# FPGA VIRTUALIZATION, MIGRATION AND RESOURCE ELASTICITY – RELATED WORK

Presented by Harsh Khetawat

03/28/2019

The University of Manchester

# Resource Elastic Virtualization for FPGAs using OpenCL

Anuj Vaishnav
Co-Authors: K. D. Pham, D. Koch & J. Garside
School of Computer Science
The University of Manchester

# FPGA Deployment

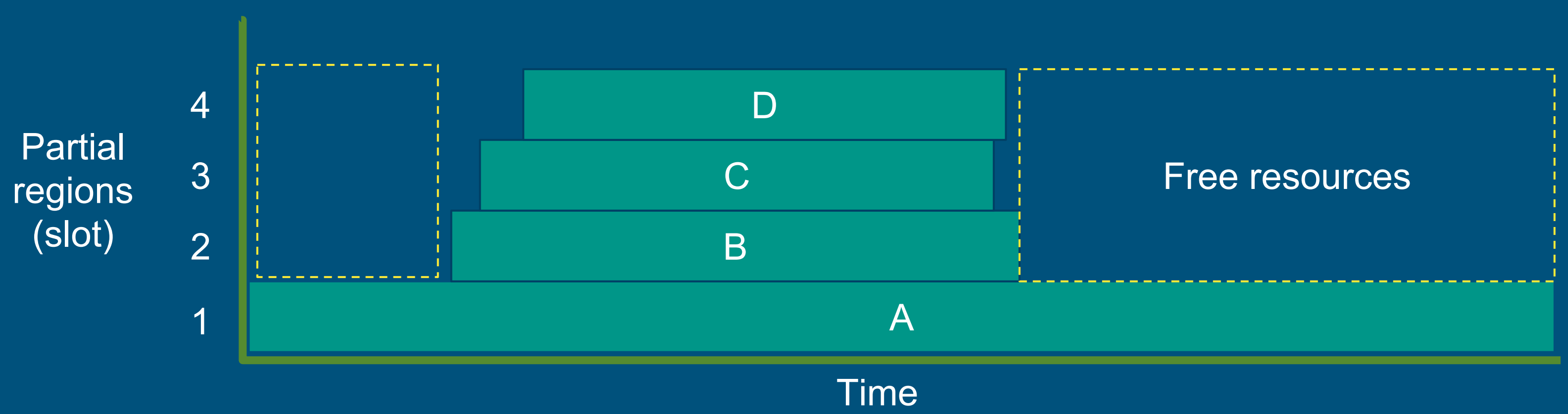
- Cloud Providers: Amazon, Baidu, IBM, Alibaba, Huawei, Nimbix...
- Microsoft: Behind the scene deployment in data centers across 15 countries and 5 continents*
- Backbone for ExaScale computing projects (e.g. ECOSCALE, ExaNeSt, EuroEXA)

*A. M. Caulfield et al., "A cloud-scale acceleration architecture," *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Taipei, 2016, pp. 1-13.

# Current FPGA world

Saving & Restoring state is costly

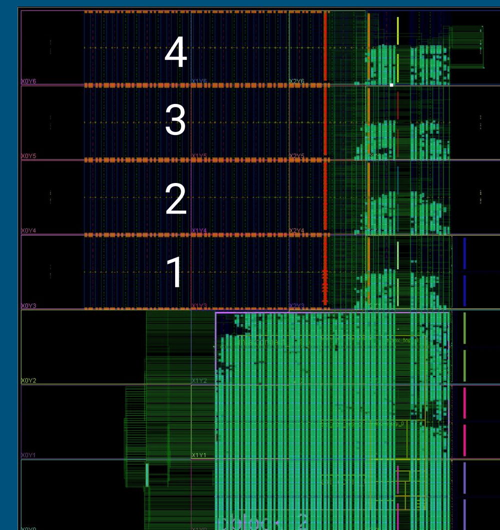
- Run-to-completion model => No Context switch (in most cases)
- No migration of workload without restarting or too many constraints
- Single user application per FPGA (in most cases)
- No scalability of performance with more FPGA resources without re-design



The University of Manchester

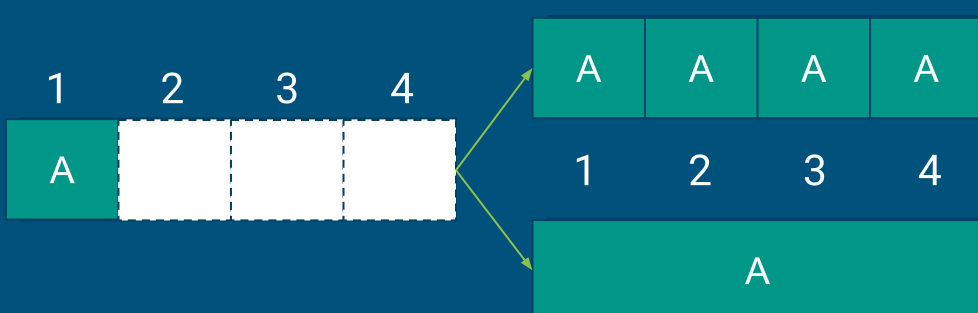
# Resource Elasticity

Definition: Ability of kernels to **grow** and **shrink** its resource footprint transparently from the user.




Grow and shrink how?

- Module replication
- Module replacement




Potentially linear performance gain

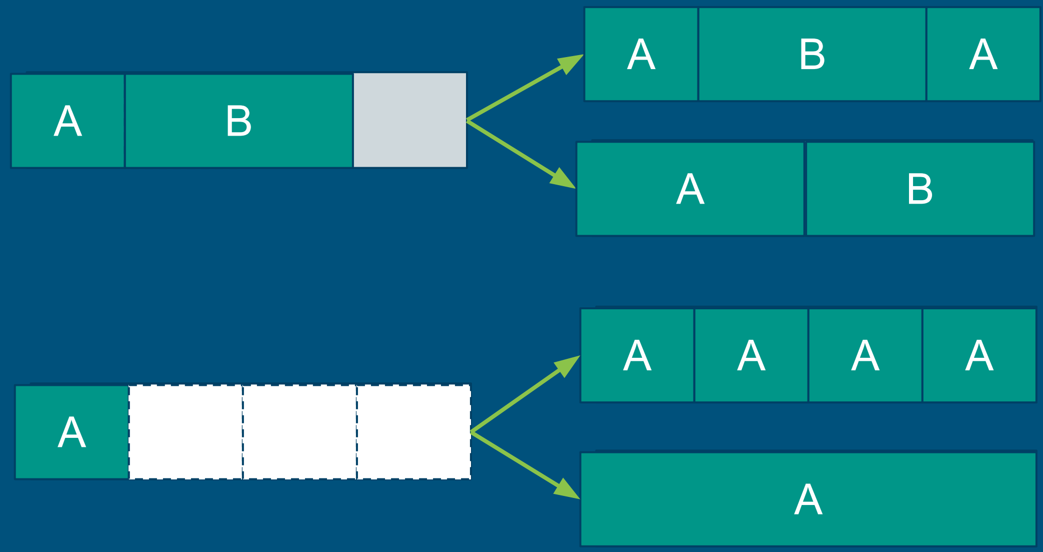
Potentially super-linear performance gain

# Resource Elasticity Trade-offs

- Multiple instances vs Different sized module
- Run to completion vs Changing module layout
- Collocated change vs Distributed change

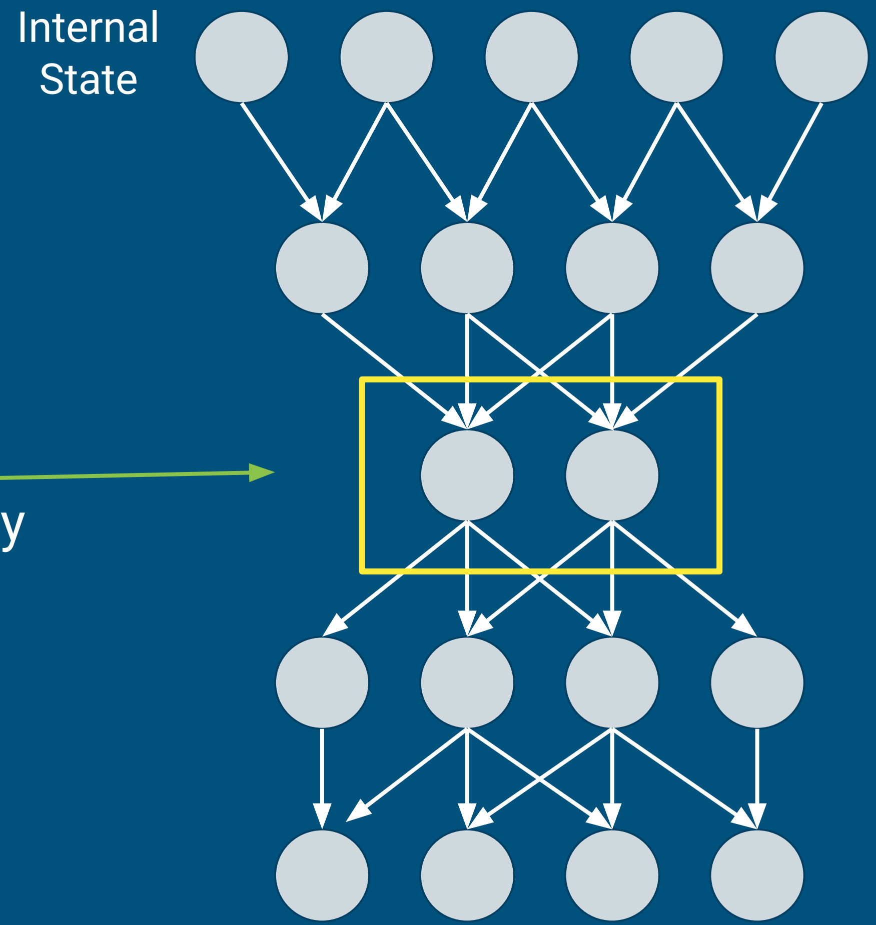
Is it worth it, given the overhead?

Deal with fragmentation



# Context Switching

- Preemptive scheduling techniques:
  - Configuration read back
  - Scan chains
- Cooperative scheduling approach: Only perform context switch at consistency points

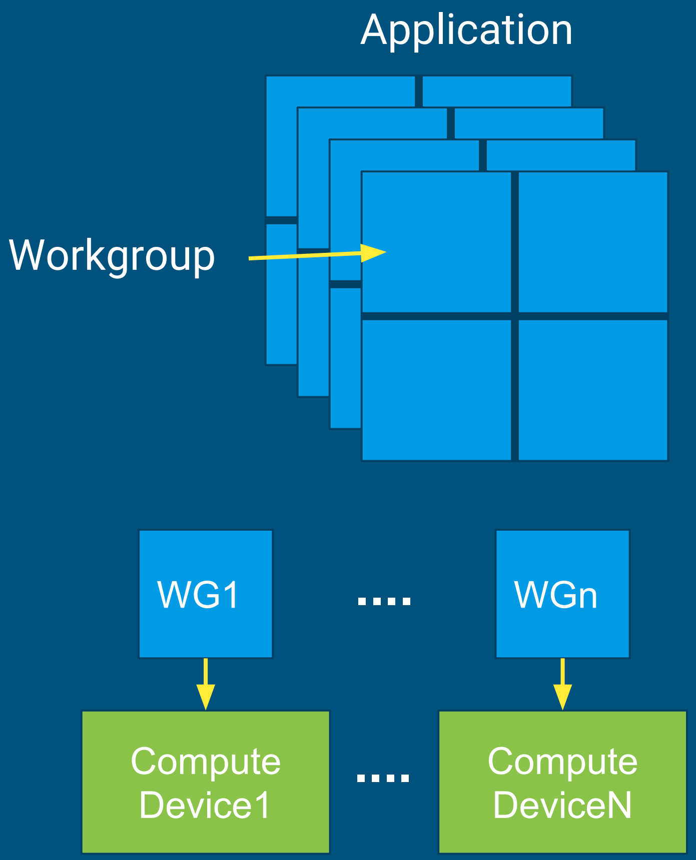


# OpenCL

- Designed for heterogeneous systems
- Work-group is made up of work-items (lightweight threads)
- Inside work-group, synchronization primitives can be used
- **No execution order or synchronization across work-groups**

Allows Context Switching: No read and write of internal state required

```
for (i = 0; i < 16; i++)
  for (j = 0; j < 16; j++)
    for (k = 0; k < 16; k++)
```



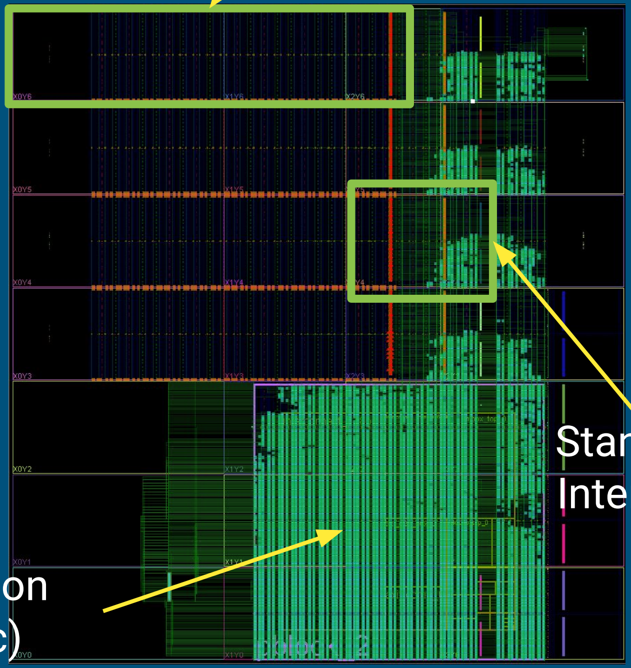
# Base infrastructure

Main features:

Flexibility of scheduling

- Multiple partial regions (also called slots) side by side
- Vivado HLS to generate OpenCL accelerators
- Placed & routed as relocatable accelerators

Perform context switch at the end of work-group (bunch of threads)




Zynq UltraScale+

# Base infrastructure

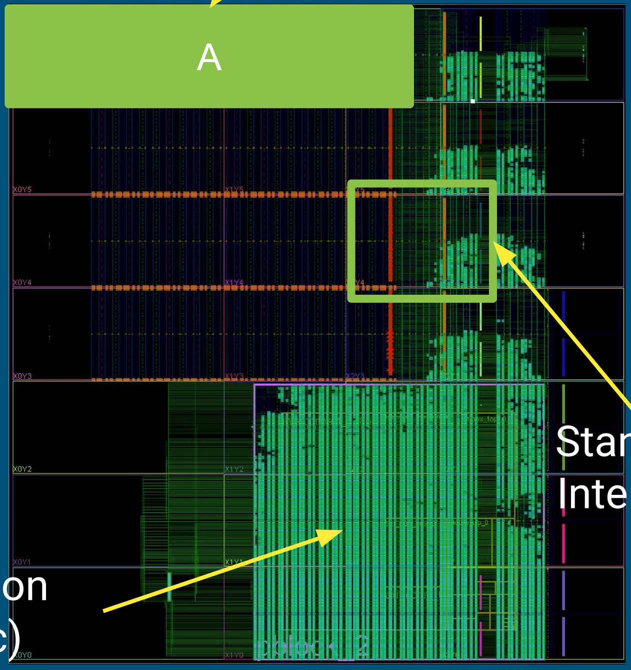
Main features:

Flexibility of scheduling

- Multiple partial regions (also called slots) side by side
- Vivado HLS to generate OpenCL accelerators
- Placed & routed as relocatable accelerators

Perform context switch at the end of work-group (bunch of threads)

Slot: where modules are placed




Standard Interface

Static region (OS logic)

Zynq UltraScale+

# Base infrastructure

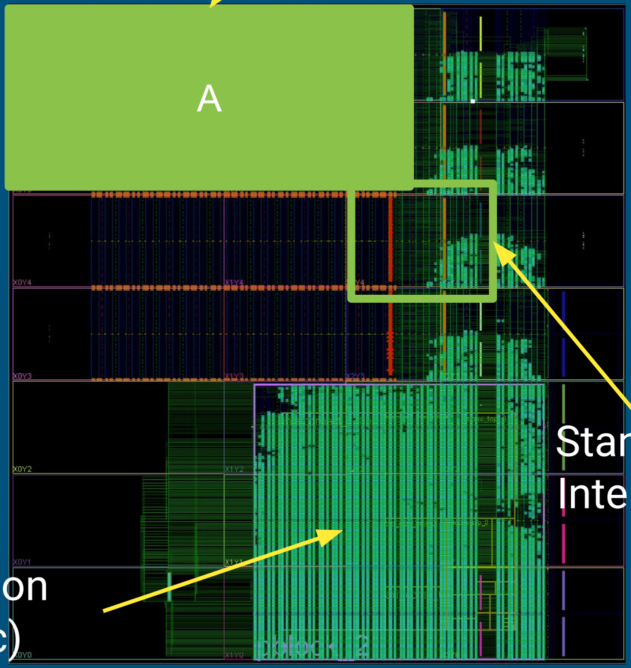
Main features:

Flexibility of scheduling

- Multiple partial regions (also called slots) side by side
- Vivado HLS to generate OpenCL accelerators
- Placed & routed as relocatable accelerators

Perform context switch at the end of work-group (bunch of threads)

Slot: where modules are placed

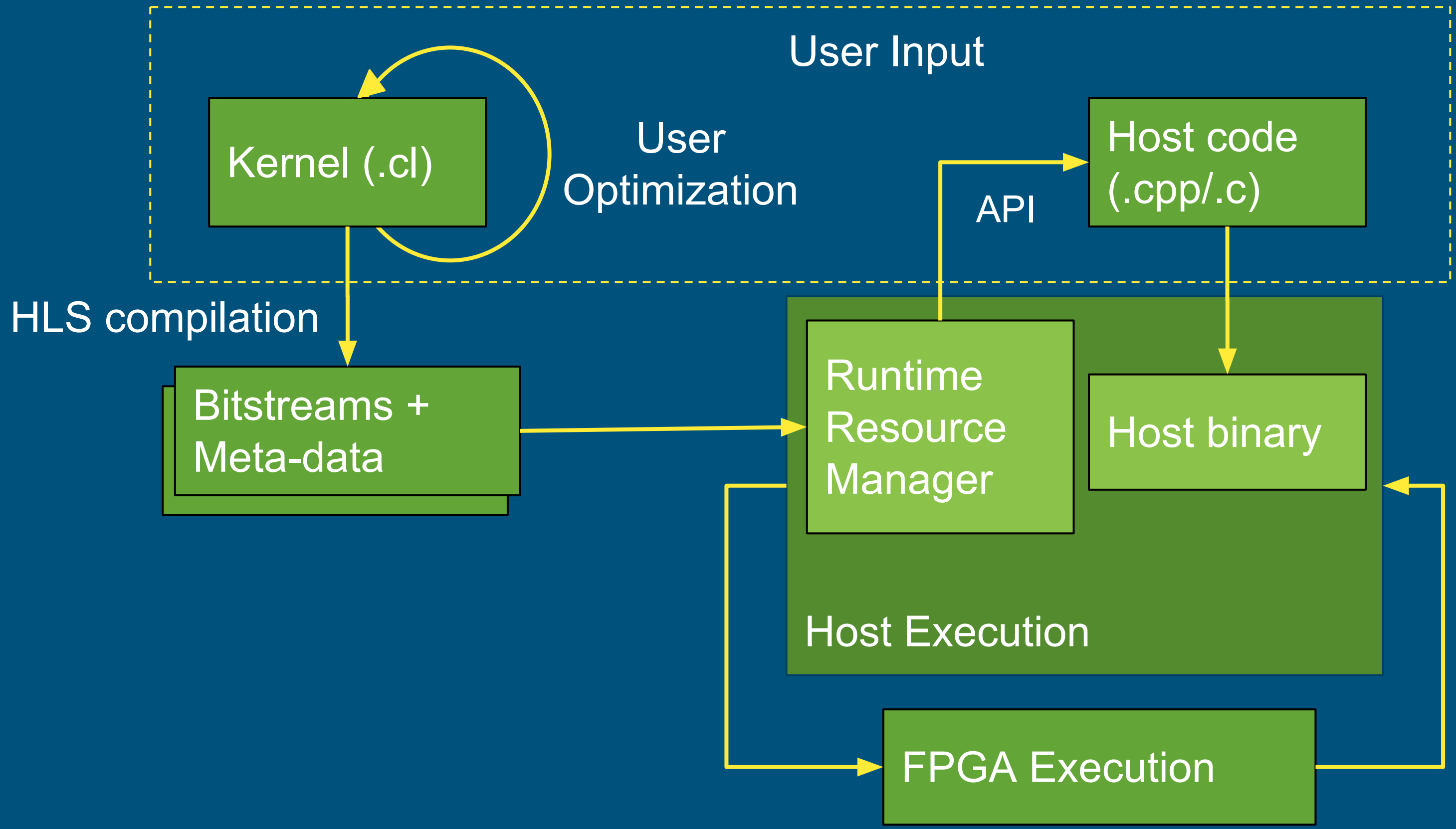



Standard Interface

Static region (OS logic)

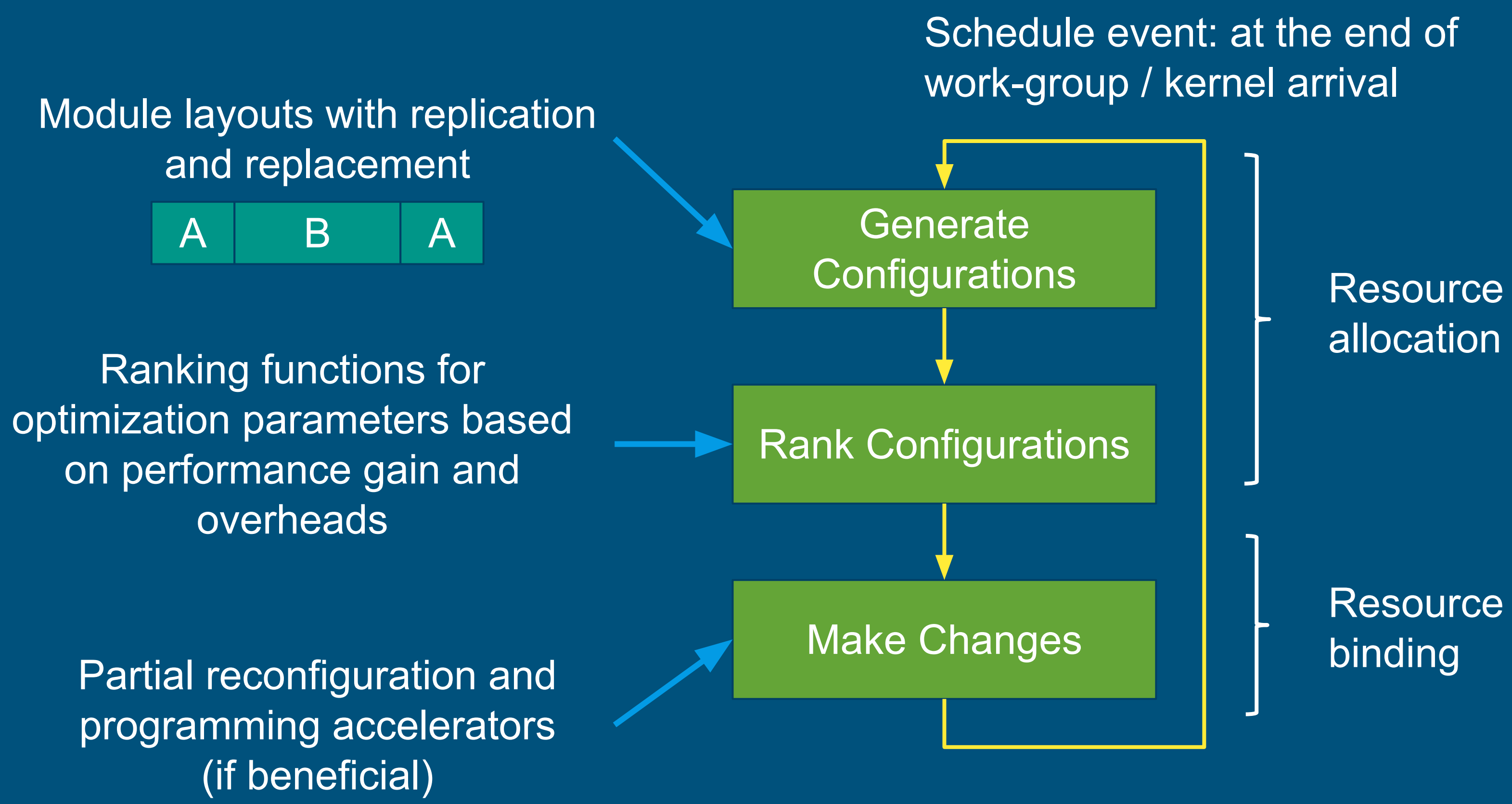
Zynq UltraScale+

# Accelerator generation and execution

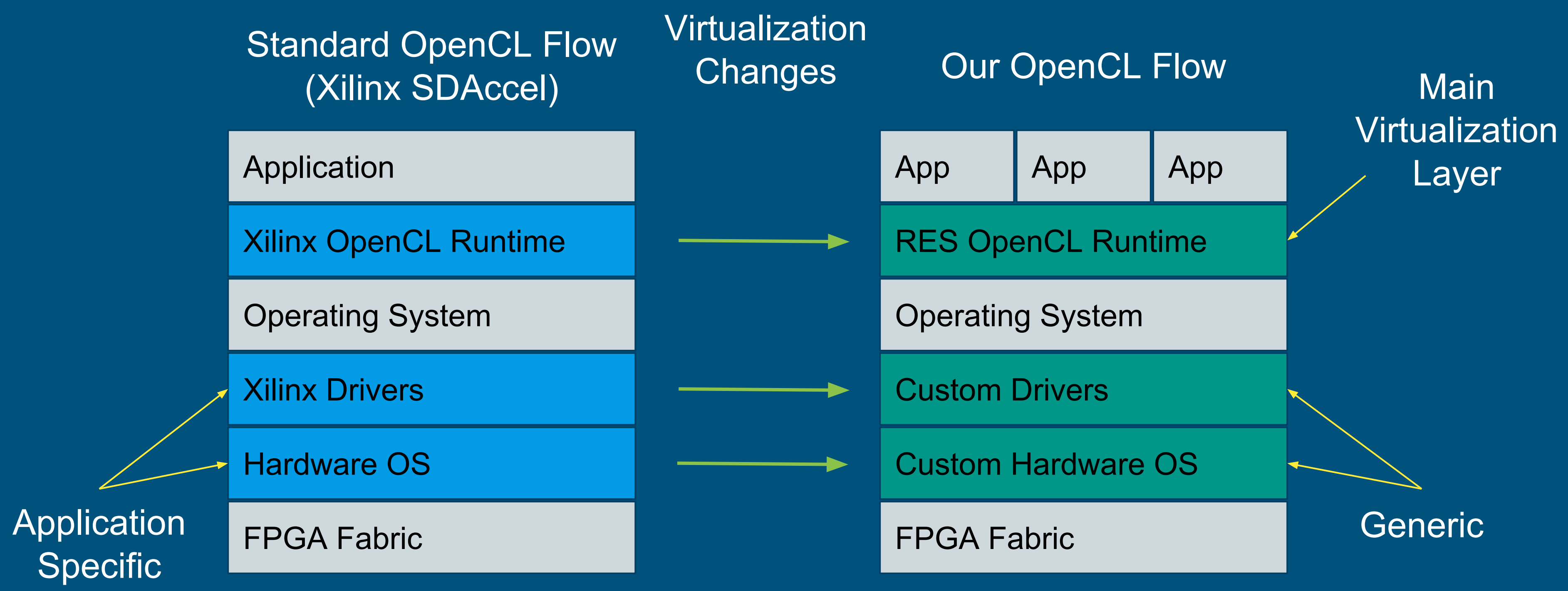


MANCHESTER 1824
The University of Manchester

# Scheduling algorithm



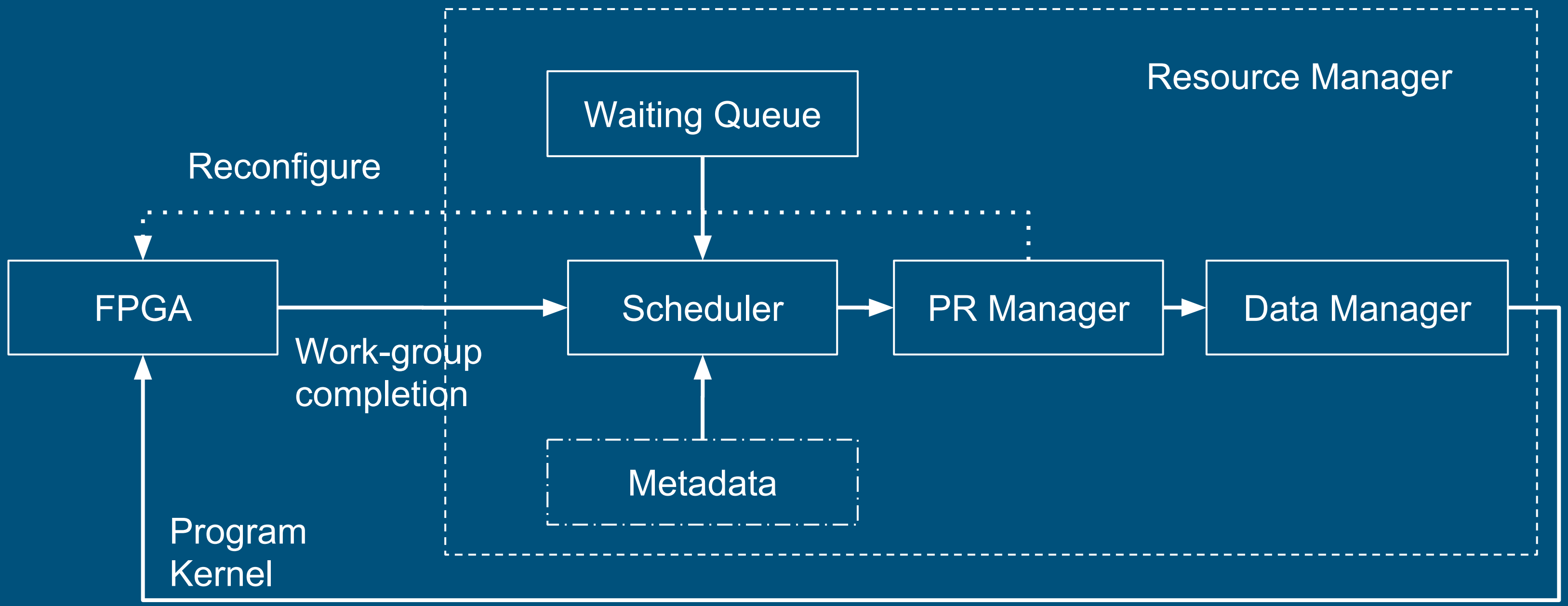
# Virtualization Architecture



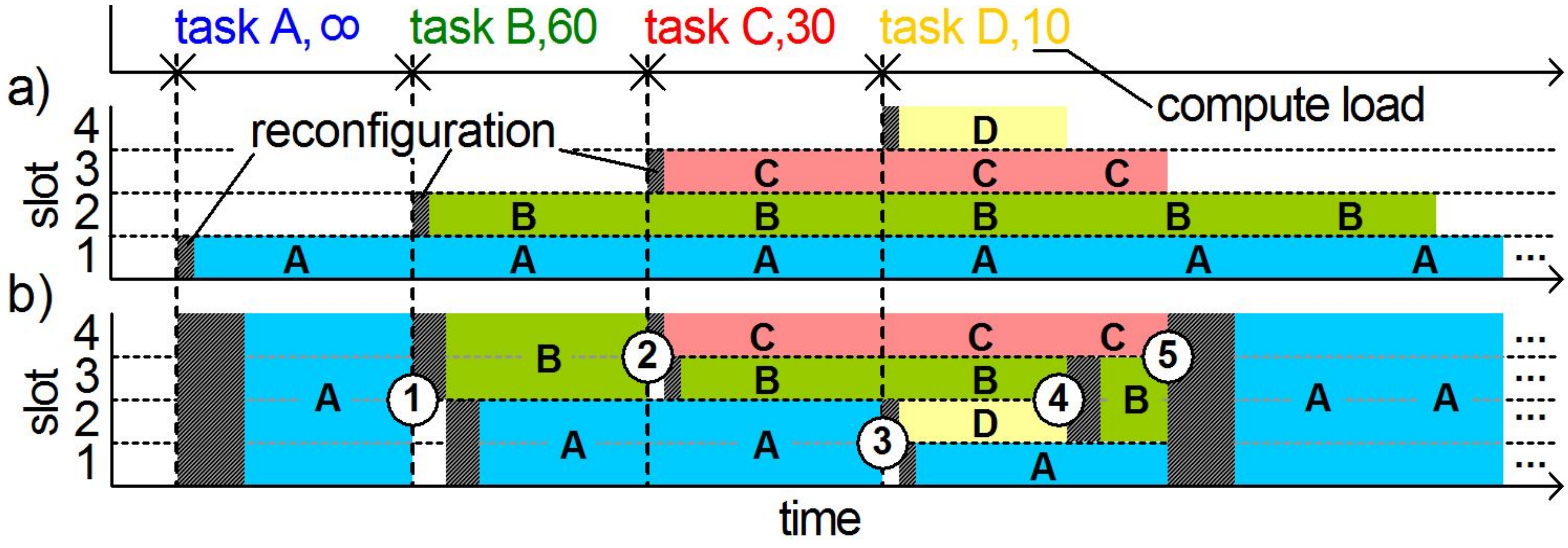
# Time multiplexing

When we run out of space: Swap kernels to waiting queue

- Allows overcommitment of resources by time multiplexing



# Space-Time Scheduling at Runtime



# Evaluation

Baseline scheduling policies:

- Run to Completion (NS)
- Conservative Cooperative Scheduling (CCS)
- Aggressive Cooperative Scheduling (ACS)

Using the same context switching mechanism

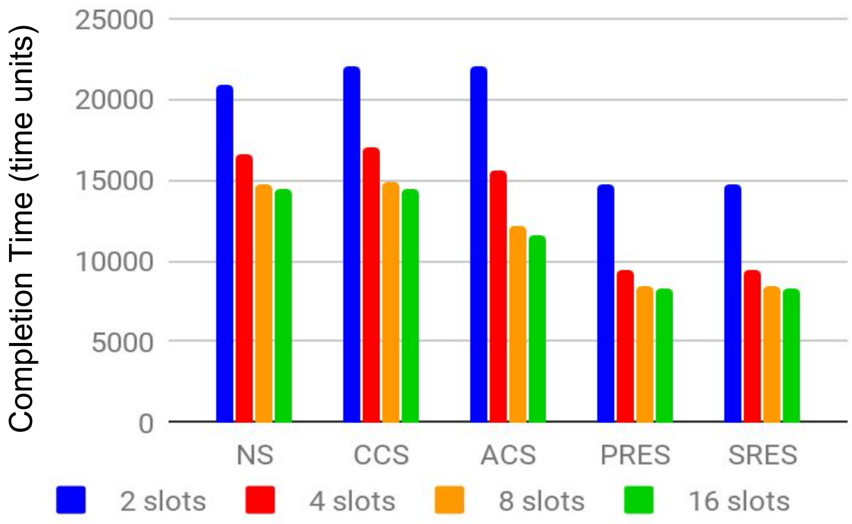
Resource elastic schedulers (RES):

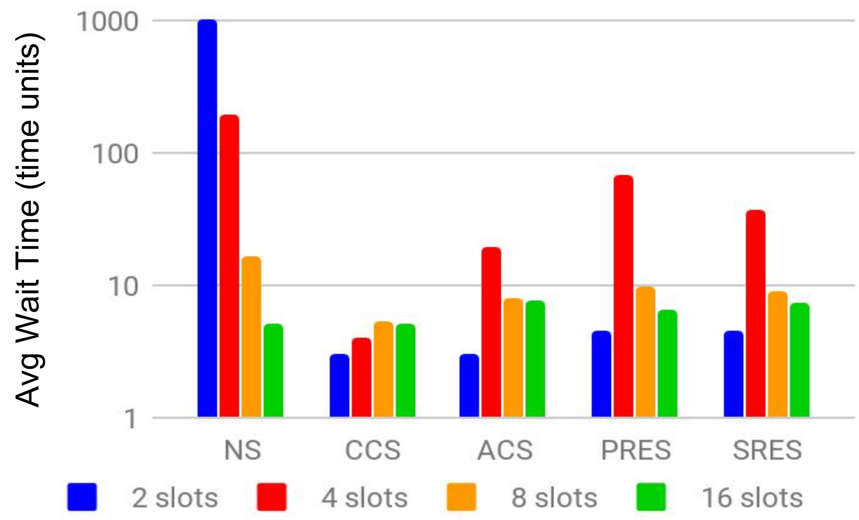
- Standard RES (SRES) : Optimizes for fairness + utilization
- Performance RES (PRES) : Optimizes for performance

Do not look into the future.

# Simulation Results

# Completion and Wait Time Results

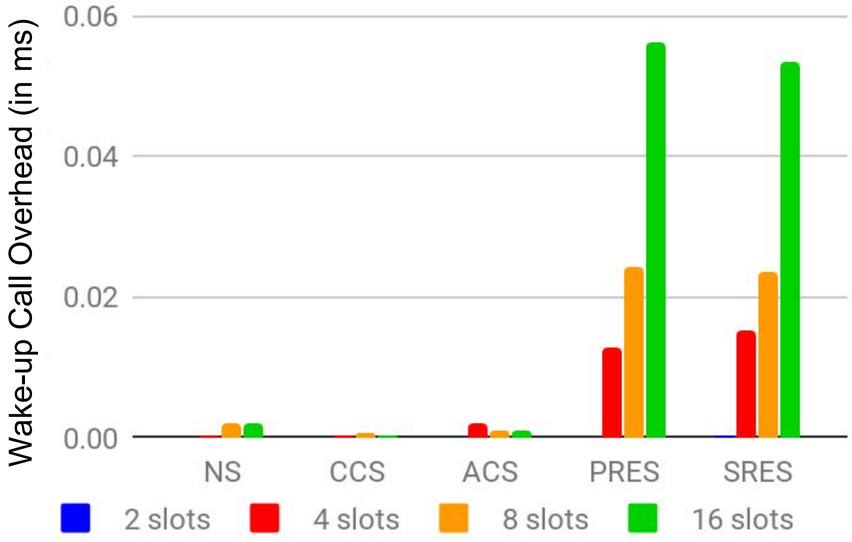


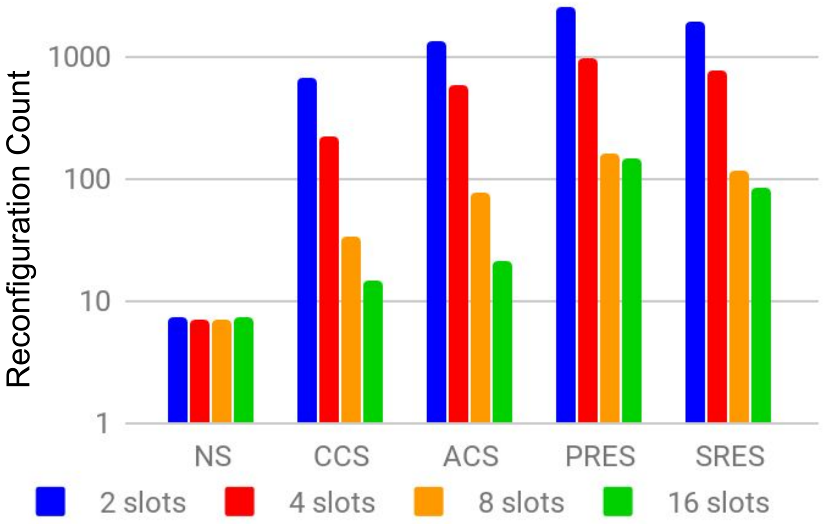




PRES can achieve performance benefit between 39% to 64%

Similar wait times unless module tends to take up the whole FPGA.

# Resource Elastic Scheduler Overhead

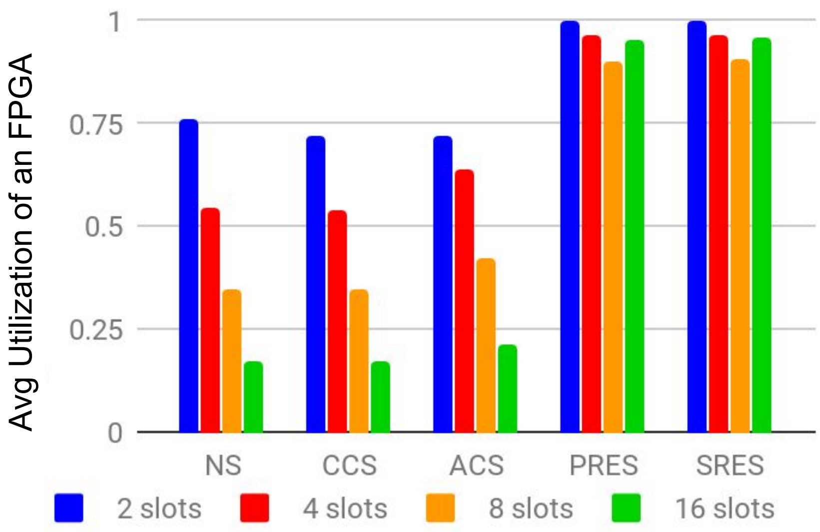


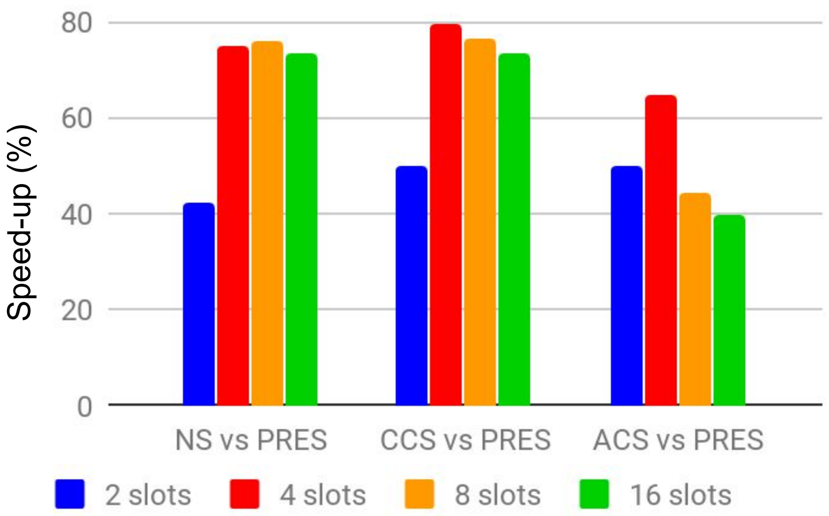




Scheduler wake up call overhead between 12x to 100x

Higher partial reconfiguration calls but relatively similar to ACS

# Utilization and Speedup



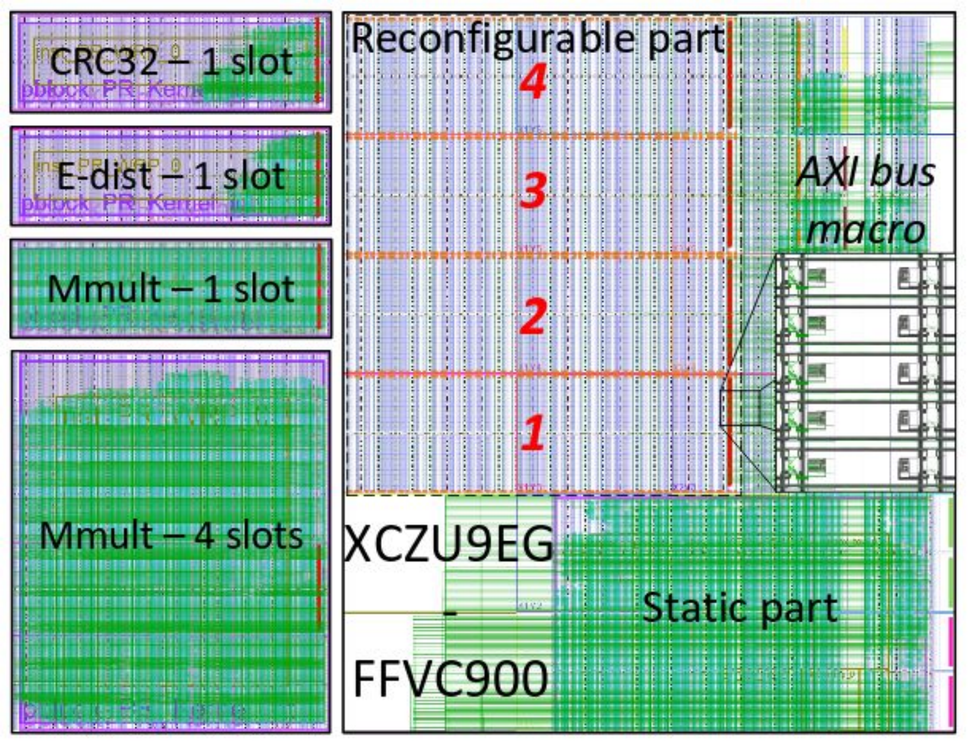




RES improves utilization by 2.3x compared to ACS and 2.7x compared to NS

Provides considerable performance benefits despite the PR overheads.

# Case Study

The static system would be introduced in FSP workshop: ZUCL




GitHub: https://github.com/zuclfpl/zucl_fsp

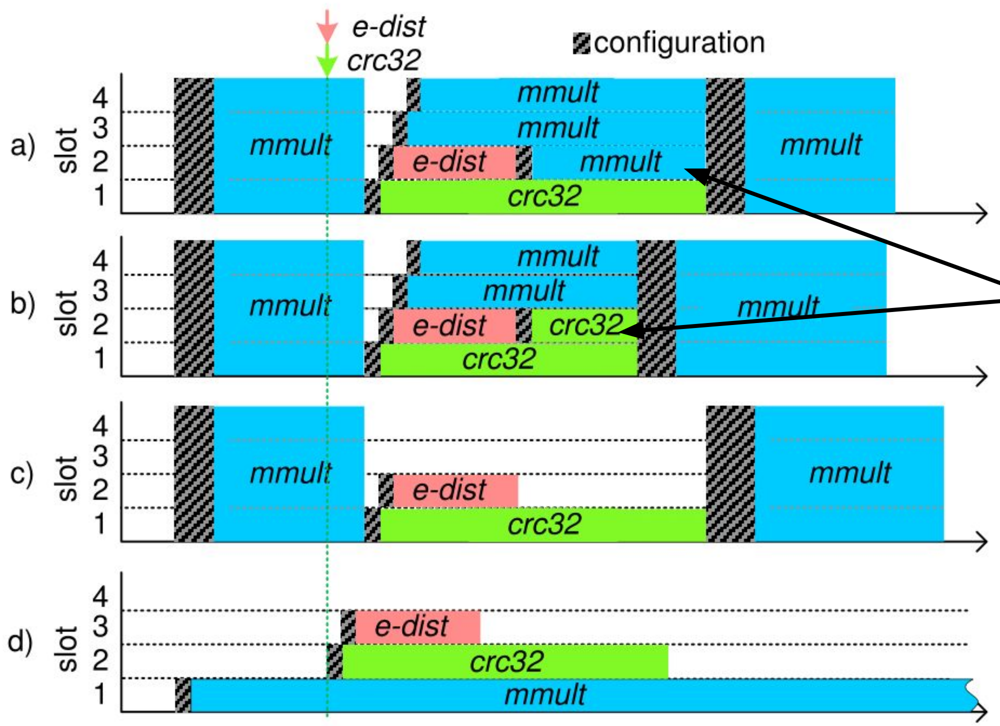
# Case Study

PRES:

SRES:

ACS:

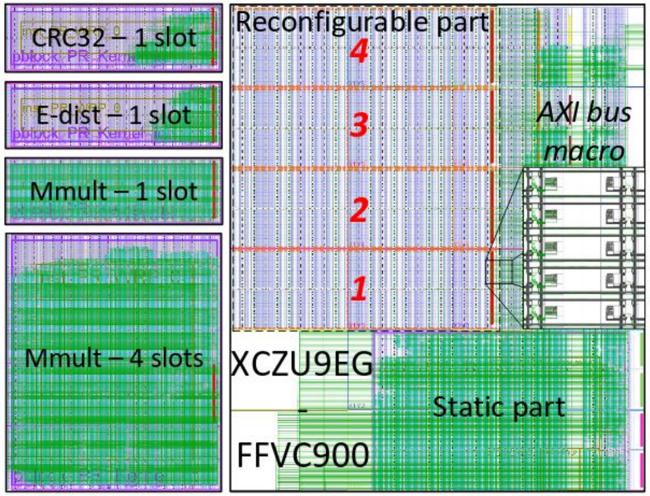
CCS / NS:




PRES gets too greedy while SRES attempts to be fair

# Case Study Results

- Similar wait time
- Performance improvement:
  - **36 % (SRES vs ACS)**
  - **73.8 % (SRES vs CCS/NS)**




| | SRES (ms) | PRES (ms) | ACS (ms) | CCS/NS (ms) |
|---|---|---|---|---|
| **Mmult wait time** | 12 | 12 | 12 | 3 |
| **CRC32 wait time** | 4 | 4 | 4 | 3 |
| **E-dist wait time** | 7 | 7 | 7 | 6 |
| **Total Completion time** | ***320*** | ***368*** | ***501*** | ***1221*** |

# Summary

- Concept of resource elasticity
- Cooperative scheduling for FPGAs
- **First working resource elastic system**
    - **Supporting tool flow: from HLS to Runtime system**
- Performance gains in the range of 39 % to 64%

MANCHESTER
1824
The University of Manchester

# Conclusions

Key takeaways:

- Future OS / virtual machines for FPGAs **need to consider spatial domain**
- **Cooperative scheduling can be a good fit** for FPGAs

Features of RES:

- Higher performance and utilization
- Scale performance with FPGA resources (using dynamic replication)
- Migration of accelerators
- Overcommitment of resources w.r.t. Quality of Service

# LIVE MIGRATION FOR OPENCL FPGA ACCELERATORS

ANUJ VAISHNAV CO-AUTHORS: K. D. PHAM, D. KOCH

SCHOOL OF COMPUTER SCIENCE

THE UNIVERSITY OF MANCHESTER

# INTRODUCTION

- Migration of accelerators across nodes with zero downtime
  - Fault tolerance
  - Maintenance
  - Resource Management
- Same idea as last paper
  - Context switches between workgroups
  - Resource Elasticity – Due to different resource availability

# MIGRATION

- Usually requires saving state
- For CPU task
  - Register state
  - Memory state
  - Etc.
- More difficult for FPGAs
- Migrate between OpenCL workgroups
  - Consistency point
  - Two methods – blocking, non-blocking.

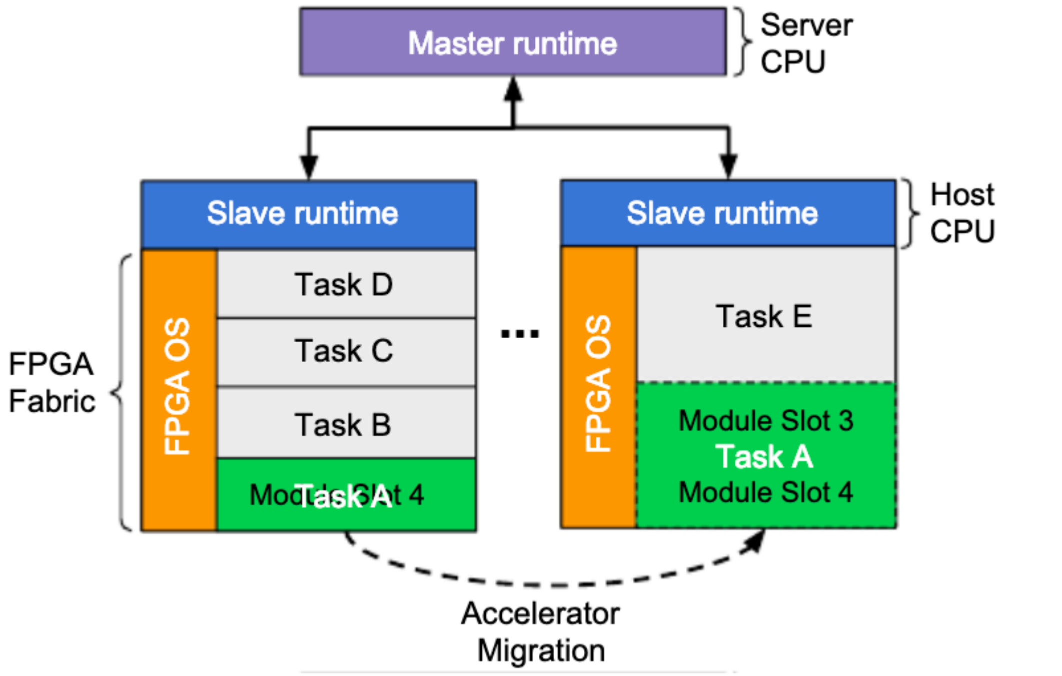
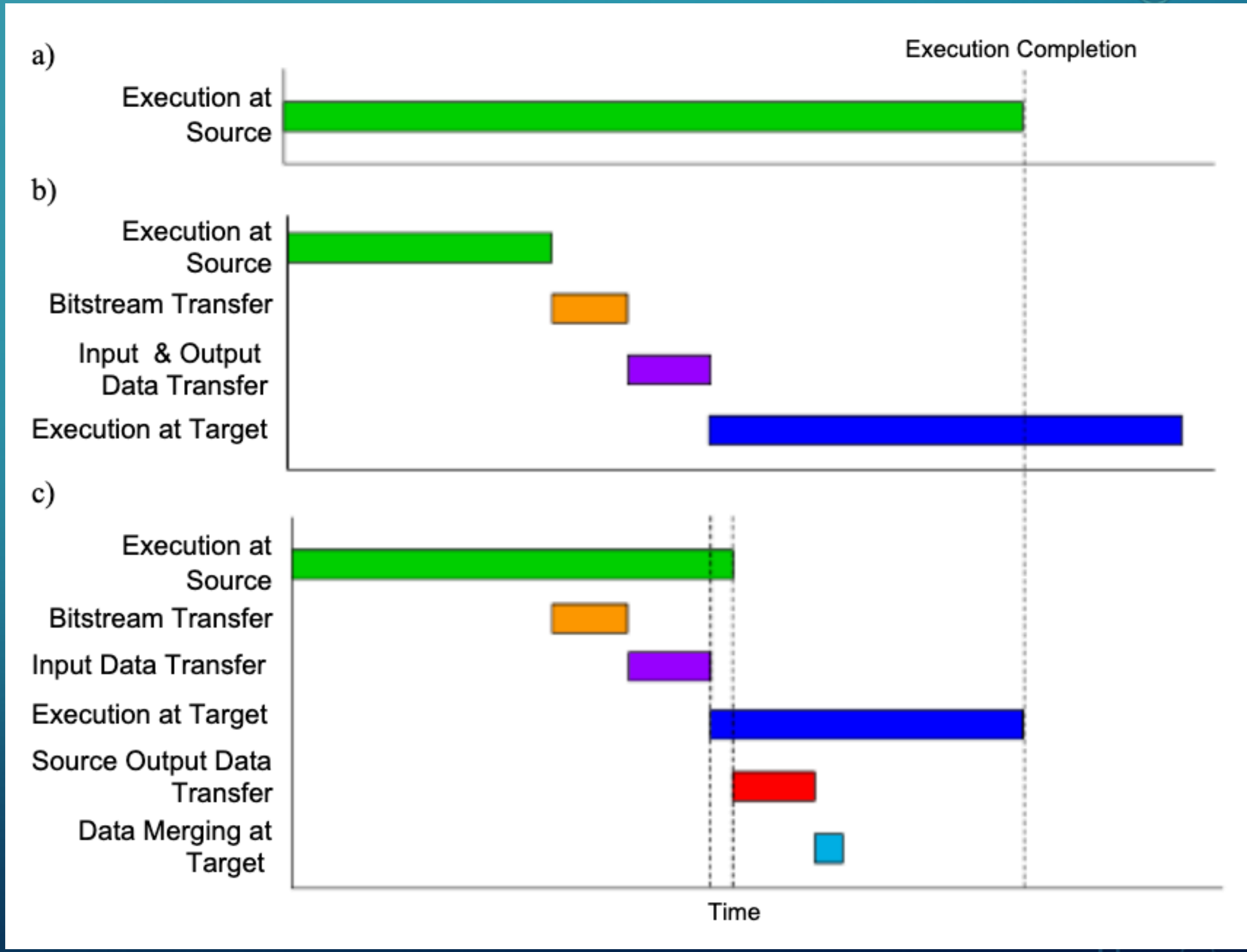


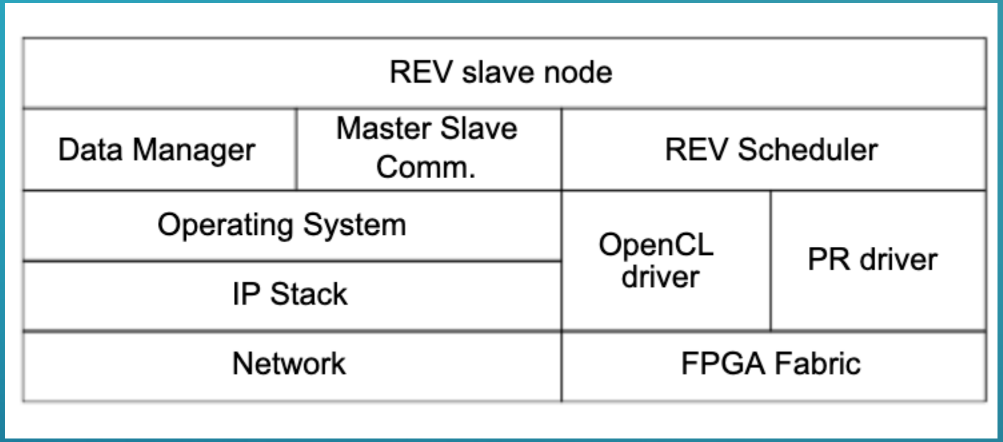

Fig. 1: Migration of FPGA accelerators.

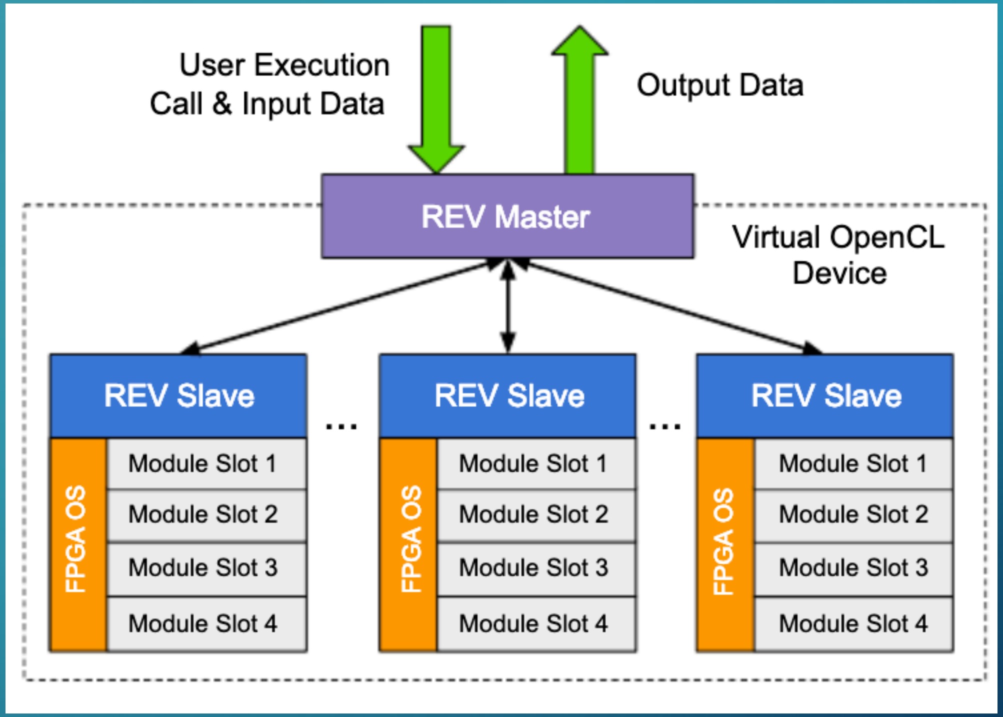
# MIGRATION (CONTD.)

- No Migration
  - Run to completion
- Blocking
  - Pause execution at the end of workgroup
  - Transfer accelerator bitstream, input/output
  - Restart execution
- Non-Blocking
  - Transfer bitstream, input data while previous work group is still in execution
  - Restart next workgroup at target
  - Transfer output from source and merge



# ARCHITECTURE




- Master/Slave approach
- Master runs host code and resource allocation
- Slaves execute accelerators, heartbeat to master
- Enables load balancing

# CONCLUSION

- Live Migration of OpenCL accelerators
  - Asynchronous
  - Transparent
  - Negligible Overhead
- Enables
  - Fault tolerance
  - Load balancing
  - Maintenance

# DISCUSSION

- Efficiently utilize compute devices in HPC nodes
  - Including CPU(s), GPU(s), FPGA(s)
  - Co-schedule job across compute devices by partitioning – Stencil/Mesh applications
    - What ratios?
    - Hide communication – How?
  - Let multiple jobs share the same node
- Currently have same OpenCL kernel running across all devices
  - As a first example split vector addition across all devices (in progress)
- Can we build a virtual OpenCL device combining CPU, GPU(s), FPGA(s)
  - Work group granularity
  - Depending on application(s) – Partition kernel over multiple devices, or share resources with other jobs
  - GPUDirect (maybe FPGADirect) for intra-node/inter-node communication
    - No CPU overhead (almost)