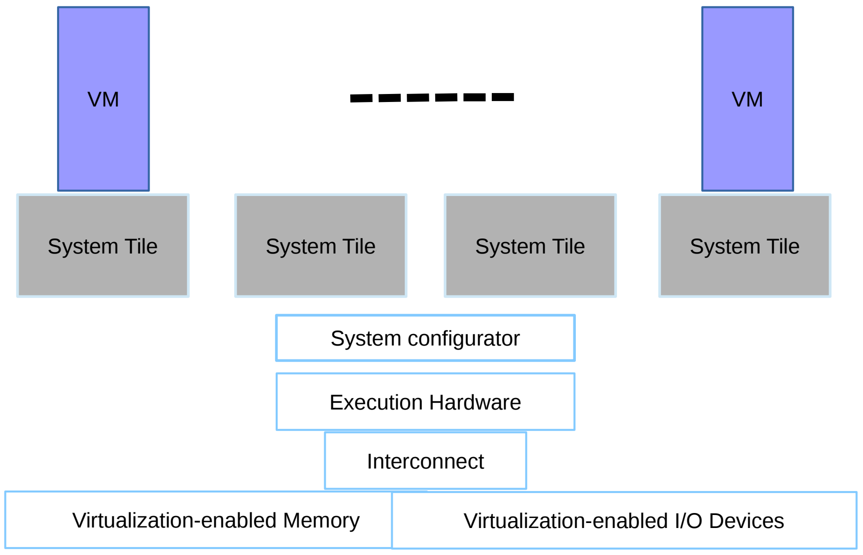
The proposed NoC is a key element in two ways:

1. It provides complete path isolation for all communication in and out of resources allocated to a virtual machine.
2. It checks and enforces the boundaries assigned to the virtual machine resources for every data transaction.

These and other components of the architecture are described in greater detail in the rest of the section.

3.1. System Architecture

Figure 4 shows the physical components of the architecture and the system tile abstraction that can be defined over the physical resources. The system configurator facilitates the creation of these isolated and secure system partitions called system tiles. A system tile is a hardware abstraction of a subset of the system's physical resources with clearly defined resource boundaries. It is created at the time of VM initiation based on the resource requirements specified by the VM.



3.2. System Tile Abstraction

A system tile is defined as a logical partition of the complete system hardware allocated to a virtual machine. **Figure 5** shows a representation of the notion of a System tile. A full-fledged operating system (OS) can be hosted on a system tile. A system tile is a logical container for processing cores, memory contexts and I/O device contexts assigned to a VM. Once a system tile is created, the resources contained in the tile are available for direct access by the guest operating system (guest OS) of the virtual machine to which the tile is assigned, without intervention from the system configurator. The processing cores shown here are actual physical CPU cores. The memory context defines the range of the physical memory of the host system that a VM can access. Any access of memory beyond this allocated range is flagged by the system as an access violation. An I/O device context describes the portion of I/O device that is accessible or allocated to a VM. Any access to device memory beyond this range is also flagged as an access violation.

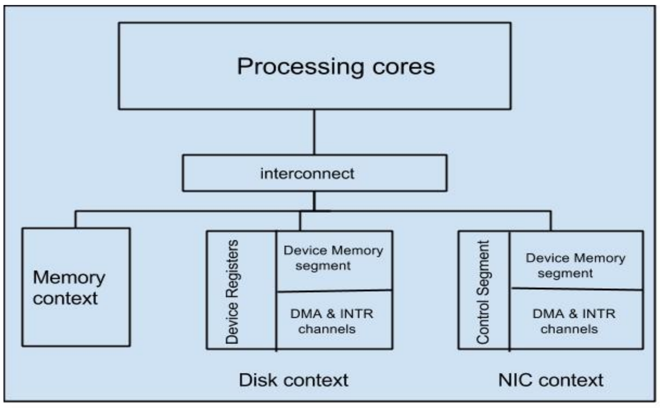


Figure 5: Abstraction of a System tile and its components

3.3. System Configurator

The system configurator is a hardware entity that maintains resource tables that give an idea of the total system resources, records the subset of allocated resources per system tile and identifies the tile associated with each

VM. The system configurator is cognizant of the notion of a VM and maintains its identification by a VM-ID. Each system tile is associated with a single VM-ID.

The system configurator performs the following functions:

1. VM_Init: Check if requested resources are available, create a system tile logically encapsulating this subset of resources, bind the system tile to the VM_ID and provide a boot record for the guest OS on that VM to boot over that system tile.
2. VM_Exit: Initiated when a VM has completed execution, all the resources assigned to the system tile associated with the VM_ID of the exiting VM are relinquished and marked as available in the resource tables maintained by the system configurator.
3. VM_Scale: If a VM wishes to request for additional resources or relinquish certain resources. In case of additional resources, the request is satisfied by the system configurator only if the additional resources requested are available in the resource tables. These resources are then added to the system tile and marked as unavailable in the system configurator's resource tables. If resources are relinquished, they are decoupled from the system tile and marked as available in the resource tables.

These functions are exposed as a programmable interface by way of which an external entity like the cloud resource manager can issue instructions to the system configurator for gathering system status and usage information, send out commands for instantiating and shutting down VMs based on their resource demands and lifetimes. The system configurator creates a system tile only when a VM creation request arrives and has fine-grained control over marking resource boundaries for the system tile. Once a system tile is defined for a VM, the system configurator binds the tile to the corresponding VM-ID and initiates the boot sequence for the guest OS.
A system configurator with this limited functionality can be viewed as a simple finite state machine. It can be analyzed through formally provable methods for security and reliability and thus lend itself to the building of trust-based systems. A system configurator implemented in hardware can be protected using hardware protection mechanisms such as tamper-proofing or using a trusted platform module (TPM) [35].

3.4. System Fabric and Interconnection Elements

Figure 6 shows the conceptual architecture of the system fabric. The system fabric consists of CPU cores, a programmable interconnect for communication and controllers for memory and device access such as the MCI (Memory Controller Interface), DCI (Disk Controller Interface) and NCI (NIC or Network Interface Card controller interface). The red rectangles represent the CPU cores. The blue box just below the CPU core is the cache associated with the core. The system fabric is composed of several processing cores (like those in a single socket) connected to a set of memory banks using an MCI.

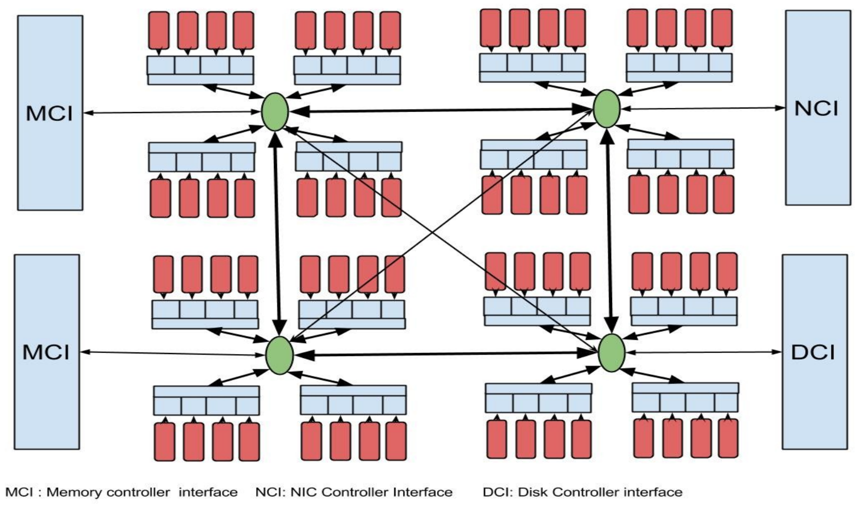

MCI
NCI
MCI
DCI
MCI : Memory controller interface
NCI: NIC Controller Interface
DCI: Disk Controller interface

Figure 6: Conceptual System fabric with cores, interconnects and memory and device interfaces

The physical memory in this sense has some notion of affinity to a socket. The memory requested by a system tile is normally assigned out of the banks that have closer affinity to the cores allocated to the tile. When a system tile comprises of multiple processor cores, the memories associated with the cores together form the memory context for the system tile. If a processor requires access to any memory in the memory context, it has to go through the interconnect. The router attached to that core checks each request and only allows valid requests to propagate forward. The blue horizontal bar below the last level cache represents the interface to the on-chip NoC. The interconnection routers represented by the green ovals provide communication amongst cores and connect the cores to all other system resources. The MCI/DCI/NCI are control interfaces that ensure the sanctity of memory and device contexts. Virtualization-enabled devices would have several sets of logical device level contexts that are associated with actual hardware entities. Based on the arbiter's selection of a device context, one of the sets is dynamically associated or enabled for the data transfer. Hence these control interfaces have context configuration information and also the control to ensure the sanctity of these contexts with regard to their access to a system tile. These contexts are essentially identified by the specific memory or device address ranges.

3.5. Resource Isolation through Spatial Partitioning

One of the chief goals of the architecture is system isolation for achieving performance and security benefits. With that view in sight, the design ensures that the allocation constructs for the resources are such that no two VMs have any common resource shared among them. This is achieved as follows.

1. Physical CPU core allocation to a VM is exclusive.
2. Separate physical segments of memory are used to create memory contexts. There is no shared memory space between two VMs.
3. Every device context is also exclusive to the VM. There is no shared device context across VMs.

Each memory or device context is basically a small spatial partition of the basic resource.

Figure 7 shows a representation of a virtualization enabled device that exposes multiple device contexts. The device is aware of the existence of multiple VMs and associated a device context with only one VM. Device resources such as memory and registers are not shared across contexts. **Figure 8** shows a device context with its own set of registers, DMA and interrupt channels and associated memory segment.

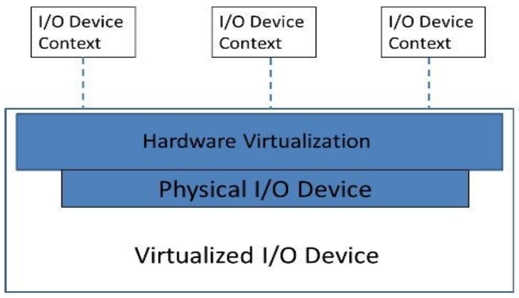


Figure 7: Conceptual virtualization enabled device with spatially separated device contexts

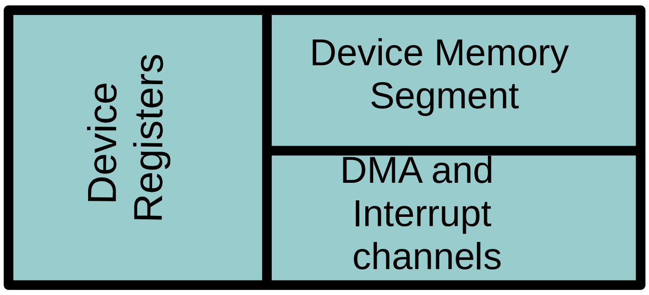


Figure 8: Representation of a device context

4. Emulation and Testing

In Section 2, we demonstrated performance benefits of the proposed architecture by drawing an analogy with the non-virtualized case. The spatial partitioning constructs of the architecture allow each VM to be hosted on its own partition within the physical system and allow it to control its resources directly. The rest of this section describes an emulation of the proposed architecture on Xilinx Virtex 7 XC7V2000T.

We have chosen Chisel [16] as the base language in which to implement our architecture design. Chisel's use of Scala [17] lends it easy extension and customization. Chisel generates regular Verilog code which was adapted to the emulation platform.

4.1. Architecture Description of the Emulation Model

Described here, is an overview of the system built to demonstrate spatial partitioning of available resources to accommodate multiple VMs. This multiprocessor system uses MicroBlaze^{TM} cores as Processing Elements (PEs) and a custom built NoC as the interconnection fabric. We have used the Xilinx Memory Interface Generator (MIG) IP to connect DDR3 memory to the system and UARTLite IP to perform basic I/O with PEs. An AXI Timer IP was used to measure the execution times for various benchmarks run on the system. For system verification and debugging, we used the Xilinx Integrated Logic Analyzer (ILA) cores. We used AXI4-Lite as the means for interfacing these modules. **Figure 9** shows the system architecture and various modules used.

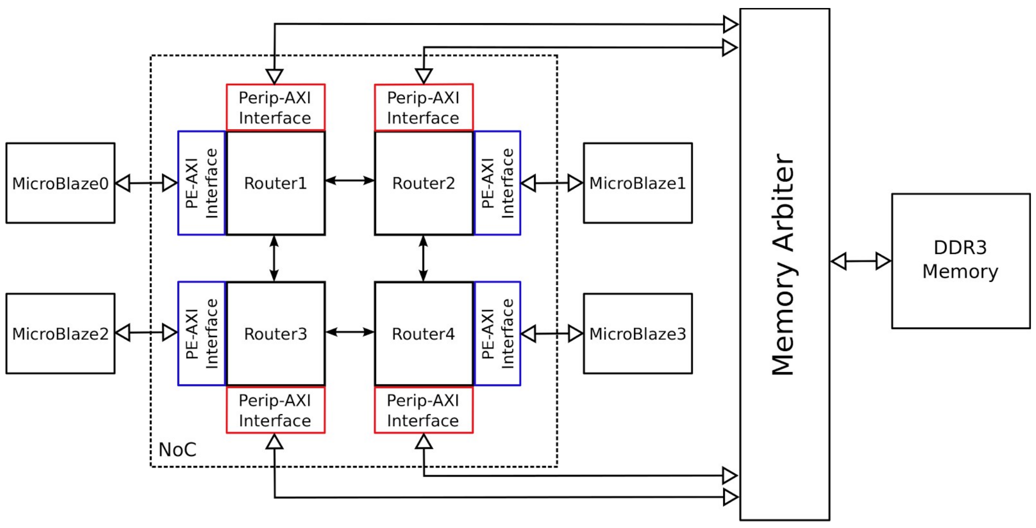


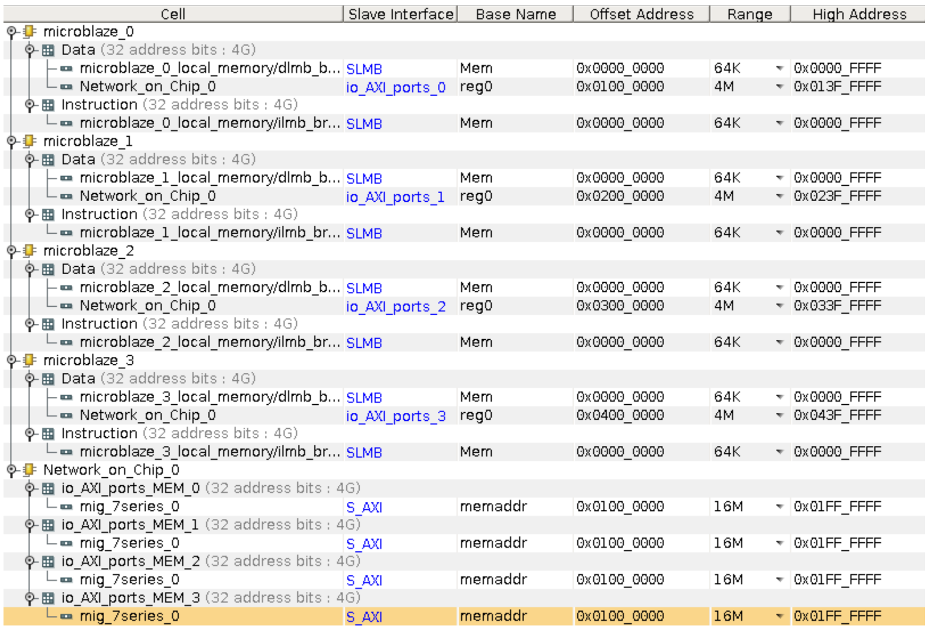
Figure 9: Block diagram of the emulation model

4.2. Address Mapping and Sytem Specifications

Since all the peripherals are memory mapped we need an address map. **Figure 10** shows how the peripherals are mapped to 32-bit address space of the MicroBlaze^{TM}. The entire system has been implemented on a Virtex-7 FPGA. **Table 2** lists the timing and resource utilization of the design.

Table 2: Timing and resource utilization specifications

Max clock frequency	100 MHz
Slice LUT	25225
Slice Registers	18102



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_b...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network_on_Chip_0	io_AXI_ports_0	reg0	0x0100_0000	4M	0x013F_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_br...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
microblaze_1					
Data (32 address bits : 4G)					
microblaze_1_local_memory/dlmb_b...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network_on_Chip_0	io_AXI_ports_1	reg0	0x0200_0000	4M	0x023F_FFFF
Instruction (32 address bits : 4G)					
microblaze_1_local_memory/ilmb_br...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
microblaze_2					
Data (32 address bits : 4G)					
microblaze_2_local_memory/dlmb_b...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network_on_Chip_0	io_AXI_ports_2	reg0	0x0300_0000	4M	0x033F_FFFF
Instruction (32 address bits : 4G)					
microblaze_2_local_memory/ilmb_br...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
microblaze_3					
Data (32 address bits : 4G)					
microblaze_3_local_memory/dlmb_b...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network_on_Chip_0	io_AXI_ports_3	reg0	0x0400_0000	4M	0x043F_FFFF
Instruction (32 address bits : 4G)					
microblaze_3_local_memory/ilmb_br...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network_on_Chip_0					
io_AXI_ports_MEM_0 (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x0100_0000	16M	0x01FF_FFFF
io_AXI_ports_MEM_1 (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x0100_0000	16M	0x01FF_FFFF
io_AXI_ports_MEM_2 (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x0100_0000	16M	0x01FF_FFFF
io_AXI_ports_MEM_3 (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x0100_0000	16M	0x01FF_FFFF

Figure 10: Address mappings used for mapping peripherals to MicroBlaze™ cores

4.3. MicroBlaze™

The MicroBlazeTM embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs). Each microblaze processor was instantiated with the configuration described in **Table 3**.

Table 3: MicroBlaze™ Configuration specifications

Block RAM	64KB
Cache	Disabled
Debug module	Disabled
AXI Data port	Enabled
Interrupt controller	Disabled

The code section of the programs run on MicroBlaze™ are stored in block RAM which is connected using the Local Memory Bus (LMB) interface. The data section is stored in DDR3 memory.

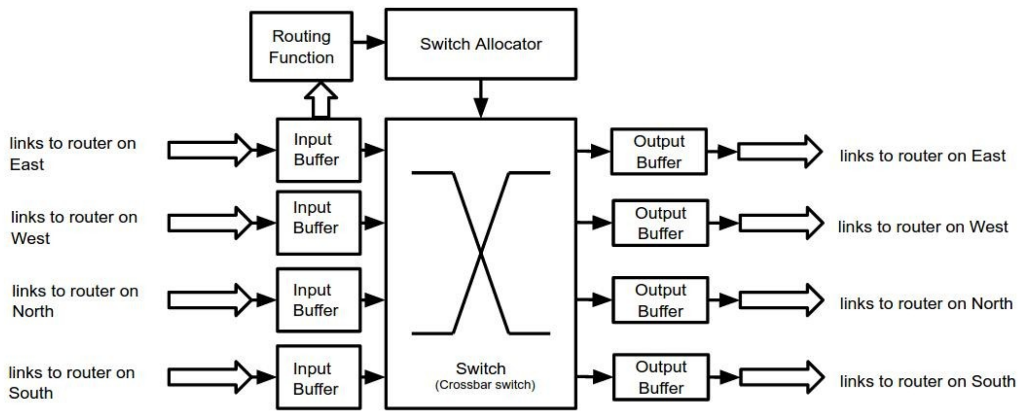


Figure 11: Architecture of each router in the NoC grid

4.4. Network on chip (NOC) and the Routers in the NoC

The NoC topology we used is a standard and basic grid which can be parameterized to support any size grid of routers (viz., 2x2, 3x3, and so on). **Table 4** gives the details about the configuration of each router in the NoC.

Table 4: Specifications for Network on chip

Routing Algorithm	Dimension Order Routing
Flow Control Mechanism	Credit Based
Switch Arbitration	Round Robin
Virtual Channels	Not supported
Radix	7 (4-Router to Router, 1-PE, 1-Mem, 1-I/O)
Buffer size	16
Cache coherency	Not supported
Switch architecture	Crossbar switch

The NoC routes flits from one router to another through channels connecting the routers. We used very basic routing mechanism which is Dimension Order Routing (DOR) to keep it simple. In order to support spatial partitioning we implemented the LBDR [18] framework. **Figure 11** shows the architecture of a router in the NoC and its component modules.

4.4. Interface to connect a PE to an NoC Router

The NoC routes only flits, so the incoming read and write transactions has to be converted into flits and routed through the NoC. This conversion is done by the PE Interface module. It is designed as an FSM that reads the address and data from AXI4-Lite port and converts them to flits. Similarly, for response to a PE request, flits are converted into a valid AXI4-Lite transaction. This interface generates a unique ID for every request it receives from the PE so that the correct response is sent for every request. **Figure 12** shows the FSM for the interface.

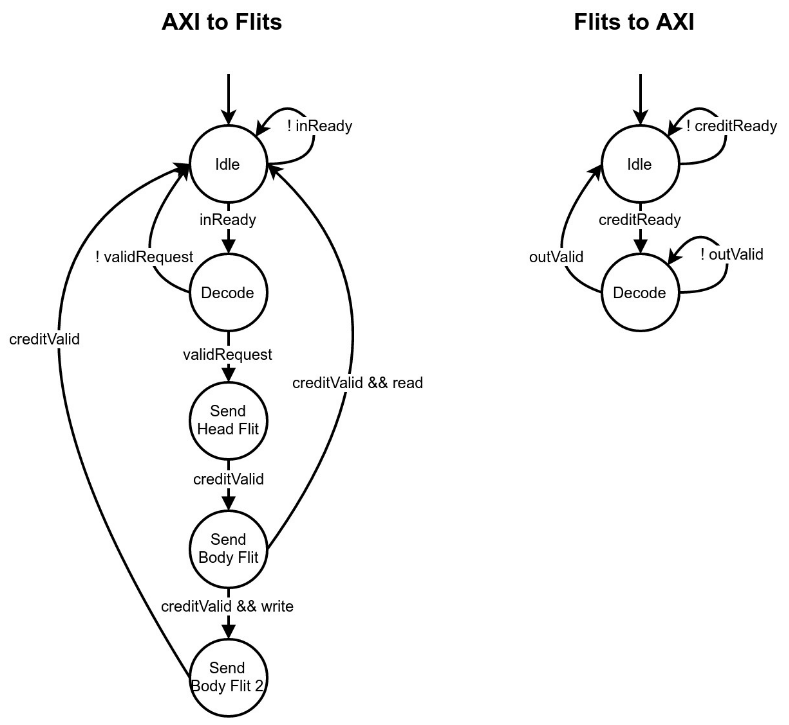


Figure 12: FSM for the PE to NoC interface

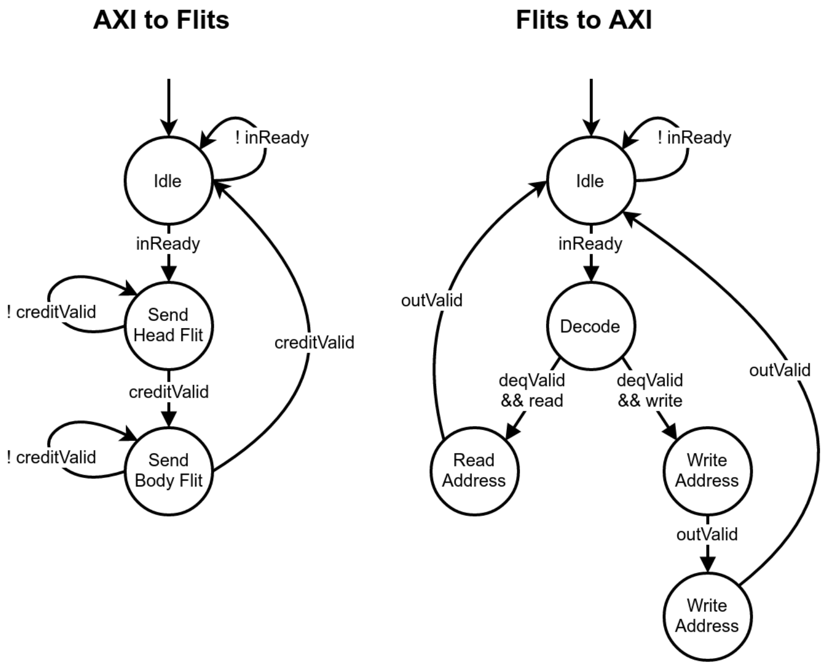


Figure 13: FSM for the interface between peripheral device and NoC

4.5. Interface to connect a peripheral to the NoC

PEs initiate read and write requests and expect a response from the peripheral devices. This interface is designed to facilitate this communication. The interface converts flits to valid AXI4-Lite transactions when a PE generates a request and the reverse conversion for a response from a peripheral device. The interface tracks each request by a unique request ID so the correct response can then be communicated to the requesting PE. **Figure 13** shows the FSM for this interface.

4.6. Peripherals and Debugging

A DDR3 memory was interfaced with the design using MIG with Data width (in bits) set to 512. Re-ordering of accesses and burst mode were disabled.
A UART module was used for basic I/O. It was configured with a baud rate of 9600 Bd. The Data bits were set to 8 and Stop Bits to 1.
Integrated Logic Analyzer (ILA) cores were used to view the various AXI transactions. The log generated is useful when debugging for errors.

4.7. Testing and Evaluation Framework

The NOC has been designed to provide support for spatial partitioning as described in the system design.
The memory and I/O devices are connected to the NOC using AXI4-lite interfaces. A simple system configurator has been designed that can initialize the resources required for a VM (viz. PEs, memory etc).This system has been tested for isolation of system resources such as PEs, memory and interconnection elements. The current emulation model assumes memory mapped I/O and hence the same arbitration mechanism suffices for memory as well as I/O. Future improvisations would include OS support on the PEs and connection to actual I/O devices that can support device context. In its current manifestation, the model aims to reinforce the notion of complete isolation provided by spatial partitioning of resources.
Each VM is allocated with some memory space and IO bandwidth which is not shared among other VMs. This memory space is represented using a Base and a Bound register. We treat I/O devices in a similar manner as they are assumed to be memory-mapped. The resource boundaries are preserved by checking each memory and I/O request for the origin of the request and whether it falls within the ranges allocated to the system tile. The memory and device interfaces such as the MCI, DCI and NCI are implemented as arbiters in the current implementation. They are responsible for scheduling memory and I/O requests by providing dedicated channels to and from the NoC and appropriately sized buffers per VM to ensure a fair sharing of resources and to prevent denial-of-service attacks or unrestricted device usage by malicious VMs.
Each router of the NoC has a PE (Processing Element) port that connects to the corresponding processor attached to the router, a memory port and an I/O port. Each of these ports are bidirectional. At the time of VM creation, a system tile is allocated with a set of processors. This implies that the set of those routers which are attached to the allocated processors is also allocated for the VM. Path isolation is achieved as a boundary is created around this subset of routers using a routing technique called Logic Based Distributed Routing (LBDR) [18]. For the proposed NoC, path isolation is achieved when the LBDR connection bits (Ce, Cw, Cn, Cs) are set by the system configurator. This allows the system configurator to setup the routing for the VM in such a way that all invalid input port to output port combinations are turned off by judiciously setting these connection bits. This ensures isolation of the available resources and non-interference among VMs.
Each memory module that is connected to the memory port is said to have an affinity to its own respective processor. When a system tile is formed with more than one processor, the collective memories form a globally addressable system tile memory for all the processors. Access to the this system tile memory is facilitated by the routers associated with the system tile and is akin to a NUMA-like access. Each router carries all the base-bound pairs that define the individual segments that constitute the global memory for the tile. For every memory or I/O request from the processor, the destination address is compared against the base and bound registers and the request is validated. If the request is invalid it is rejected and an exception is raised. For a valid request, the

destination is set by a Look-up Table (LUT) depending on the address requested and then Dimension Ordered Routing (DOR) is used to route the packets to the destination.
The base and bound registers and the LUT are populated by the system configurator when a VM is instantiated and cannot be accessed except by using the hardware constructs exposed by the system configurator.
Since current IO devices do not support the notion of device context, we implement an arbiter that provides the abstraction of a device context. The arbiter arbitrates between all the I/O-ports of the NoC.

4.7. Security and Isolation

We revisit the security problems listed in **Table 1** in Section 2 and describe the key design components of the architecture that aid in mitigation of these concerns.

1. *Malicious VM trying to access another VM's resources:* There is no sharing of processor cores across VMs. Path isolation provides spatial provisioning of communication resources. A memory or I/O device access is validated by the NOC before being propagated. Thus resources of a VM are completely isolated and protected from the rest of the system.

2. *Non-availability of device access paths or Denial- of-Service (DoS) attacks:* A malicious VM may try to launch a DoS attack by using an unfair amount of the system resources (I/O). This is prevented by the concept of device context which ensure that no particular I/O resource is unfairly occupied by a VM.

3. *Attacks on the virtualization layer:* The resource allocation responsibilities of a software virtualization layer are mapped to a hardware system configurator and the resource management is through the routers assigned to the system tile for a VM. The software virtualization layer constitutes just a simple manager to start and stop VMs. So the attack surface is greatly reduced.

4. *Packet sniffing or Spoofing:* The interconnect is designed in such a way that the NoC router to which a PE is attached checks each request going out of the PE. The source address is inserted by the router into the outgoing packet so a malicious VM cannot force a PE in its system tile to spoof the source address. Path isolation discourages packet sniffing.

5. *Maintaining different security levels for each VM:* Since each system tile is an isolated space with well-defined boundaries, the use of the resources within the system tile is completely independent of other system tiles and hence it is possible to define different security levels for different VMs.

Testing for system tile isolation

Experimental set-up: An NoC grid topology of 2X2 routers was implemented to support a maximum of four VMs. We have one MicroBlaze^{TM} and a memory space of 4MB attached to each of the routers. A peripheral UART was attached as an I/O to the NOC to view the output from the MicroBlaze^{TM} processors. This entire system was implemented on a Virtex-7 FPGA. The processors are used as a standalone system and no Operating System is booted. We used Xilinx SDK to run programs on MicroBlaze processors.
The following tests were conducted to check for the sanctity of system tile boundaries and to check for path isolation.

Test 1: A VM trying to access resources outside its boundaries: The configurator instantiates 4VMs in the system so that each VM have a MicroBlaze^{TM} processor and 4MB of memory space allocated. A program on each processor generates read and write requests for the entire 16MB of memory. The program gets response from only the 4MB memory that its VM has access to and for the remaining requests an exception was received.

Test 2: Non-interference between VMs: We instantiated four VMs. In the first instance, the STREAM benchmark was run only on one MicroBlaze^{TM} processor and in the second instance it was run on all four processors independently at the same time. In each case, the time taken for completion of the program was the same.

5. Related Work

Cloud security has been a topic of significant research as cloud computing gains momentum. Various studies and surveys have been performed for different cloud delivery models [28, 29, 30, 31, 32]. Standards bodies such as NIST have created guidelines such as the NIST Cloud Computing Reference Architecture [27].

In this paper, the focus is on the IaaS (infrastrucure-as-a-service) model. Virtualization has been successfully adopted by most IaaS implementations. Virtual machine monitors (VMMs) and hypervisors have been studied extensively. VMware's vSphere ESXi, Microsoft's Hyper-V, and open source solutions such as Xen and KVM [3, 6, 7, 11, 19, 20] are among the most popular tools for workload consolidation.

A major vulnerability in commodity virtualization systems is the need for implicit trust in the virtualized platform. In a physical system the OS trusts the hardware to a large degree. Likewise, in a VM, the operating system is required to trust the virtual hardware. The virtualization layer presents a single point of failure, and a malicious, compromised, or otherwise problematic entity may observe or interfere with the VMs. Thus secure virtualization relies on the authenticity and integrity of the virtualization layer, and in some cases upon the security or identity of the underlying hardware [8]. In existing virtualization solutions, the VMM and underlying hardware are especially vulnerable due to the time-sharing of resources. The onus is on the VMM to ensure that hardware resources requested by a VM are protected and available to meet the performance requirements of the VM. This induces significant overheads as the VMM has to closely monitor operations of each VM and employ protection measures to secure its own boundaries and those of individual VMs.

There are several examples in literature that prove the possibility of cache-based attacks, unrestricted access to I/O devices and denial-of-service attacks that stem from time-sharing of resources using a centralized control entity for resource management [24, 25, 26, 36, 37].

Most of the work is related to securing the virtualization layer and simplifying it to make it provably secure. Some of the work in this area includes the Terra Hypervisor [13], and sHype [12] from IBM. The sHype is a secure hypervisor architecture whose goal is to provide a secure foundation for server platforms. These mitigation strategies result in a complex virtualization layer as time-sharing of resources does not lend itself easily to a clean separation of physical resources. The increased complexity makes it more difficult to use formal methods of verification to establish a trust base in such systems.

In the proposed architecture, we introduce a paradigm shift from time-sharing of resources to spatial partitioning of resources. The key functions of the hypervisor or VMM are now handled by the system configurator and the NoC. The system configurator manages the allocation of resources. The NoC provides a framework for the VM to directly access the resources allocated to it. The system configurator can be secured by hardware measures that prevent any unauthorized modifications or interference in the operation of the system configurator. The distributed control provided by the routers of the NoC and the spatial partitioning aid in fault containment and non-interference. The behavior or execution of any single VM does not affect the operation of any other VM or the system configurator.

6. Conclusions

The future of cloud systems depends to a large extent on the trust base provided by the system. In most cloud infrastructures, virtualization is the driving force for resource allocation and management. The indirection provided by a virtualization layer in current systems has its own set of benefits and is necessary in time-shared systems that have multiple VMs vying for the same set of resources. In most current virtualization systems, this is achieved using a very complex virtualization layer that is involved in many aspects of hosting virtual machines as resource management has to necessarily be a function of the virtualization layer. Several approaches have been proposed to make this virtualization layer more secure through innovative mechanisms and hardware support. However, most of these mitigation strategies contribute to increased complexity in the virtualization layer and affect application performance. The increased complexity also makes these systems vulnerable to numerous new security challenges. The focus of this paper is to examine a virtualization and security aware architecture that uses spatial partitioning to cleanly separate the tasks of resource allocation and resource management so that a lean virtualization layer can be used to ensure a proper trust base for the cloud system. This separation allows the host system to allow a VM to directly control the set of resources assigned to it. A complex virtualization layer is replaced with a simple and minimalist resource allocation layer. As a

consequence of these isolation mechanisms being a part of the architecture design, it is possible to provided tangible isolation and non interference benefits for applications running on such systems. With reduced overheads and spatial separation, the architecture provides application performance and security benefits.
This is supported by the following key design aspects of the architecture:

1. Dedicated resources for each VM.
2. Hardware enforced resource partitioning.
3. Dedicated I/O device contexts that are controlled by the guest OS on the VM directly.

This architecture, thus, provides the basis for performance and isolation similar to dedicated physical systems while allowing multiple VMs to be hosted on separate spatial partitions on the same physical system.

References

[1] K. Adams, and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2006.

[2] Advanced Micro Devices, "*Secure Virtual Machine Architecture Reference Manual*," May 2005. Revision 3.01.

[3] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of USENIX Annual Technical Conference, FREENIX Track*, pp. 41–46, 2005.

[4] W. Chen, H. Lu, L. Shen, Z. Wang, N. Xiao, and D. Chen, "A novel hardware assisted full virtualization technique," in *Proceedings of 9th IEEE International Conference for Young Computer Scientists (ICYCS),* pp. 1292–1297, 2008.

[5] Y. Dong, Z. Yu, and G. Rose, "SR-IOV networking in Xen: Architecture, design and implementation," in *Proceedings of the Workshop on I/O Virtualization*, 2008.

[6] F. Guthrie, S. Lowe, and K. Coleman, "VMware vSphere design," John Wiley & Sons, 2013.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of Symposium on Operating Systems Principles (SOSP),* Oct. 2003.

[8] R. Perez, L. van Doorn, and R. Sailer, "Virtualization and hardware-based security," IEEE Security & Privacy Magazine, Volume 6, Number 5, pp. 24–31. 2008.

[9] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor," in *Proceedings of Workshop on Operating System and Architectural Support for the On Demand IT InfraStructure (OASIS),* Oct. 2004.

[10] Intel, "*Intel Virtualization Technology Specification for the Intel Itanium Architecture (VT-i)*," Apr. 2005. Revision 2.0.

[11] A. Velte and T. Velte, "Microsoft virtualization with Hyper-V," McGraw-Hill, Inc., 2009.

[12] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. van Doorn, J. L. Griffin, S. Berger, "shype: Secure hypervisor approach to trusted virtualized systems," in Tech. Rep. RC23511, 2005.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," ACM SIGOPS Operating Systems Review, volume 37, pp. 193–206, 2003.

[14] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of Design Automation Conference*, 2001.

[15] F. Triviño, J. L. Sánchez, F. J. Alfaro, and J. Flich, "NoC reconfiguration for CMP virtualization," *in Proceedings of 10th IEEE International Symposium on Network Computing and Applications (NCA)*, Aug. 2011, pp. 219-222.

[16] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanovic, "Chisel: Constructing hardware in a Scala embedded language," in *Proceedings of the 49th Design Automation Conference*, Jun. 2012, pp. 1212–1221.

[17] M. Odersky, L. Spoon, and B. Venners, "Programming in Scala: A Comprehensive Step-by-step Guide," 2nd Ed., Artima, 2010.

[18] J. Flich and J. Duato, "Logic-Based Distributed Routing for NoCs," in Computer Architecture Letters 7.1, pp. 13-16. 2008.

[19] I. Habib, "Virtualization with KVM," *Linux Journal*, Issue 166, 2008.

[20] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kVM: The linux virtual machine monitor," *in Proceedings of Linux Symposium*, Volume 1, pp. 225-230, 2007.

[21] N. Jain and J. Lakshmi, "PriDyn: Framework for Performance specific QoS in Cloud Storage," *in Proceedings of IEEE 7th International Conference on Cloud Computing*, pp. 32-39, Jun. 2014.

[22] M. Dhingra, J. Lakshmi and S. K. Nandy, “Resource Usage Monitoring in Clouds,” *in Proceedings of 13th IEEE/ACM International Conference on Grid Computing (GRID12)*, Sept, 2012.

[23] J. Lakshmi, “System virtualization in the multi-core era, a QoS perspective,” Dissertation, Indian Institute of Science, Bangalore, 2010.

[24] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," *in Proceedings of 16th ACM conference on Computer and communications security*, pp. 199-212, 2009.

[25] M. Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment." *IEEE Transactions on Cloud Computing*, Volume 2, Number 4, pp. 395-408, 2014.

[26] G. I. Apecechea, M. S. Inci, T. Eisenbarth, and B. Sunar. "Fine grain Cross-VM Attacks on Xen and VMware are possible!” IACR Cryptology ePrint Archive, 2014.

[27] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "NIST cloud computing reference architecture," *NIST special publication* 500, no. 2011 (2011): 292.

[28] S. Carlin and K. Curran, "Cloud computing security," International Journal of Ambient Computing and Intelligence, Volume 3, Issue 1, pp. 14-19, 2011.

[29] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," Future Generation Computer Systems, Volume 28, Issue 3, pp. 583-592, Mar. 2012.

[30] R. L. Krutz and R. D. Vines, “*Cloud security: A comprehensive guide to secure cloud computing,*” Wiley Publishing, 2010.

[31] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, Volume 8, Issue 6, pp. 24-31, Nov.-Dec. 2010.

[32] J. C. Roberts II and W. Al-Hamdani, "Who can you trust in the cloud?: a review of security issues within cloud computing," *in Proceedings of 2011 Information Security Curriculum Development Conference*, pp. 15-19, 2011.

[33] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "IaaS cloud architecture: From virtualized datacenters to federated cloud infrastructures," *Computer,* Volume 45, Issue 12 pp. 65-72, 2012.

[34] S. Xu and H. Pollitt-Smith, "A multi-microblaze based SOC system: from SystemC modeling to FPGA prototyping," *in Proceedings of 19th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP08)*, pp. 121-127, 2008.

[35] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. van Doorn, “*A practical guide to trusted computing,*” Pearson Education, 2007.

[36] R. C. Chiang, S. Rajasekaran, N. Zhang, and H. H. Huang, "Swiper: Exploiting virtual machine vulnerability in third-party clouds with competition for I/O resources," *IEEE Transactions on Parallel and Distributed Systems,* Volume 26, Number 6, pp. 1732-1742, 2015.

[37] A. Richter, C. Herber, H. Rauchfuss, T. Wild, and A. Herkersdorf, "Performance isolation exposure in virtualized platforms with PCI passthrough I/O sharing," *in Proceedings of 27th International Conference on Architecture of Computing Systems (ARCS)*, pp. 171-182, 2014.