Heterogeneity-Aware Scheduling Algorithms for FPGA Workloads in Cloud Environments

Anand Krishna Rallabhandi
Advisor: Dr. PhD. Atsushi Koshiba
Chair of Distributed Systems and Operating Systems
https://dse.in.tum.de/

15.02.2023 – 15.08.2023

Outline

- Motivation
  - Research Context, Research Gap, Challenges, etc.
- Background
- Design
- Evaluation
- Summary

Research context

- FPGAs deliver increased performance at a lower cost compared with CPUs
  - Used for offloading specific processing - Machine learning, graph processing
- Current approaches do not support heterogeneous FPGAs
  - Insufficient resource assignment

Disclaimer

Work in the thesis mainly deals with scheduling and will be integrated into HeteroFunky

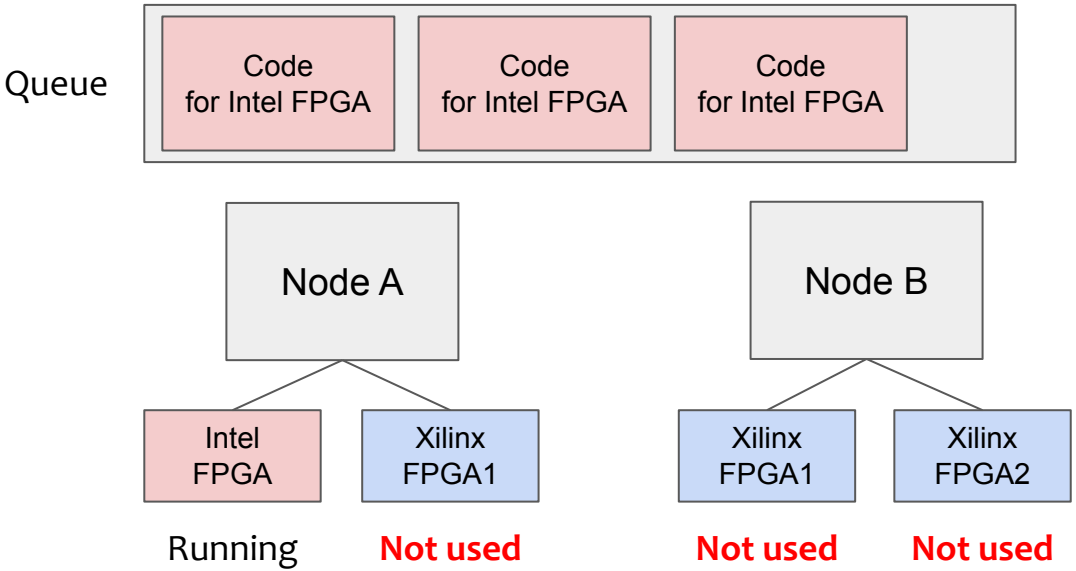
- HeteroFunky is a framework for heterogeneous FPGA virtualization and management
- This work abstracts away virtualization challenges

Research Gap

- Current approaches do not support heterogeneous FPGAs
  - Workloads with different types of FPGAs will have low utilization
  - Do not incorporate performance differences b/w bitstreams compiled from same OpenCL code

Challenge: Underutilization of FPGAs

- Distribution of synthesized bitstreams skewed towards only one target
  - Other FPGAs are not utilized



Synthesize for different targets and pick the as per criterion at a later stage

Challenge: Performance Difference

- Performance difference in same OpenCL kernel code compiled for different FPGAs
  - Differences in Memory Banks, frequency, LUTs, etc. influence this

Consider such properties due to heterogeneity to improve Kernel I/O Throughput

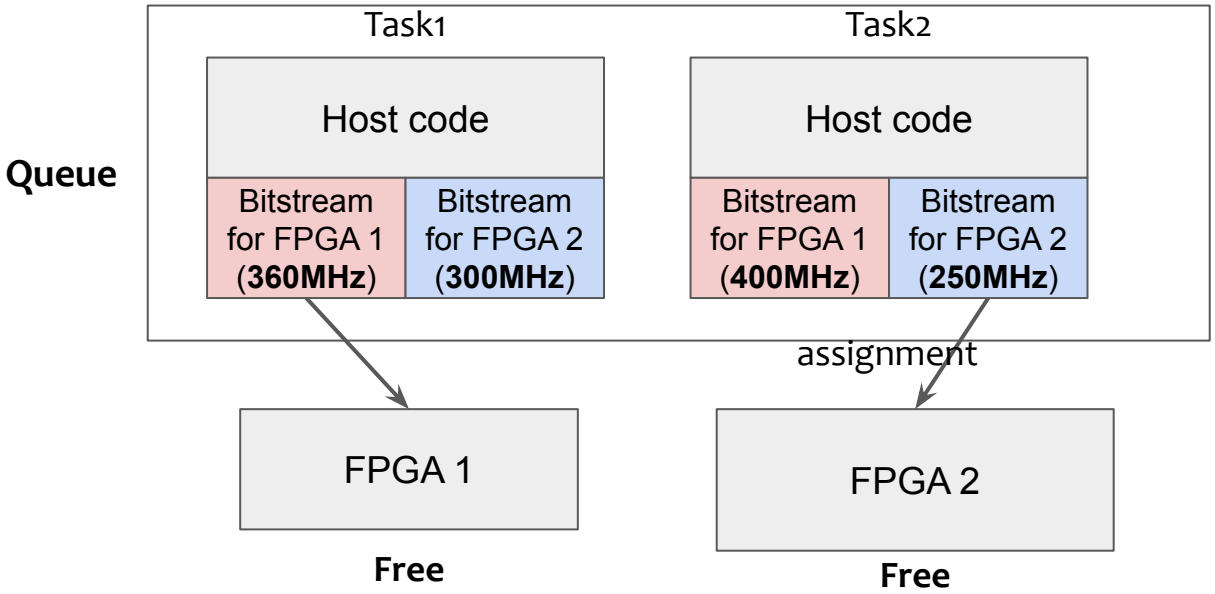
Problem statement

What parameter affects the I/O throughput and how to design schedulers using this parameter to increase I/O throughput?

Outline

- ~~Motivation~~
- Background
  - Frequency-aware Scheduling
- Design
- Evaluation
- Summary

Frequency-aware Scheduling



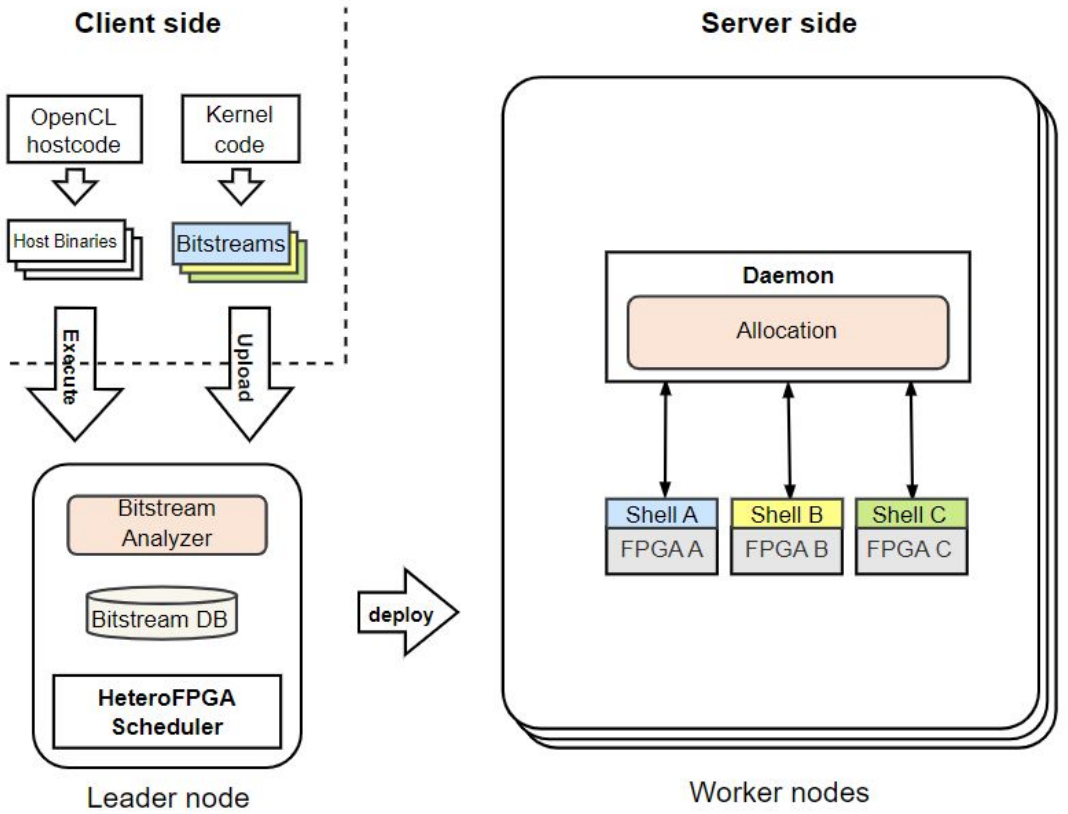
Sort bitstreams in descending order and try to pick the ***best***

Outline

- ~~Motivation~~
- ~~Background~~
- Design
  - System Design, Workflows, Bitstream Storage, etc.
- Evaluation
- Summary

System Design

- Two operations
  - Bitstream Upload
  - Task Execution
- Primary handles bitstreams
  - Uses a Redis datastore
- Workers handle execution
  - Fork-exec-signalfd



Bitstream Storage

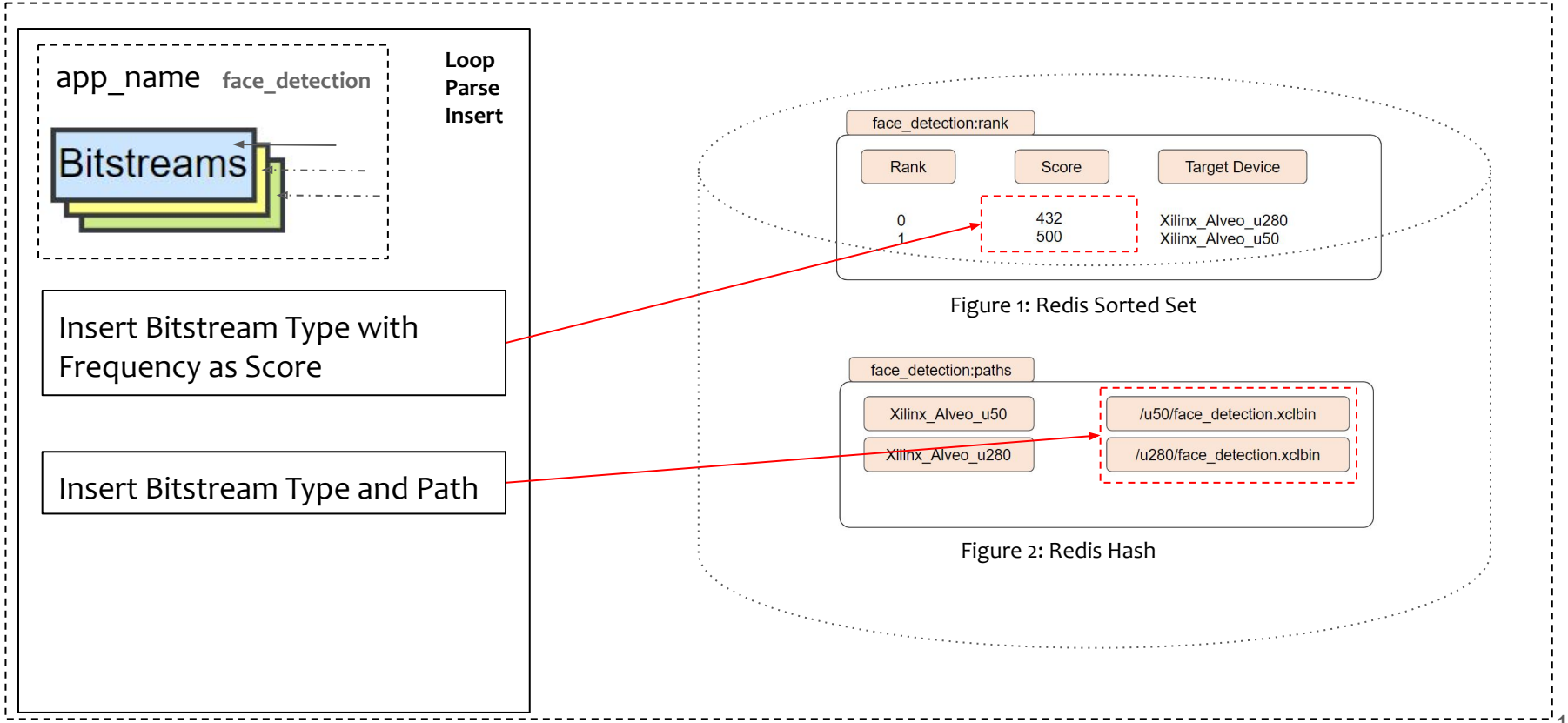
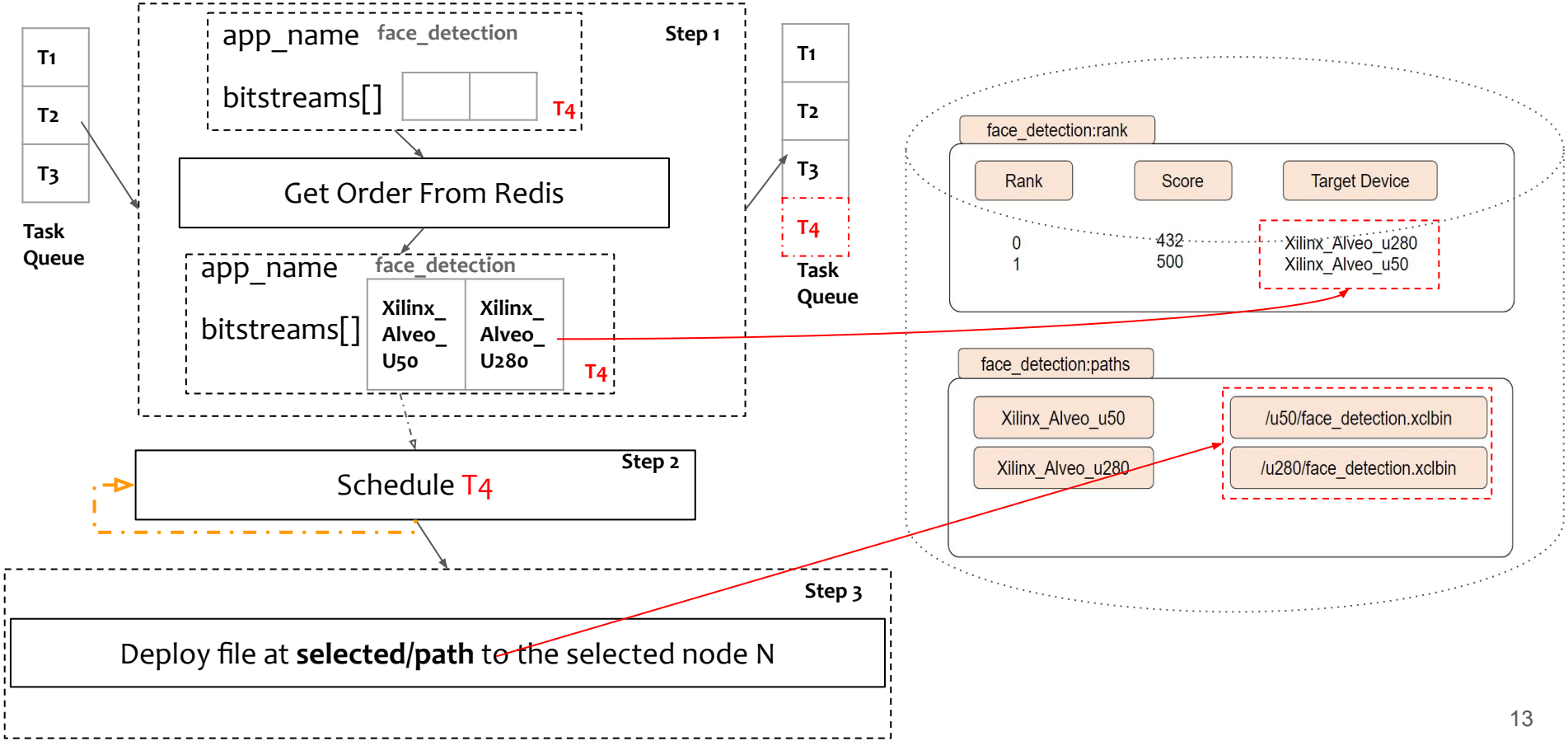


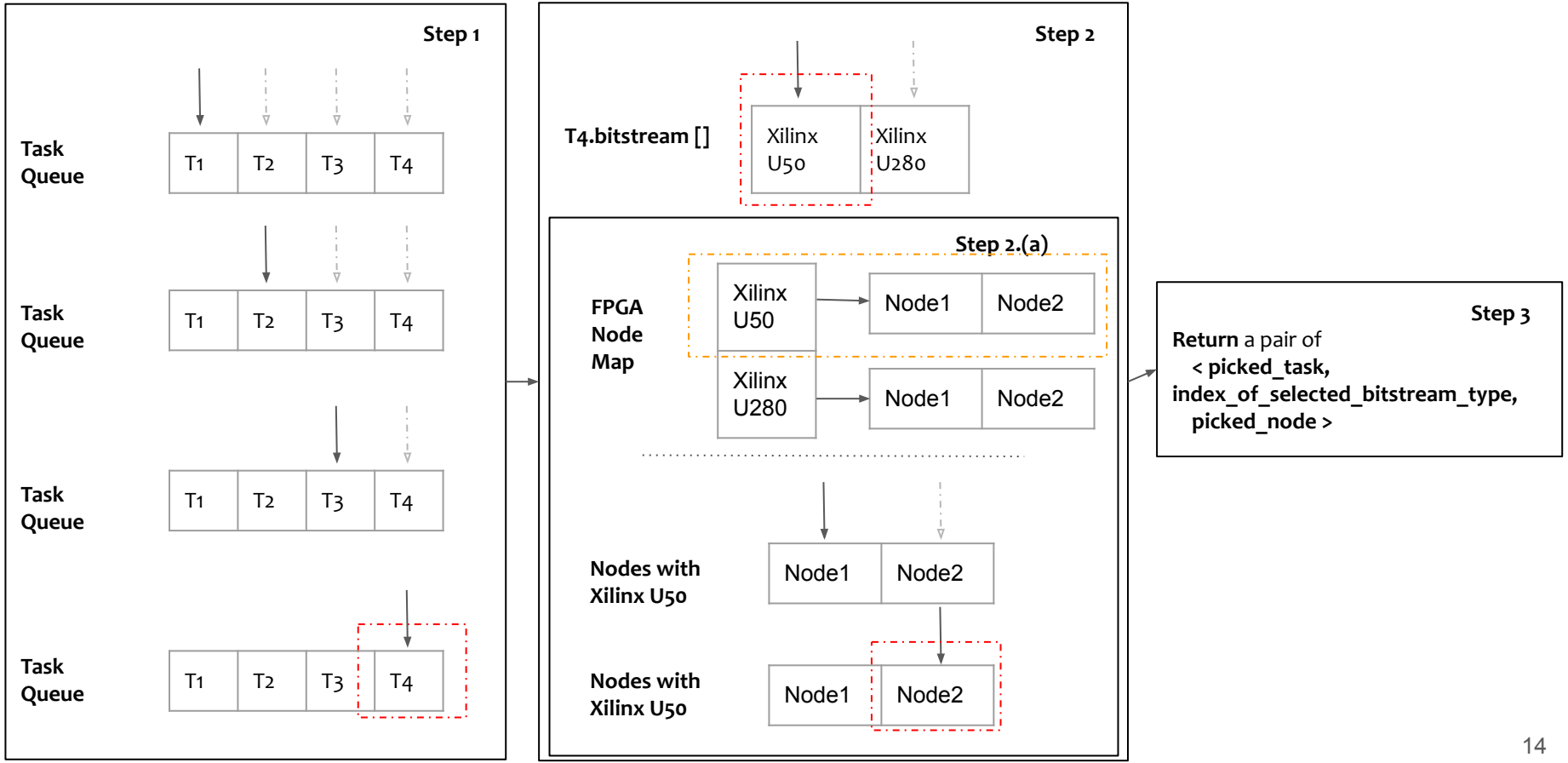
Figure 1: Redis Sorted Set

Figure 2: Redis Hash

Execution


T1
T2
T3
Task Queue
app_name face_detection
bitstreams[]
T4
Step 1
Get Order From Redis
app_name face_detection
bitstreams[]
Xilinx_ Alveo_ U50
Xilinx_ Alveo_ U280
T4
Schedule T4
Step 2
Step 3
Deploy file at selected/path to the selected node N
T1
T2
T3
T4
Task Queue
face_detection:rank
Rank
Score
Target Device
0
1
432
500
Xilinx_Alveo_u280
Xilinx_Alveo_u50
face_detection:paths
Xilinx_Alveo_u50
Xilinx_Alveo_u280
/u50/face_detection.xclbin
/u280/face_detection.xclbin

Scheduling


Step 1
Task Queue
T1
T2
T3
T4
Task Queue
T1
T2
T3
T4
Task Queue
T1
T2
T3
T4
Task Queue
T1
T2
T3
T4
Step 2
T4.bitstream []
Xilinx U50
Xilinx U280
Step 2.(a)
FPGA Node Map
Xilinx U50
Node1
Node2
Xilinx U280
Node1
Node2
Nodes with Xilinx U50
Node1
Node2
Nodes with Xilinx U50
Node1
Node2
Step 3
Return a pair of
< picked_task,
index_of_selected_bitstream_type,
picked_node >

Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- Evaluation
  - Evaluation, Research Questions, Scheduling Policies etc.
- Summary

Evaluation

- Experimental setup:
  - momiji - Intel Xeon Gold CPU (28 cores), 256 GB DRAM
  - Xilinx Alveo u50 and u280 w/ Vitis 2022.1 shells
- Policies:
  - FCFS_RO → First Come First Serve_Random Order
  - FCFS_FA → First Come First Serve_Frequency Aware
- Benchmarks (Overclocked):
  - Vitis Accel Examples (v2022.1)
  - Rosetta

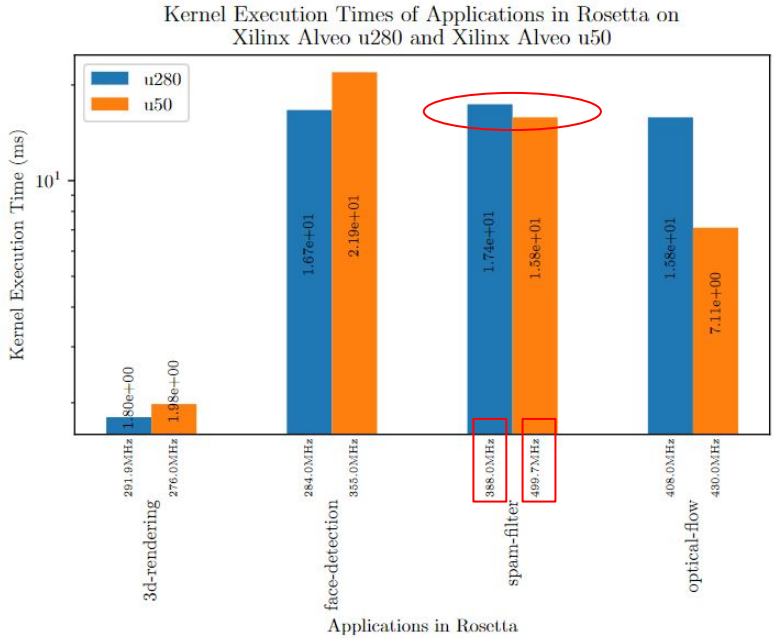
Research Questions

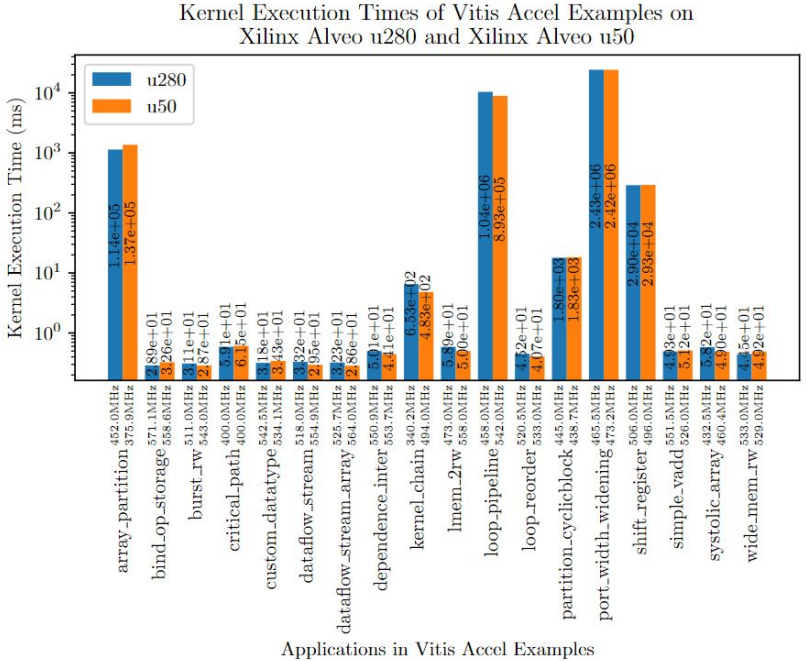
- How does the kernel clock frequency of the bitstream affect the runtime and I/O throughput of the OpenCL kernel?
  - Execute benchmarks on two FPGAs and contrast the measured results
  - **Expectation: (↑ Clock Frequency ⇒ ↑ Kernel I/O Throughput and ↓ Kernel Execution Time) ⇒ Frequency is an influential parameter to use while scheduling**

- Does the I/O throughput increase when we use frequency to schedule?
  - Execute random batches of applications for two policies and measure the total I/O throughput for each policy
  - **Expectation: I/O throughput should increase when we use kernel clock frequency**

Kernel Execution Time

Lower is better

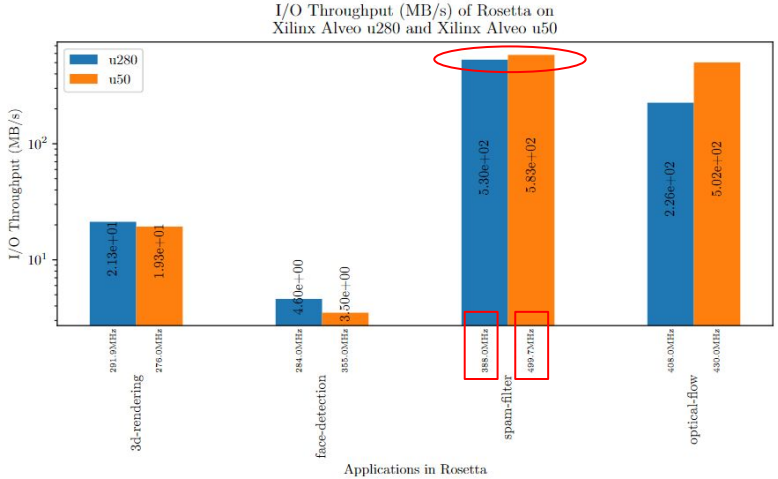


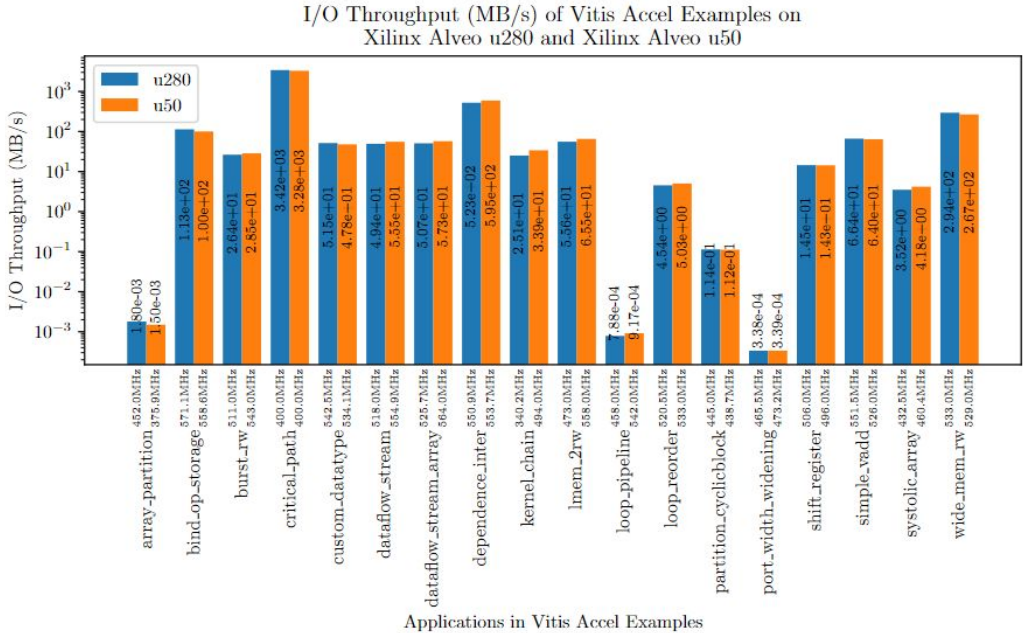


Matches Expectation: ↑ Clock Frequency ⇒ ↓ Kernel Execution Time

I/O Throughput

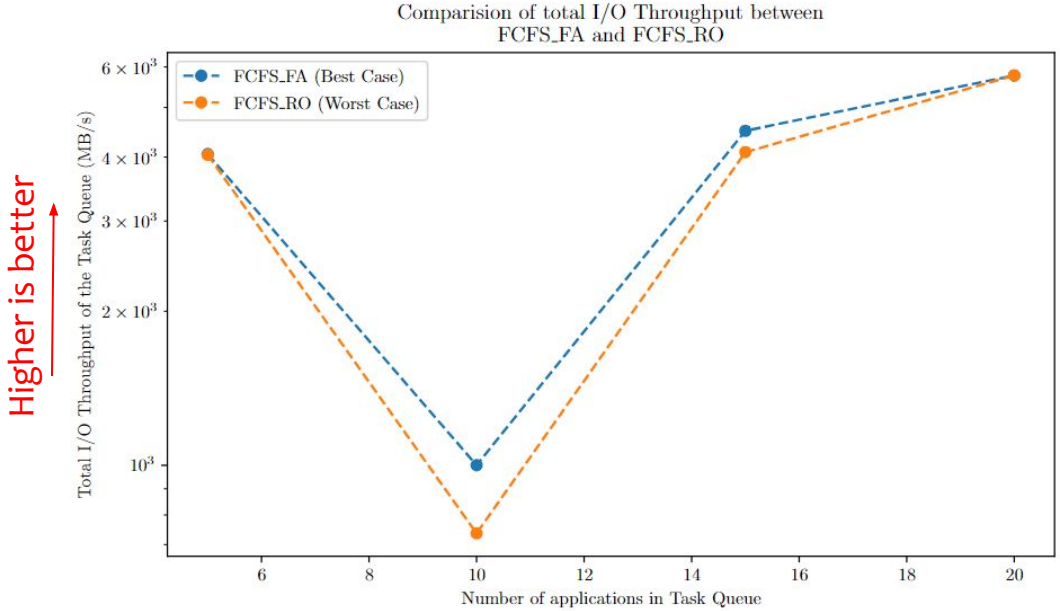
Higher is better





Matches Expectation: ↑ Clock Frequency ⇒ ↑ Kernel I/O Throughput

Scheduling Policies



(b) Kernel I/O throughput comparison across policies.

I/O Throughput more for scheduling policy taking frequency into consideration

Outline

- ~~Motivation~~
- ~~Background~~
- ~~Design~~
- ~~Evaluation~~
- Summary
    - Summary, Future Work

Summary

- Empirical evidence shows that Kernel Clock Frequency has a direct impact on kernel I/O throughput across devices
  - Trend holds good even though U280 has more resources compared to u50 ⇒ **Kernel Clock Frequency is a strong param. influencing I/O throughput**
- Scheduling by considering frequency parameter improves the Kernel I/O throughput
  - Policy trying to pick bitstream with higher frequency 1st performs relatively better ⇒ **Scheduling w/ frequency is a good decision**

Future Work

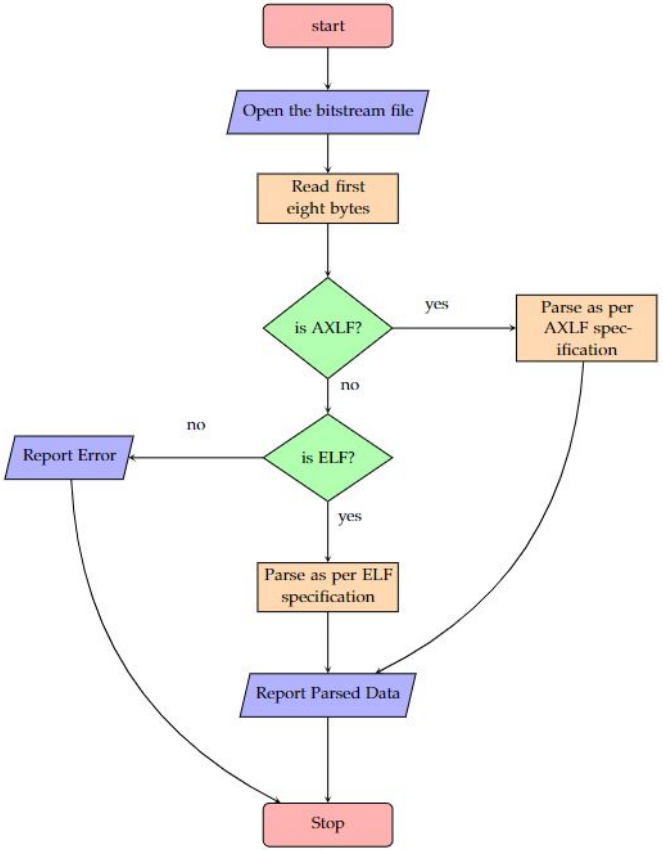
- Consider other factors such as memory bandwidth for memory-bound workloads
- Move to data-driven scheduling algorithms
  - Estimate runtime of an app., for e.g., runtime = f(freq., mem-bw.), and use it to reorder the task queue so that apps. with less runtime finish 1st

Thank You

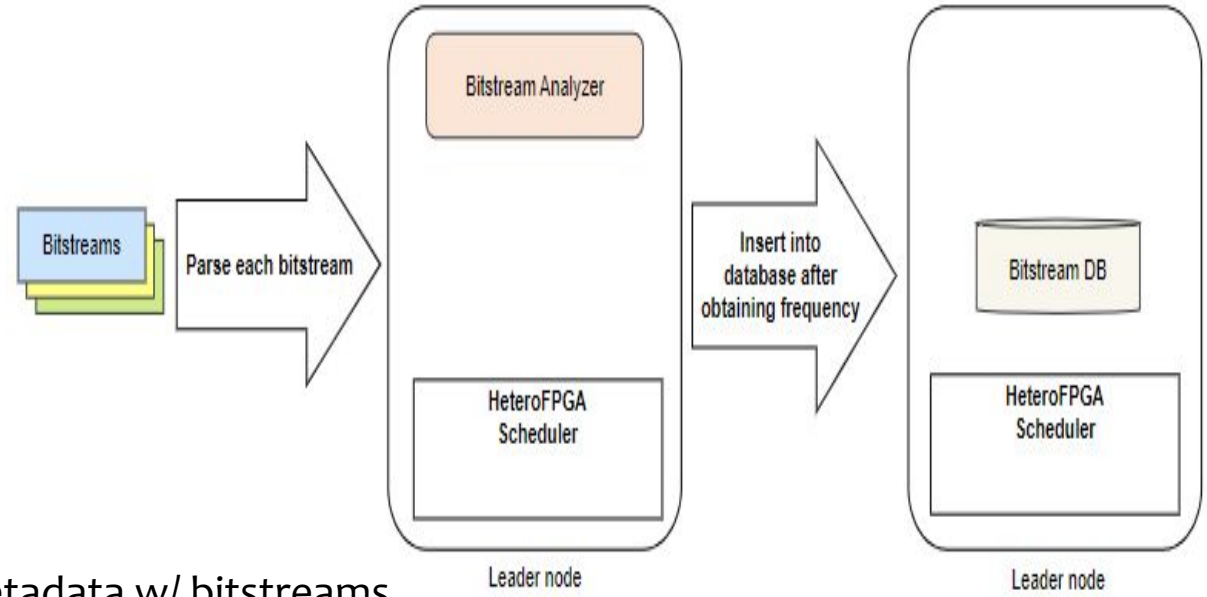
Backup

Working with Bitstreams

- Xilinx FPGAs
  - AXLF file (xclbin binary format). Well structured API from Xilinx for parsing
- Intel FPGAs
  - ELF file. Requires Ad-hoc code for parsing - No support from Intel
- Obtain ***kernel clock frequency*** from the bitstream files after parsing specific sections

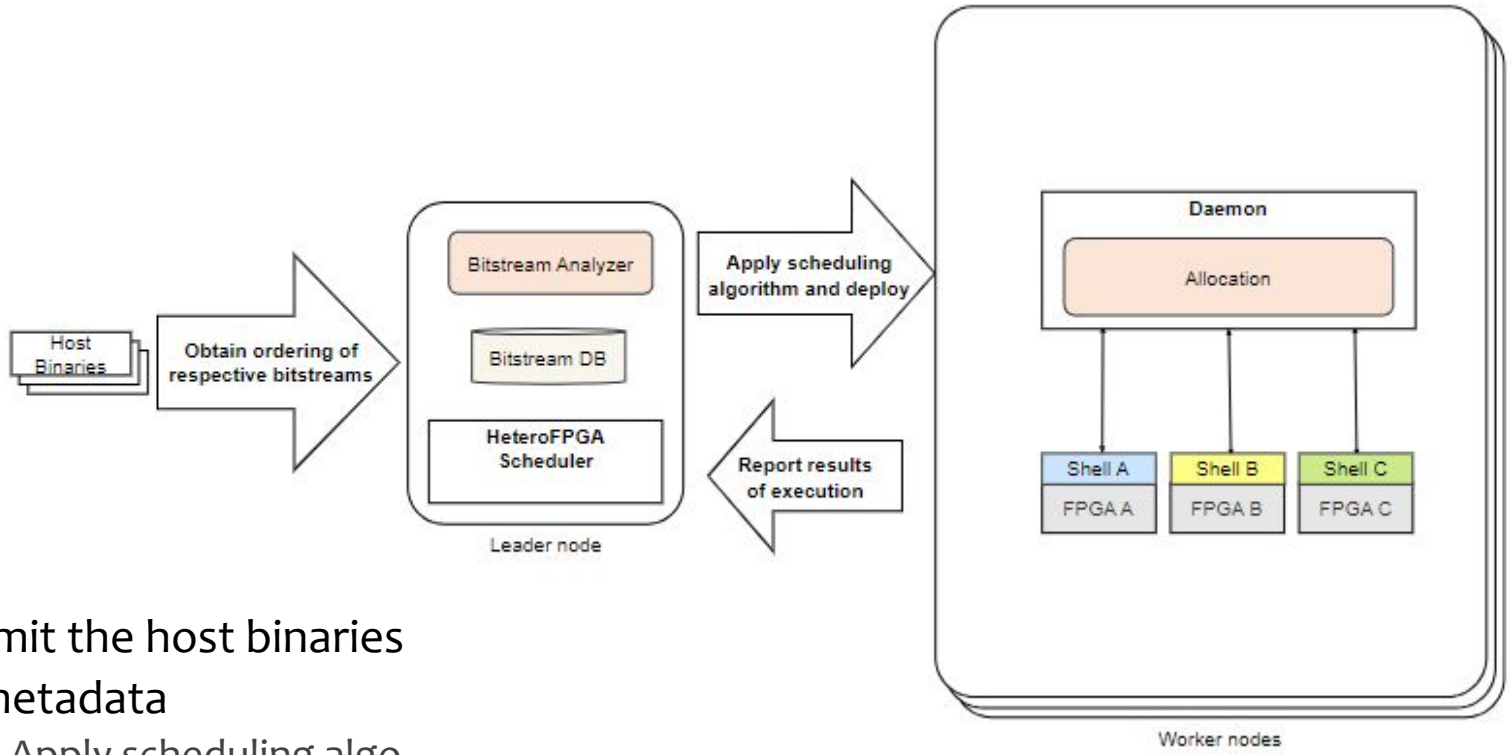


Bitstream Upload - Workflow



- Submit metadata w/ bitstreams
  - Parse and obtain frequency values
  - Insert bitstream related data into Redis

Task Execution - Workflow



- Submit the host binaries w/ metadata
  - Apply scheduling algo.
  - Deploy to worker

Scheduling Algorithm

Algorithm 1: Scheduling Algorithm

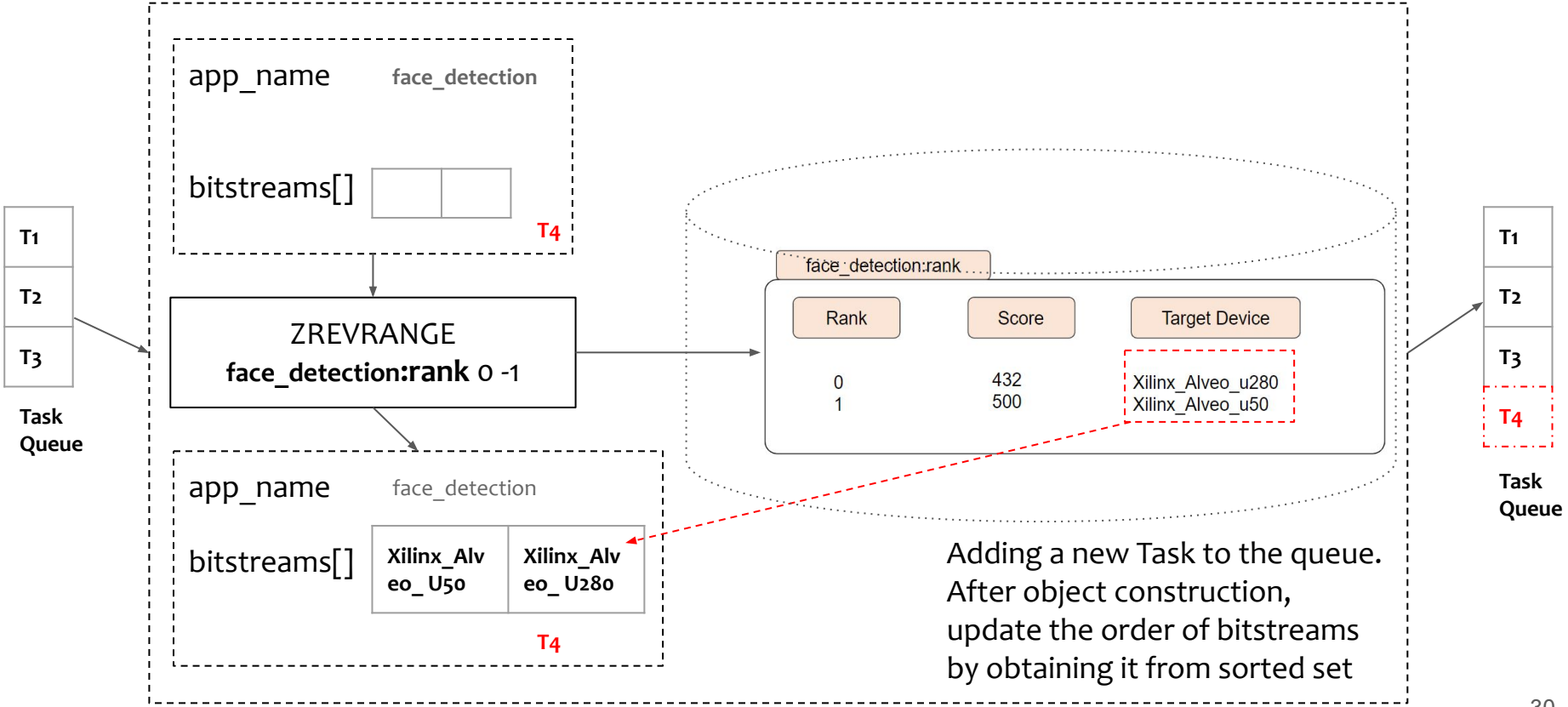
```
Input  : Task list, Nodes map
Output: Picked Task, Picked Node, Index of Bitstream to execute
/* Pick a task, pick a bitstream in it, and pick a node to run the bitstream on. */
schedule(tasks, nodes)
begin
    picked_task, picked_node, bitstream_index ← NULL, NULL, -1;
    foreach task in tasks do
        if task.state == available then
            picked_task ← task;
            break;
    end
    if picked_task == NULL then
        // Nothing to execute as of now.
        return picked_task, picked_node, bitstreamIndex;
    for i = 1 to picked_task.num_bitstreams do
        // Get the target fpga on which this will be executed.  For e.g., Xilinx u50.
        target_type ← picked_task.bitstreams[i].target_type;
        foreach node in nodes[target_type] do
            // Pick the first node that can run this.
            if node.state == available then
                picked_node ← node;
                bitstream_index ← i;
                return picked_task, picked_node, bitstream_index;
        end
    end
    return picked_task, picked_node, bitstream_index;
end
```

Workload Categorization

Suite	Workload	Category
Rosetta	3D rendering	Balanced
	Face detection	Compute-bound
	Spam detection	Memory-bound
	Optical flow	Memory-bound
Vitis Accel Examples	Array partition	None
	Burst read/write	Memory-bound
	BIND OP and STORAGE	Compute-bound
	Critical path	Compute-bound
	Custom data type	Memory-bound
	Dataflow using HLS stream	Compute-bound
	Dataflow using array of HLS stream	Compute-bound
	Loop dependency inter	Compute-bound
	Global memory two banks	Memory-bound
	Stream chain matrix multiplication	Compute-bound
	Local memory two parallel read/write	Memory-bound
	Loop pipelining	Compute-bound
	Array block and cyclic partitioning	Balanced
	Port width widening	Memory-bound
	PLRAM memory access	Memory-bound
	Vector addition	Memory-bound
	Shift register	Compute-bound
	Systolic array	Compute-bound
	Wide memory read/write	Memory-bound

Consider memory properties along with frequency

Scheduling - Task Queue



Bitstream Deployment