UF Herbert Wertheim College of Engineering
UNIVERSITY of FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

Department of Electrical & Computer Engineering

Gator Reconfigurable Cloud Computing

Hardware Virtualization Challenges

Smart System Lab

Christophe Bobda

FCCM Workshop 2020

Agenda

❑ GatorReCC Introduction

❑ Resource Provisioning

❑ GatorReCC Architecture

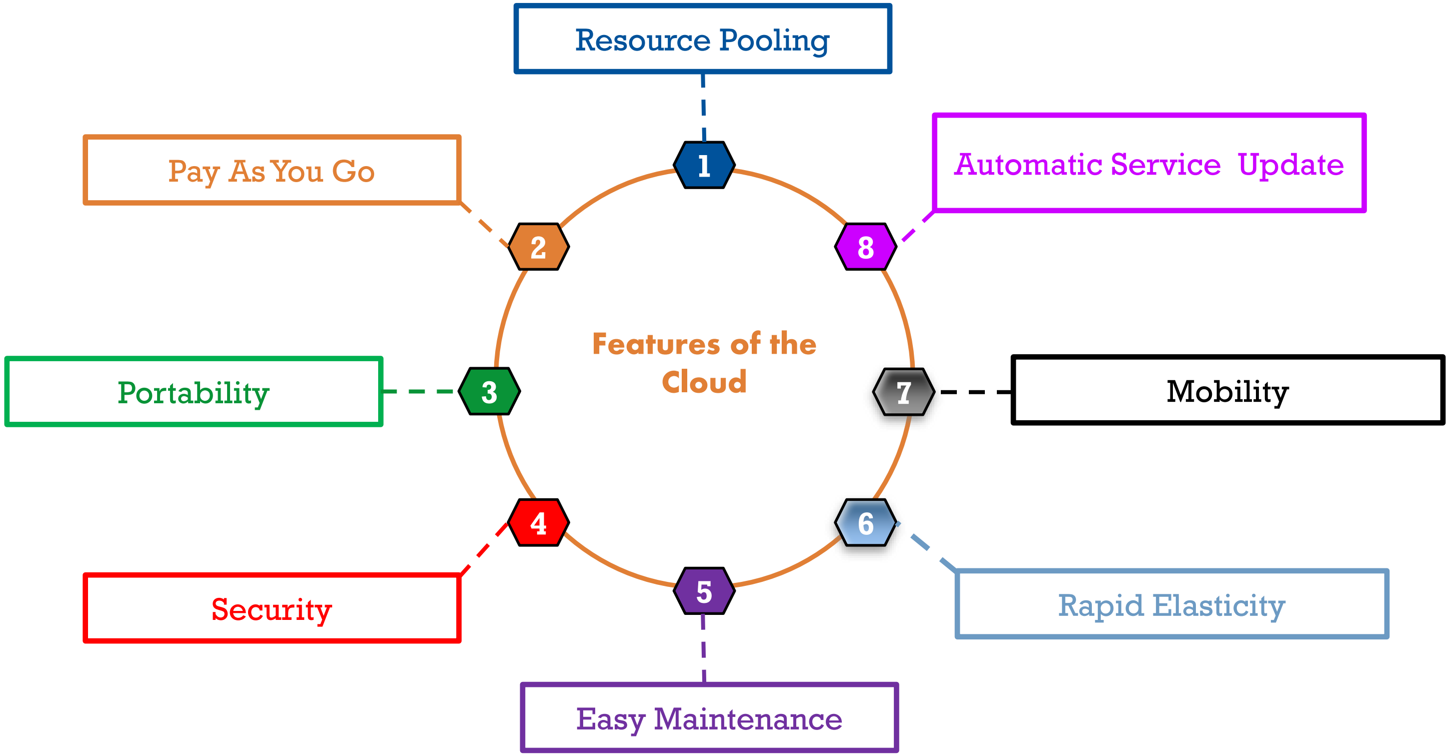
❑ Design Flow

❑ Experimental Results

❑ Challenges

❑ Future Work

FPGA-Accelerated Cloud



Features of the Cloud

1 Resource Pooling

2 Pay As You Go

3 Portability

4 Security

5 Easy Maintenance

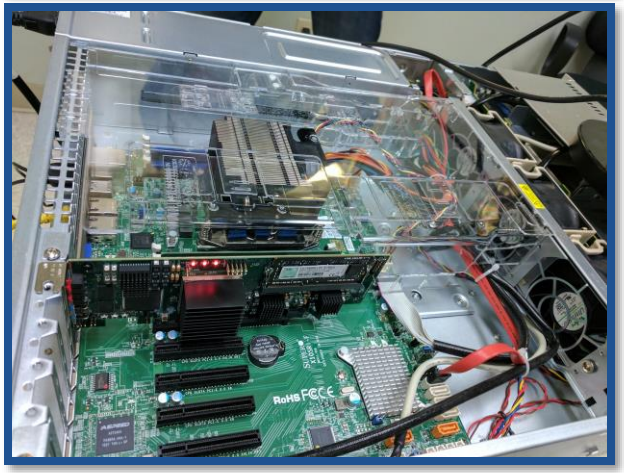
6 Rapid Elasticity

7 Mobility

8 Automatic Service Update

Gator Reconfigurable Cloud

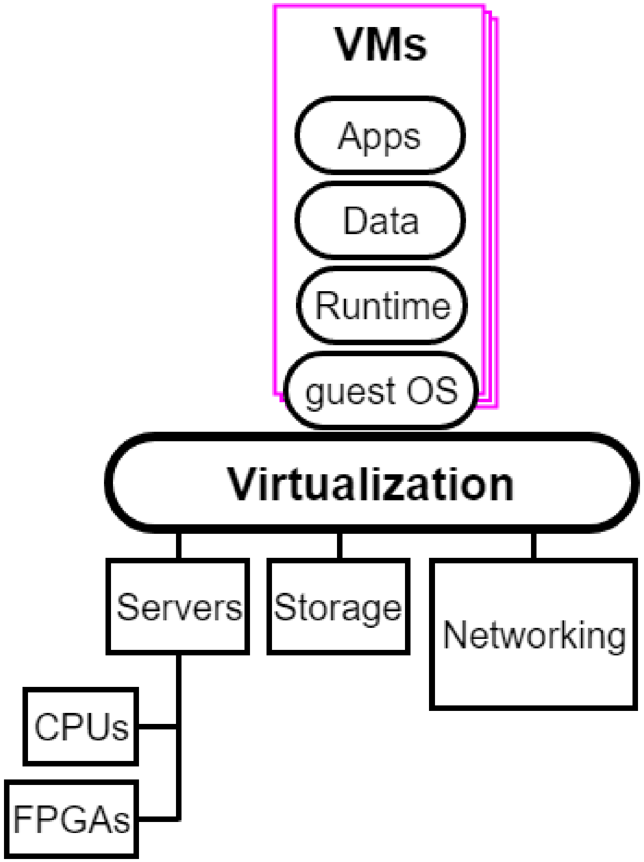
GatorReCC

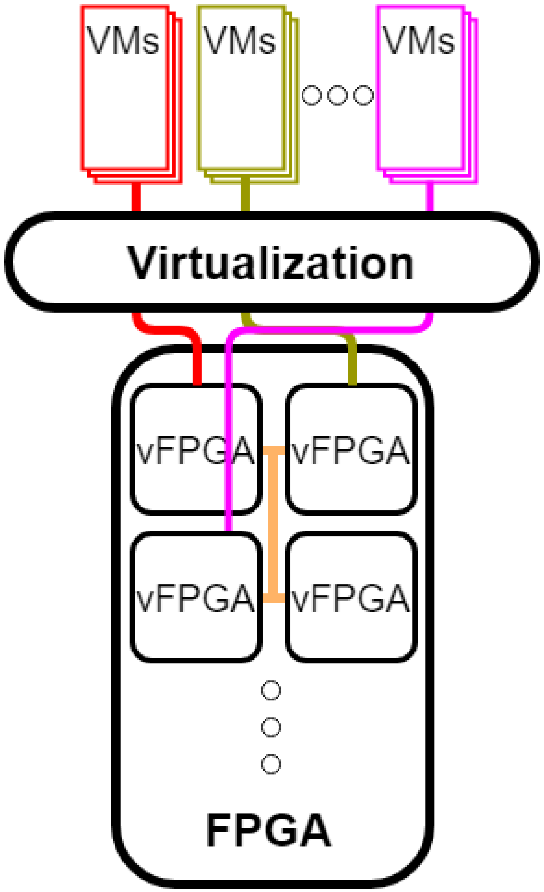


Goals

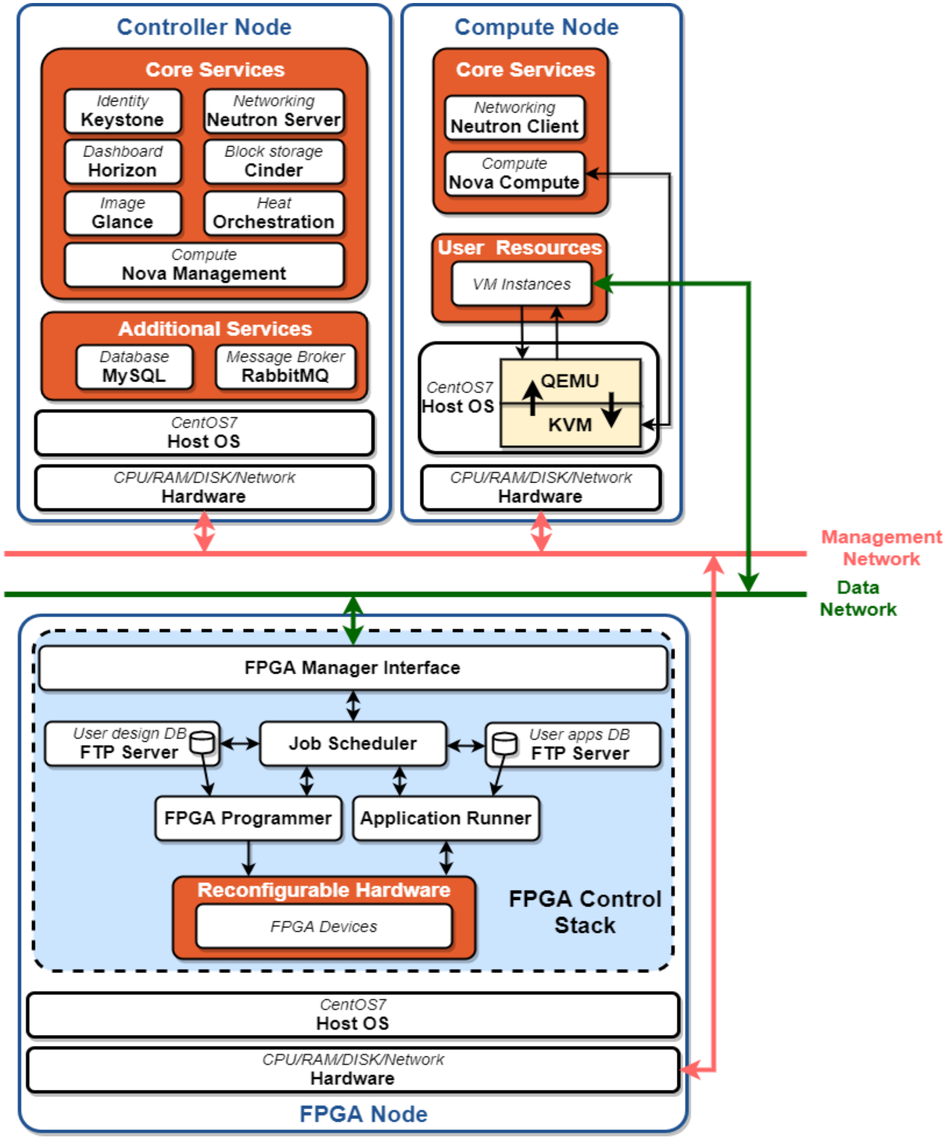
- Provide the capability to researchers to investigate.
  - Infrastructure
  - Middleware
  - Security
  - Applications
- Provide communities and academia the means to install and run their own cloud.
- Service Level
  - Infrastructure (IaaS)
  - Platform (PaaS)
  - Application (SaaS, HaaS, ACCaaSS, …)

Resource Provisioning





Architecture



GatorReCC

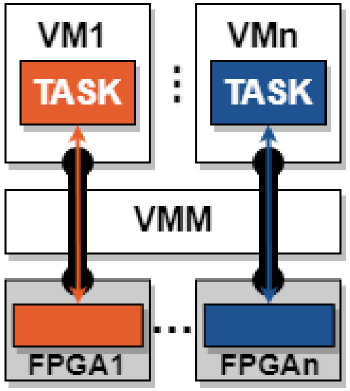
The **Gator Reconfigurable Cloud Computing** infrastructure provides an adaptable cloud architecture for performance and security threat exploration. It essentially features:

- ❑ An **OpenStack** Management stack.
- ❑ FPGAs deployed across multiple FPGA nodes.

Architecture

We aim to explore performance that can be achieved in multi-tenant FPGA deployments in the cloud.

Single-tenant FPGAs

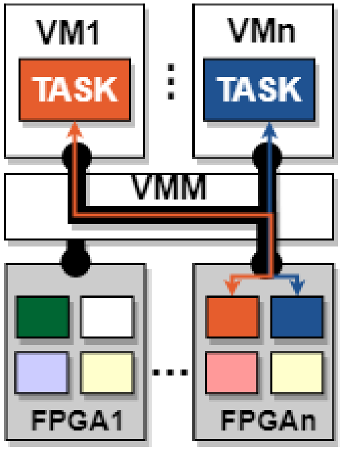


Each FPGA is entirely allocated to a VM over time.

Problem

Resource waste and low FPGA utilization as user designs do not always use 100% of the FPGA.

Multi-tenant FPGAs

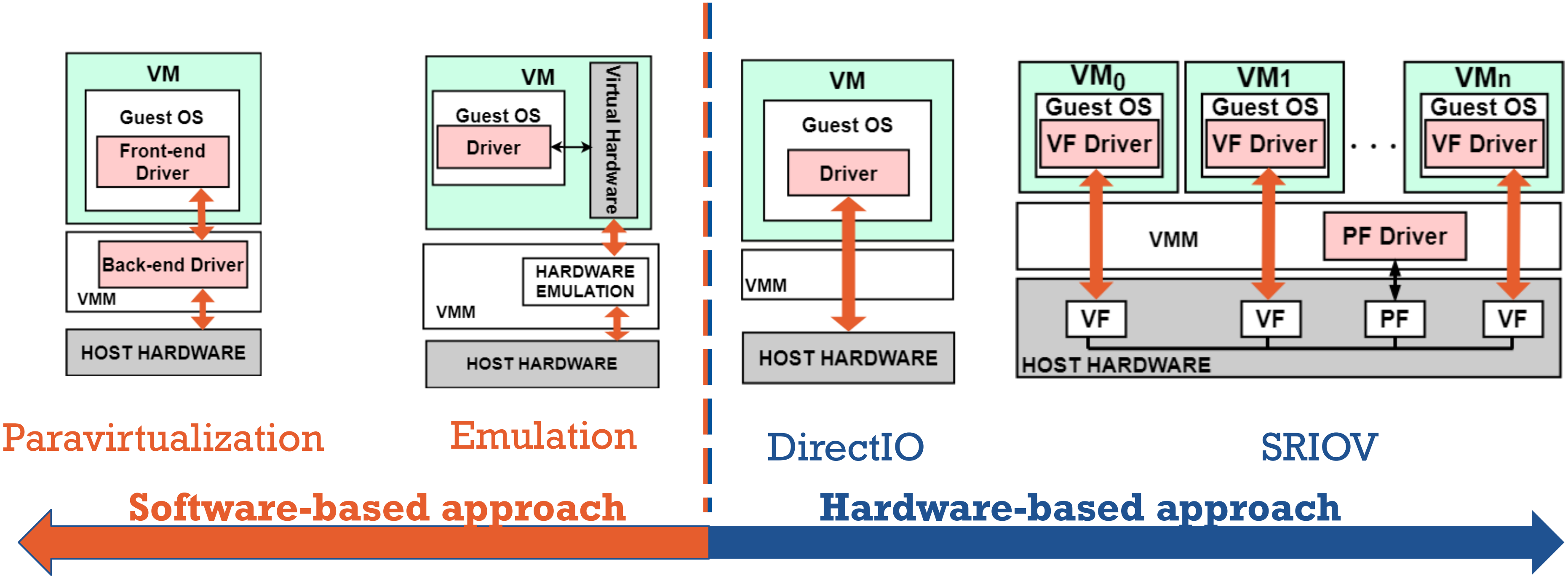


FPGA resources are time and space-shared between user workloads.

Advantage

Improved utilization of FPGA resources across cloud workloads.

Virtualization Techniques



Emulation: No modification of guest OS, but high overhead penalty.

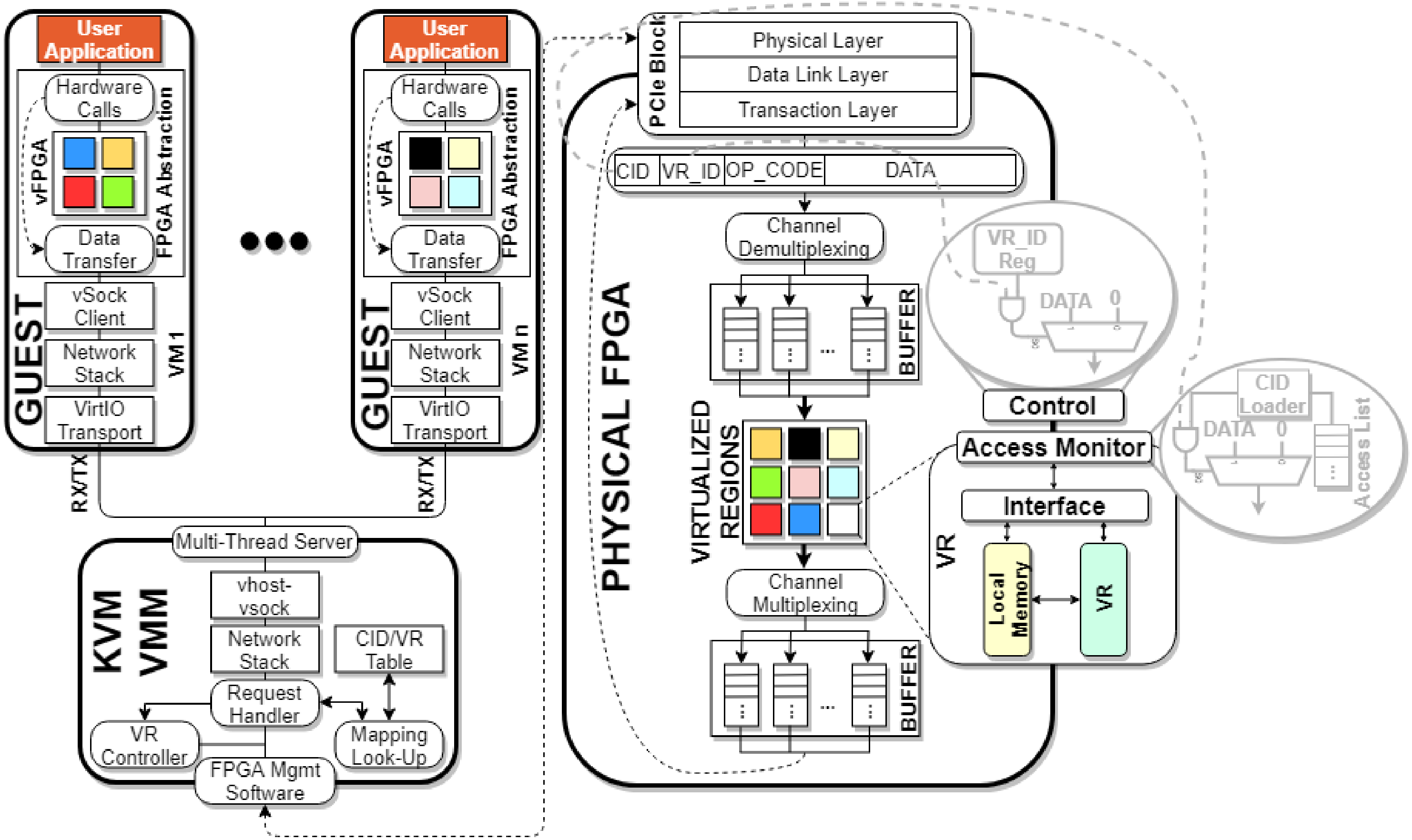
Paravirtualization: drivers in the guest connect to backend drivers in the VMM.

DirectIO: native IO performance, but no sharing.

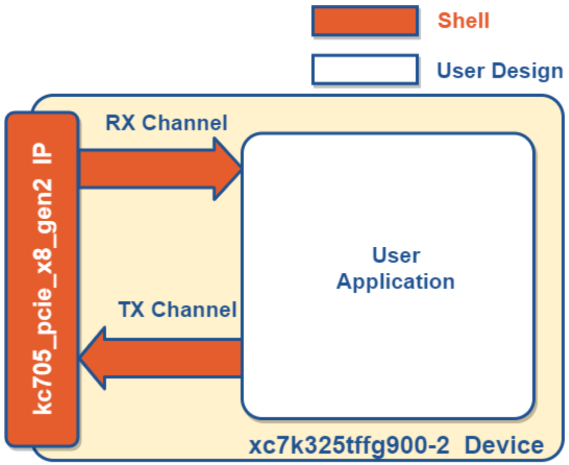
SRIOV/MRIOV: hardware sharing enabled but no run-time remapping support

Architecture

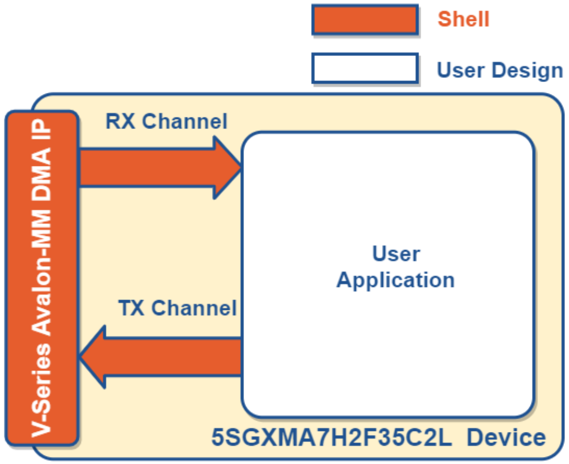
The **GatorReCC** infrastructure cloud infrastructure leverages **VirtIO** to intercept hardware calls and efficiently access FPGA accelerators.



Architecture



Kintex7 Design Block



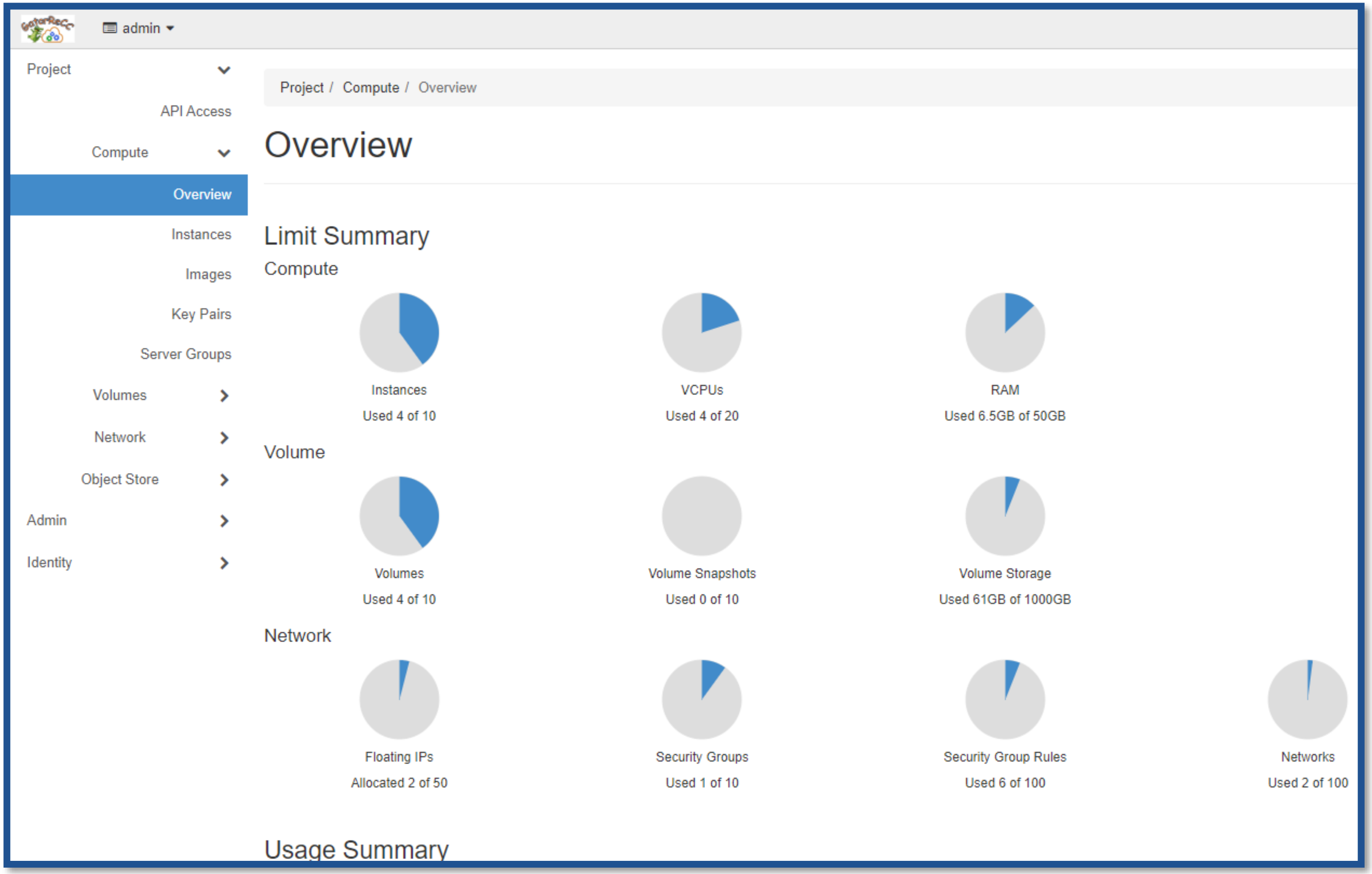
Stratix V Design Block

GatorReCC

The current deployment includes a single FPGA node provisioning a **Xilinx Kintex7 FPGA** and an **Intel Stratix V device**.

- Users are responsible from implementation hardware accelerators.
- Users also provides software applications that write and read data to their accelerators.

Architecture - Dashboard



Overview

Limit Summary

Compute

Instances
Used 4 of 10

VCPUs
Used 4 of 20

RAM
Used 6.5GB of 50GB

Volume

Volumes
Used 4 of 10

Volume Snapshots
Used 0 of 10

Volume Storage
Used 61GB of 1000GB

Network

Floating IPs
Allocated 2 of 50

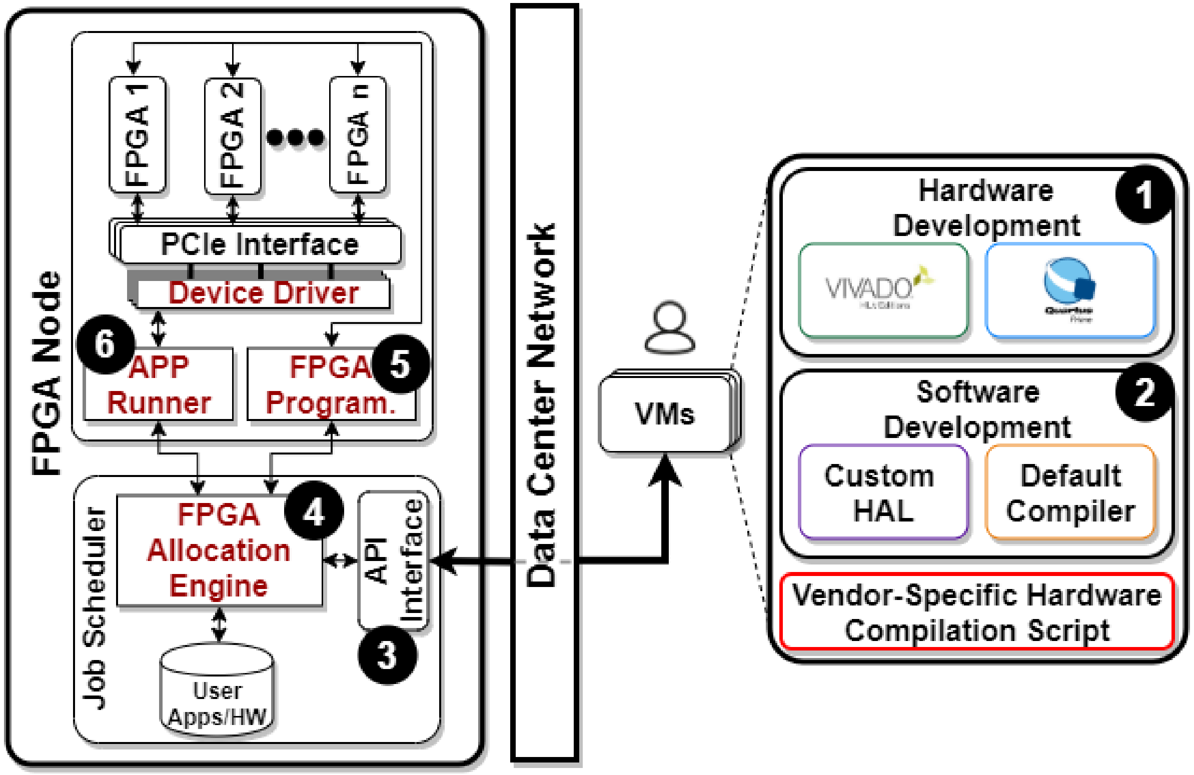
Security Groups
Used 1 of 10

Security Group Rules
Used 6 of 100

Networks
Used 2 of 100

Usage Summary

Design Flow



1. Users specify hardware kernels with HDL/HLS

2. Users specify software for reading/writing to hardware accelerators.

3. Users submit jobs (HW & SW) to the cloud infrastructure.

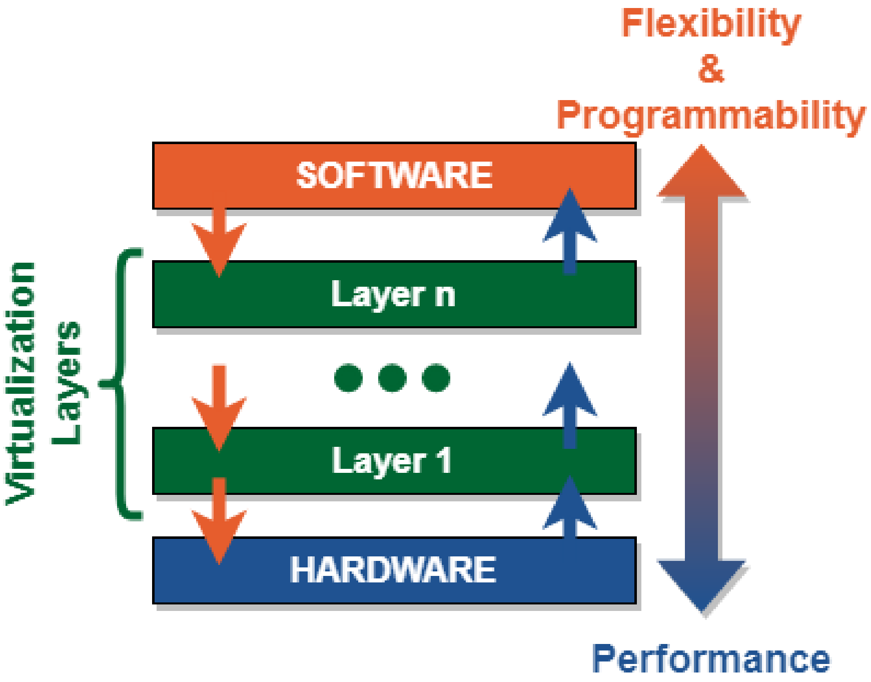
4. The **Job Scheduler** places VM jobs into a queue

5. The **FPGA Programmer** programs user kernels

6. The **App Runner** runs user software.

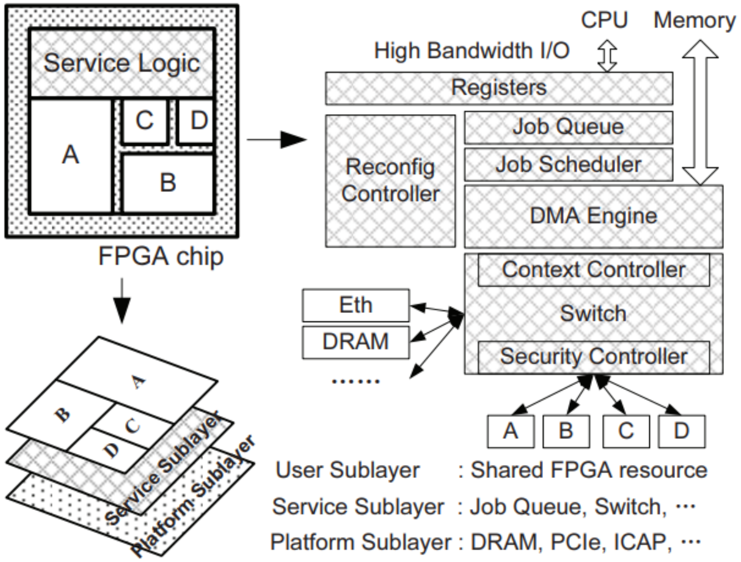
Virtualization Challenges

❑ How to achieve bare-metal-level performance with virtualization ?



FPGA VIRTUALIZATION IN THE CLOUD

- Spatio-Temporal Computing Fixed/Dynamic



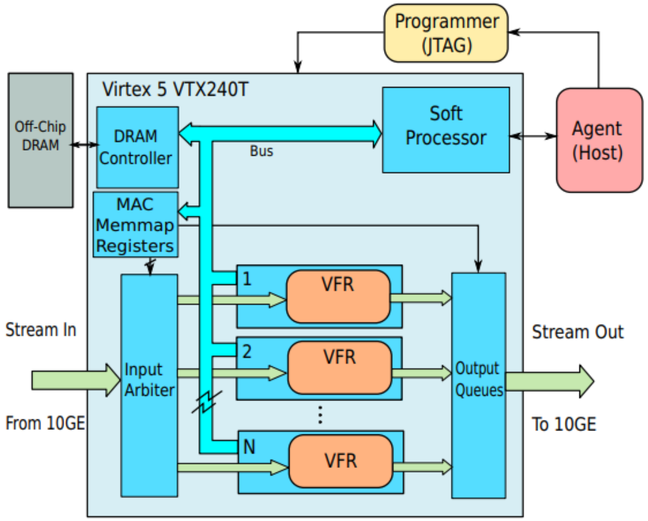
FPGA Shell and Accelerator Slots [1]

The FPGA is divided into an accelerator pool (A, B, C, and D in Fig1). Each accelerator slot can be programmed using partial reconfiguration.

The cloud provider then pre-builds a set of accelerators or compile user designs and make them available to cloud users.

FPGA VIRTUALIZATION IN THE CLOUD

- Spatio-Temporal Computing Fixed/Dynamic



System Level FPGA Diagram [2]

Proposes Virtualized FPGA Resources (VFR) surrounded by static logic (see Fig2).

The input Arbiter directs inputs to the right VFR based on MAC addresses stored in the data packet. Each VFR can then be assigned to a single VM.

The soft processor allows setting configuration values such as the MAC address of new VFRs, and freezing interfaces of VFR before it gets a new bitfile through partial reconfiguration.

FPGA VIRTUALIZATION IN THE CLOUD

❑ Existing approaches divide the FPGA into regions that are provisioned to virtual instances

Drawbacks

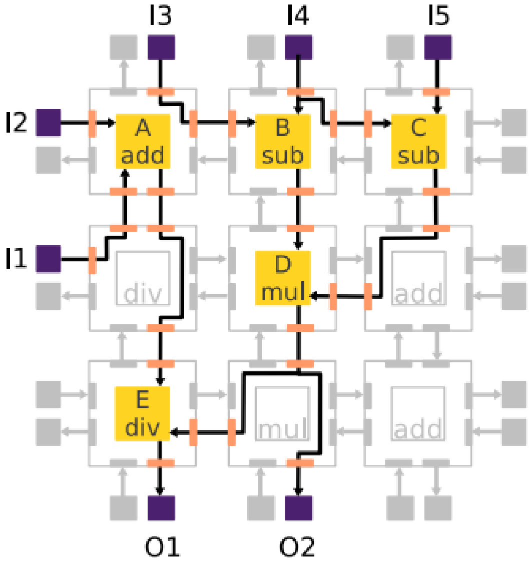
i. No communication enabled between virtual FPGA Regions
ii. No possibility to dynamically allocate additional FPGA resources
iii. Lack of security mechanism enforcing isolation between hardware tasks

FPGA OVERLAYS

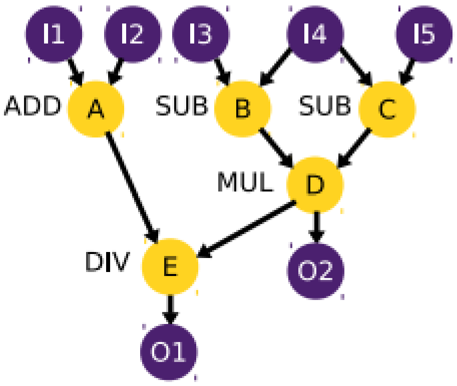
Some Overlays Facilitate FPGA Adoption in Software Stacks

[7] presents an architecture for accelerating the execution of data flow graphs. Each cell of the overlay contains a functional unit (FU) implementing an operation from a node of the DFG.

Processing take place in FUs embedded within cells.



Mapping of the DFG



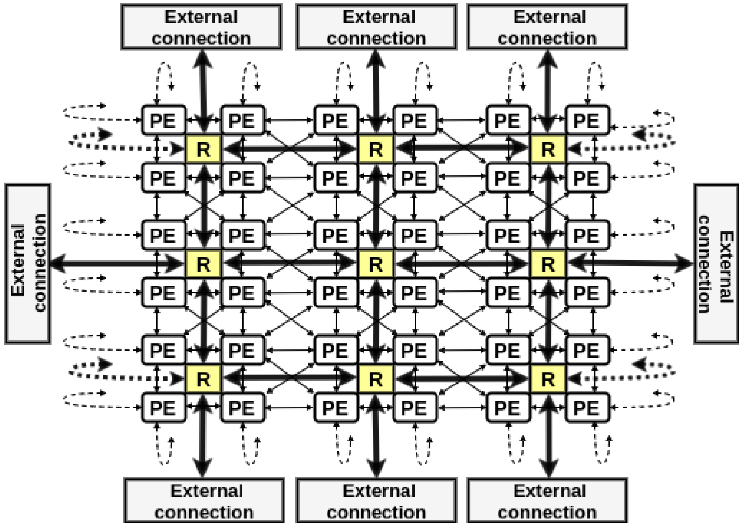
Example of DFG

FPGA OVERLAYS

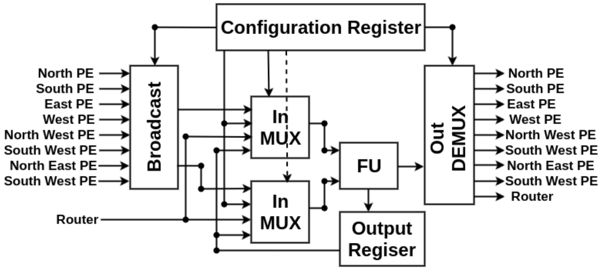
Goal: Facilitate FPGA integration in Software Stacks

The FLexiTASK architecture [3] features a flexible network-on-chip architecture facilitating multi-threading and communication between tasks.

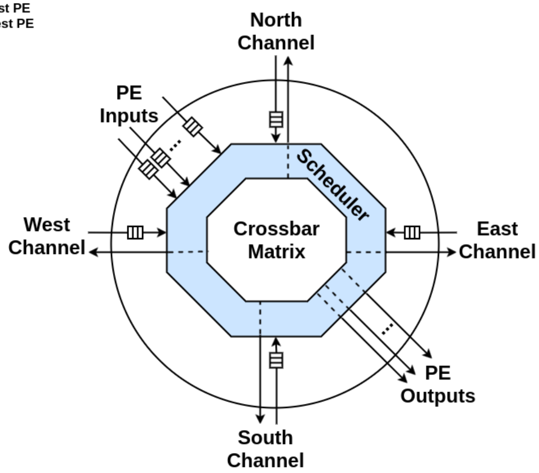
Processing Element are immersed in a Torus interconnect allowing flexible and efficient communication between close and distant cores.



FLexiTASK Architecture



Processing Element



Router Architecture

FPGA OVERLAYS

Goal: Facilitate FPGA integration in Software Stacks

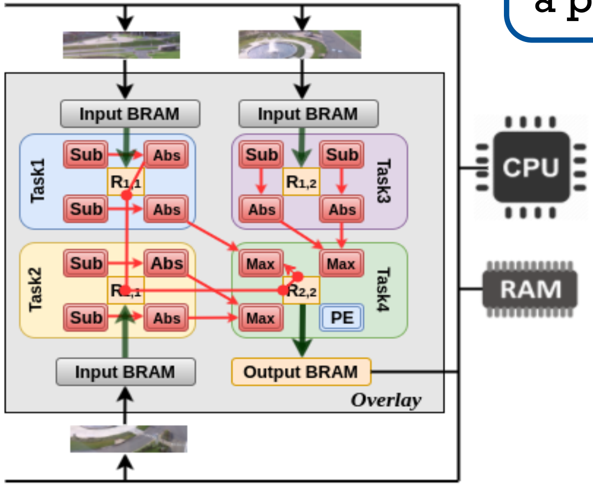
The FLexiTASK architecture [3] features a flexible network-on-chip architecture facilitating multi-threading and communication between tasks.

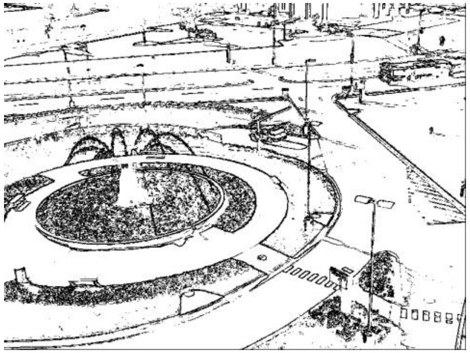
Robert Cross Filter

Intensity pixel(x,y)= max{ |M+ * pixel(x,y)|, |M- * pixel(x,y)|}

With M+=$\begin{pmatrix} 0 & +1 \\ -1 & 0 \end{pmatrix}$ and M-=$\begin{pmatrix} +1 & 0 \\ 0 & -1 \end{pmatrix}$

The initial image is divided into 3 portions fed in parallel into 3 independent tasks (Task1, Task2, Task3). The last task (Task4) computes maximums to decide if a pixel is an edge or not.

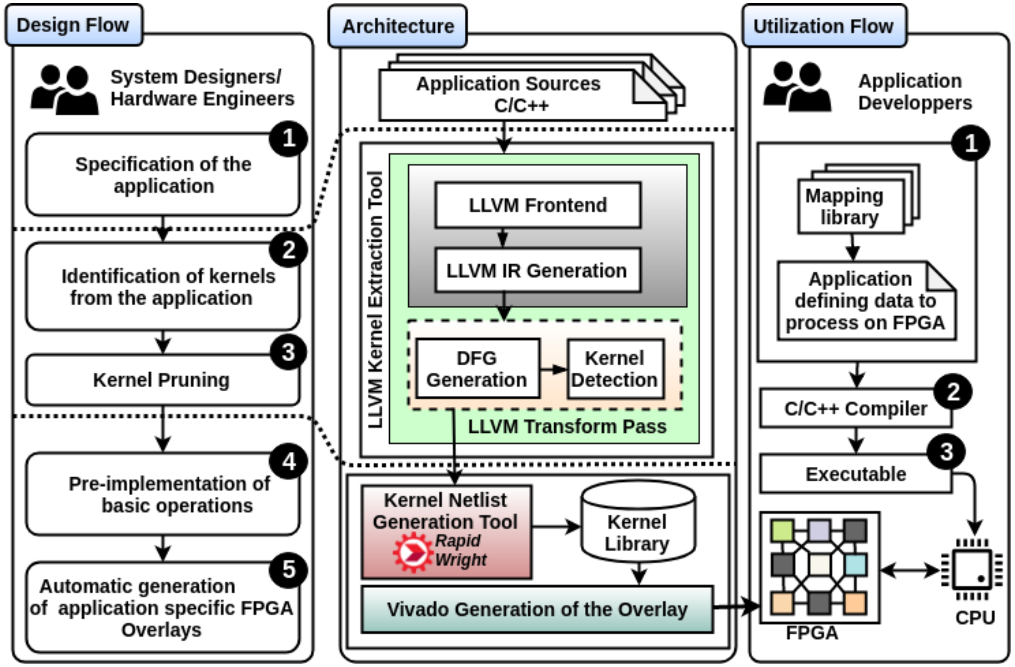




Example of use FLexiTASK for edge detection with the Robert Cross Filter

FPGA OVERLAY AUTOMATIC GENERATION

An automatic flow for application-specific FPGA accelerator generation


Design Flow
System Designers/ Hardware Engineers
1
Specification of the application
2
Identification of kernels from the application
3
Kernel Pruning
4
Pre-implementation of basic operations
5
Automatic generation of application specific FPGA Overlays
Architecture
Application Sources C/C++
LLVM Kernel Extraction Tool
LLVM Frontend
LLVM IR Generation
DFG Generation
Kernel Detection
LLVM Transform Pass
Kernel Netlist Generation Tool
Rapid Wright
Kernel Library
Vivado Generation of the Overlay
Utilization Flow
Application Developpers
1
Mapping library
Application defining data to process on FPGA
2
C/C++ Compiler
3
Executable
FPGA
CPU

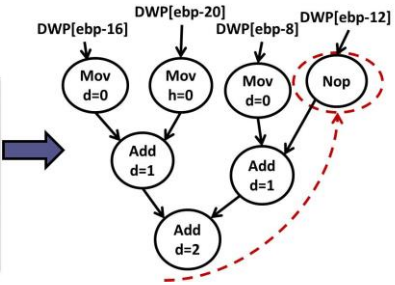
Application-Specific FPGA Overlay Flow

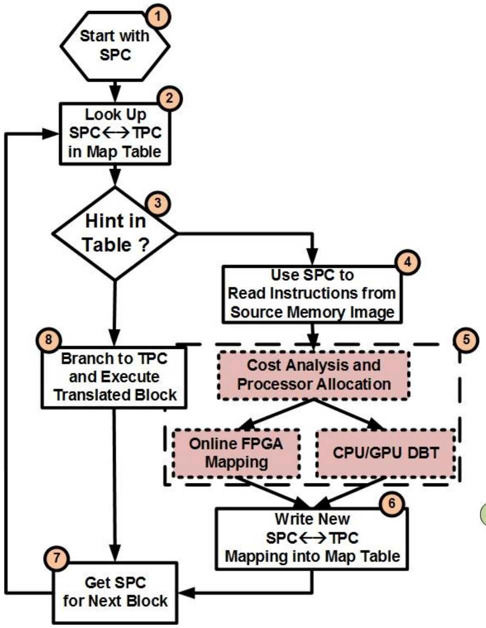
Applications are automatically parsed to extract compute intensive regions using an LLVM Transform Pass. Kernel Identified are transformed into a netlist with RapidWright, and dedicated accelerators are then generated.

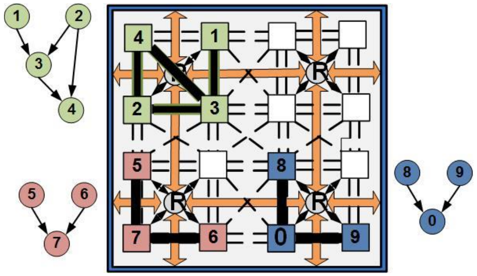
User can just use a custom library providing I/O access to accelerators to leverage FPGA Acceleration

Code Migration – Online/Offline

```
mov eax, DWP[ebp-16]
mov edx, DWP[ebp-20]
add  edx, eax
mov ebx, DWP[ebp-8]
add  ebx, DWP[ebp-12]
add  edx, ebx
```

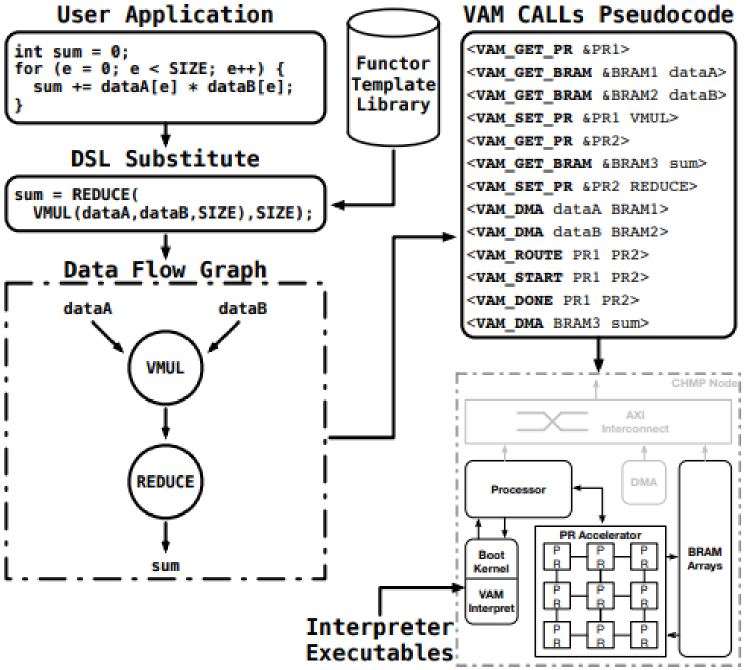



1 Start with SPC
2 Look Up SPC↔TPC in Map Table
3 Hint in Table ?
4 Use SPC to Read Instructions from Source Memory Image
5 Cost Analysis and Processor Allocation
Online FPGA Mapping
CPU/GPU DBT
6 Write New SPC↔TPC Mapping into Map Table
7 Get SPC for Next Block
8 Branch to TPC and Execute Translated Block



Application-Specific FPGA OVERLAY

An automatic flow for application-specific FPGA accelerator generation



```
int sum = 0;
for (e = 0; e < SIZE; e++) {
  sum += dataA[e] * dataB[e];
}
```

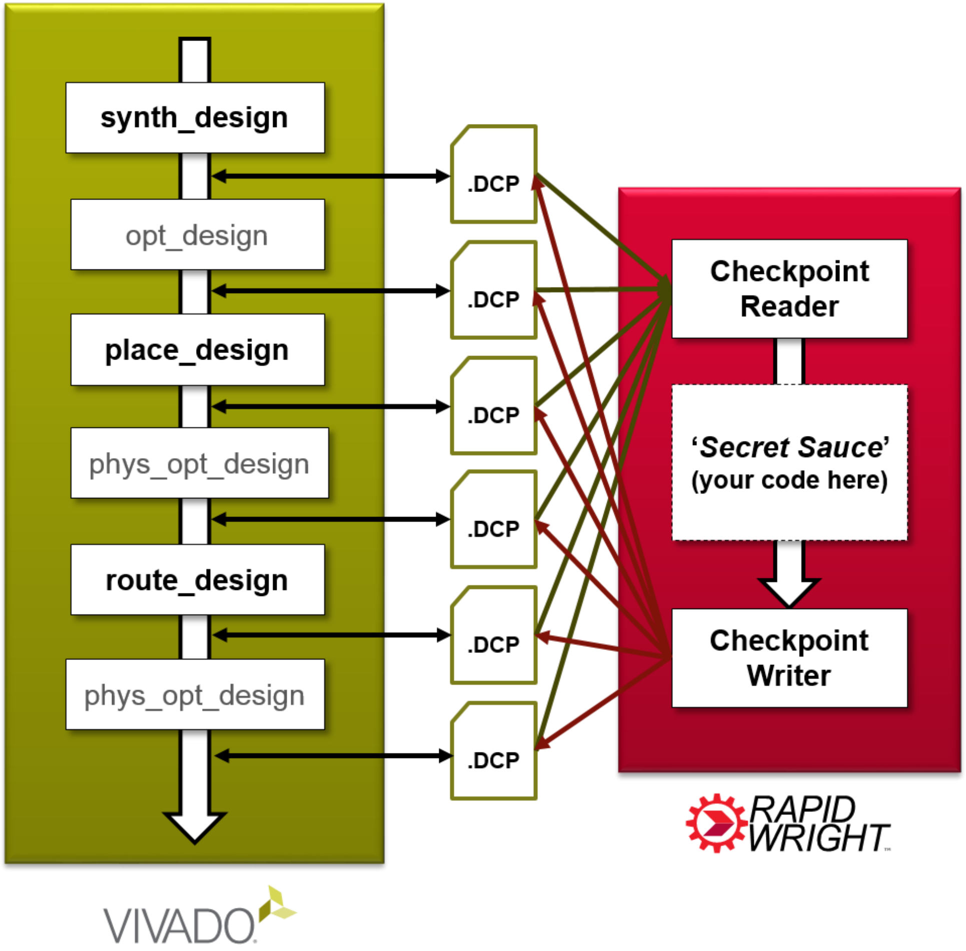
```
sum = REDUCE(
  VMUL(dataA,dataB,SIZE),SIZE);
```

```
<VAM_GET_PR &PR1>
<VAM_GET_BRAM &BRAM1 dataA>
<VAM_GET_BRAM &BRAM2 dataB>
<VAM_SET_PR &PR1 VMUL>
<VAM_GET_PR &PR2>
<VAM_GET_BRAM &BRAM3 sum>
<VAM_SET_PR &PR2 REDUCE>
<VAM_DMA dataA BRAM1>
<VAM_DMA dataB BRAM2>
<VAM_ROUTE PR1 PR2>
<VAM_START PR1 PR2>
<VAM_DONE PR1 PR2>
<VAM_DMA BRAM3 sum>
```

Application-Specific FPGA Overlay Flow

The approach pre-synthesizes some functions and create a domain-specific language (DSL). Each time a function from the DSL is used, it is replaced by a hardware call to the FPGA.

Code Migration – Online/Offline

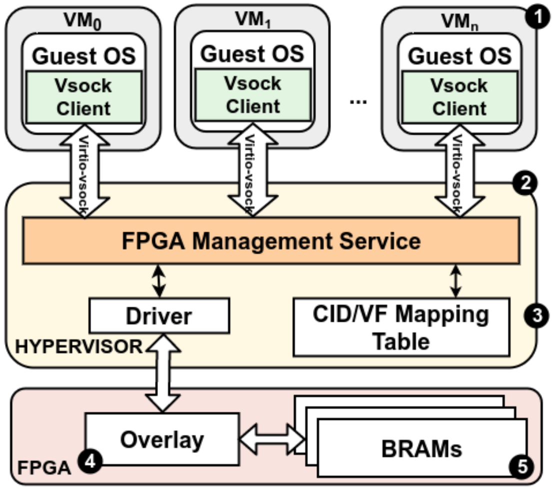


UF Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

FPGA VIRTUALIZATION IN THE CLOUD

- The FPGAVirt Framework [4] implements an FPGA management service that assign a unique context ID (CID) to VMs for keeping track of VF to VM binding



1 A VM requests FPGA resouces

2 The hypervisor consults the mapping table, looking for available spots on the FPGA

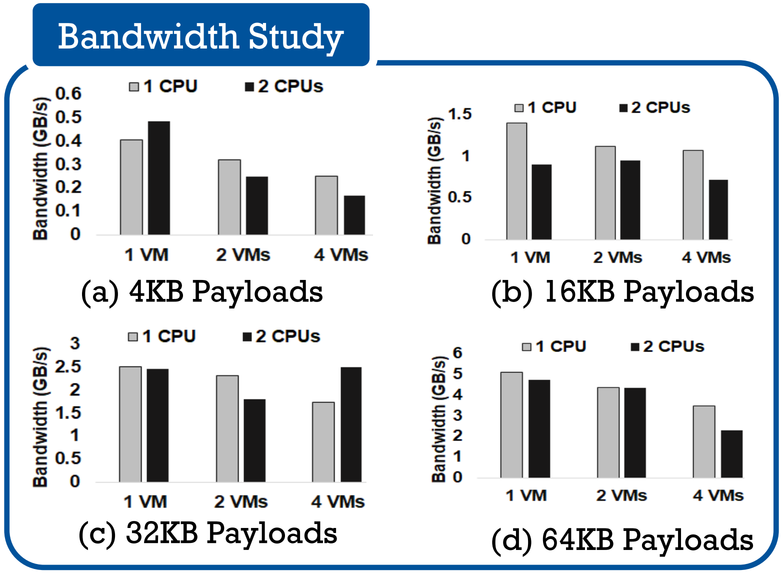
3 If no VF is available, the hypervisor contacts other servers looking for placement options

4 If available, the hypervisor assigns the VF to the VM and edits the mapping table to record it

5 A memory space is allocated to the VF for storing data

FPGAVirt Architecture [4]

For high bandwidth, Virtio is used to open communication channel between VMs and their FPGA regions



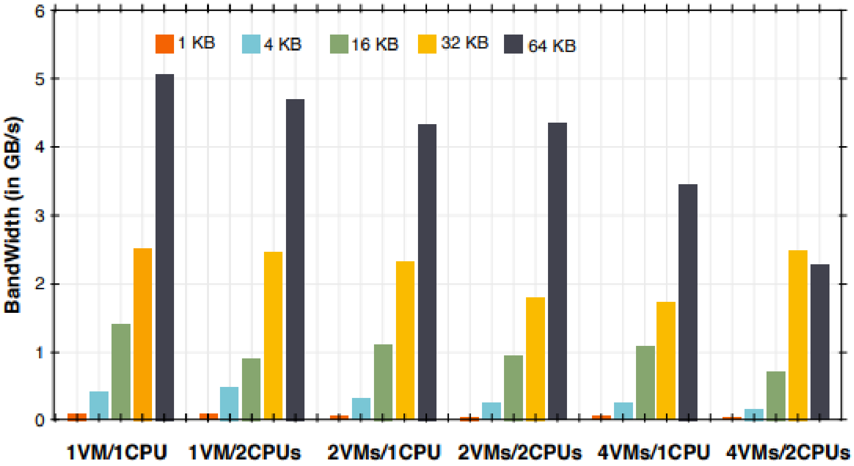
(a) 4KB Payloads

(b) 16KB Payloads

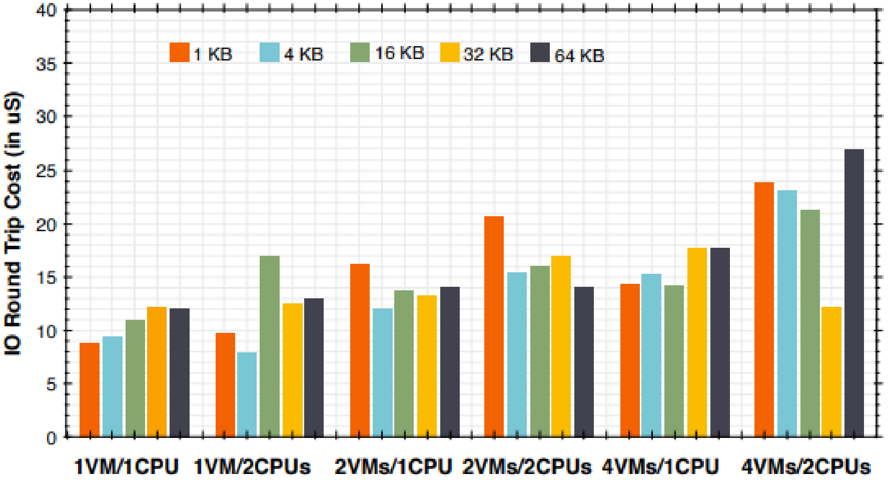
(c) 32KB Payloads

(d) 64KB Payloads

Experimental Results



VM-Accelerators IO Bandwidth



VM-Accelerators IO Roundtrip Costs