Embedded Systems & Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

Improving Job Launch Rates in the TaPaSCo FPGA Middleware by Hardware/Software-Co-Design

Carsten Heinz Jaco Hofmann Lukas Sommer Andreas Koch

2020-11-13

Embedded Systems and Applications, TU Darmstadt

Introduction

- HPC workloads increasingly diverse
- Hardware accelerators for higher performance
  - GPUs
  - Vector Processors
  - **FPGAs**

→ Heterogeneous systems

Introduction

- HPC workloads increasingly diverse
- Hardware accelerators for higher performance
  - GPUs
  - Vector Processors
  - **FPGAs**

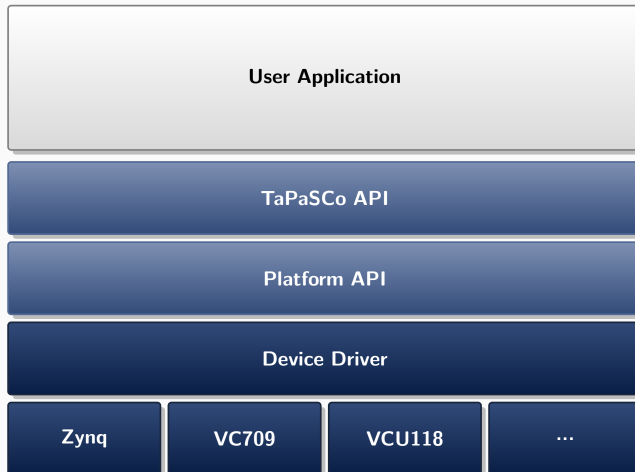
$\rightarrow$ Heterogeneous systems

- Tools required for improved usability

TaPaSCo

- System-on-Chip generator for FPGAs
- API for software interaction
- Toolflow for hardware generation
- Many supported FPGA platforms:
  - Xilinx Alveo U280
  - Amazon AWS F1
  - and over 10 more
- Available under open source license

TaPaSCo - Software API



TaPaSCo - Software API Example

```
1  /* Perform automatic initialization of first device: */
2  Tapasco tapasco;
3  /* data buffer */
4  std::vector<int> v = {0, 1, ...};
5  auto buf = makeWrappedPointer(v.data(), v.size() * sizeof(int));
6  /* Launch a TaPaSCo job */
7  auto myjob = tapasco.launch(KERNEL_ID, buf);
8  /* Wait on Future myjob for completion */
9  myjob();
10 ...
```

TaPaSCo - Software API Example

```
1  /* Perform automatic initialization of first device: */
2  Tapasco tapasco;
3  /* data buffer */
4  std::vector<int> v = {0, 1, ...};
5  auto buf = makeWrappedPointer(v.data(), v.size() * sizeof(int));
6  /* Launch a TaPaSCo job */
7  auto myjob = tapasco.launch(KERNEL_ID, buf);
8  /* Wait on Future myjob for completion */
9  myjob();
10 ...
```

TaPaSCo - Software API Example

```
1  /* Perform automatic initialization of first device: */
2  Tapasco tapasco;
3  /* data buffer */
4  std::vector<int> v = {0, 1, ...};
5  auto buf = makeWrappedPointer(v.data(), v.size() * sizeof(int));
6  /* Launch a TaPaSCo job */
7  auto myjob = tapasco.launch(KERNEL_ID, buf);
8  /* Wait on Future myjob for completion */
9  myjob();
10 ...
```

TaPaSCo - Software API Example

```
1  /* Perform automatic initialization of first device: */
2  Tapasco tapasco;
3  /* data buffer */
4  std::vector<int> v = {0, 1, ...};
5  auto buf = makeWrappedPointer(v.data(), v.size() * sizeof(int));
6  /* Launch a TaPaSCo job */
7  auto myjob = tapasco.launch(KERNEL_ID, buf);
8  /* Wait on Future myjob for completion */
9  myjob();
10 ...
```

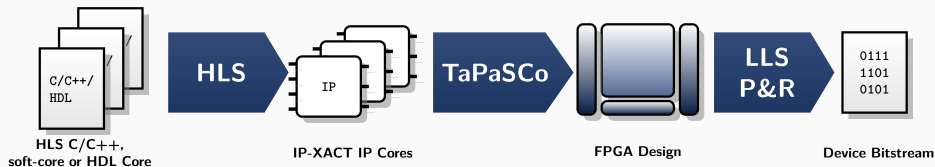
TaPaSCo - Software API Example

```
1  /* Perform automatic initialization of first device: */
2  Tapasco tapasco;
3  /* data buffer */
4  std::vector<int> v = {0, 1, ...};
5  auto buf = makeWrappedPointer(v.data(), v.size() * sizeof(int));
6  /* Launch a TaPaSCo job */
7  auto myjob = tapasco.launch(KERNEL_ID, buf);
8  /* Wait on Future myjob for completion */
9  myjob();
10 ...
```

TaPaSCo - Software API Example

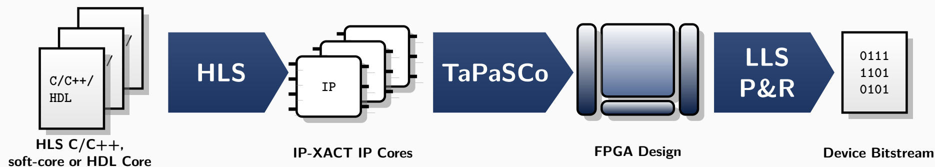
```
1   /* Perform automatic initialization of first device: */
2   Tapasco tapasco;
3   /* data buffer */
4   std::vector<int> v = {0, 1, ...};
5   auto buf = makeWrappedPointer(v.data(), v.size() * sizeof(int));
6   /* Launch a TaPaSCo job */
7   auto myjob = tapasco.launch(KERNEL_ID, buf);
8   /* Wait on Future myjob for completion */
9   myjob();
10  ...
```

TaPaSCo - Hardware Toolflow




```
tapasco compose [acc0 x 3, acc1 x 2] @ 200 MHz -p AU280
```

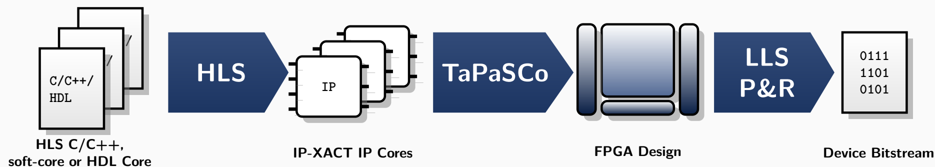
TaPaSCo - Hardware Toolflow




```
tapasco compose [acc0 x 3, acc1 x 2] @ 200 MHz -p AU280
```

↑
PE

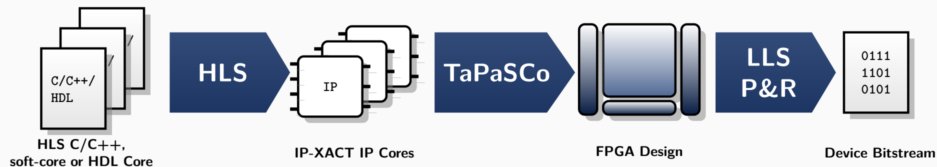
TaPaSCo - Hardware Toolflow




```
tapasco compose [acc0 x 3, acc1 x 2] @ 200 MHz -p AU280
```

↑ PE   ↑ Count

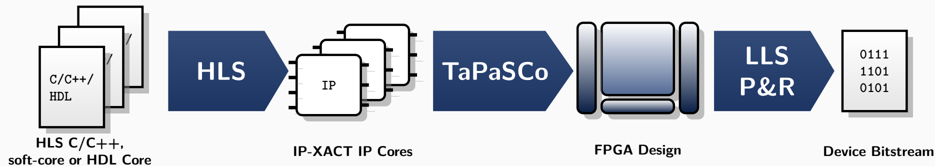
TaPaSCo - Hardware Toolflow




```
tapasco compose [acc0 x 3, acc1 x 2] @ 200 MHz -p AU280
```

↑ PE ↑ Count ↑ Frequency

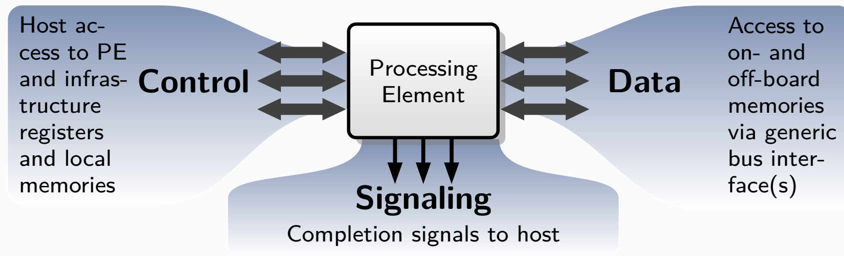
TaPaSCo - Hardware Toolflow




```
tapasco compose [acc0 x 3, acc1 x 2] @ 200 MHz -p AU280
```

↑ PE ↑ Count ↑ Frequency ↑ Platform

TaPaSCo PE - Control Interface



TaPaSCo Runtime

- Runtime is responsible for:
    - Initialization
    - Job Launch
    - Job Finalization

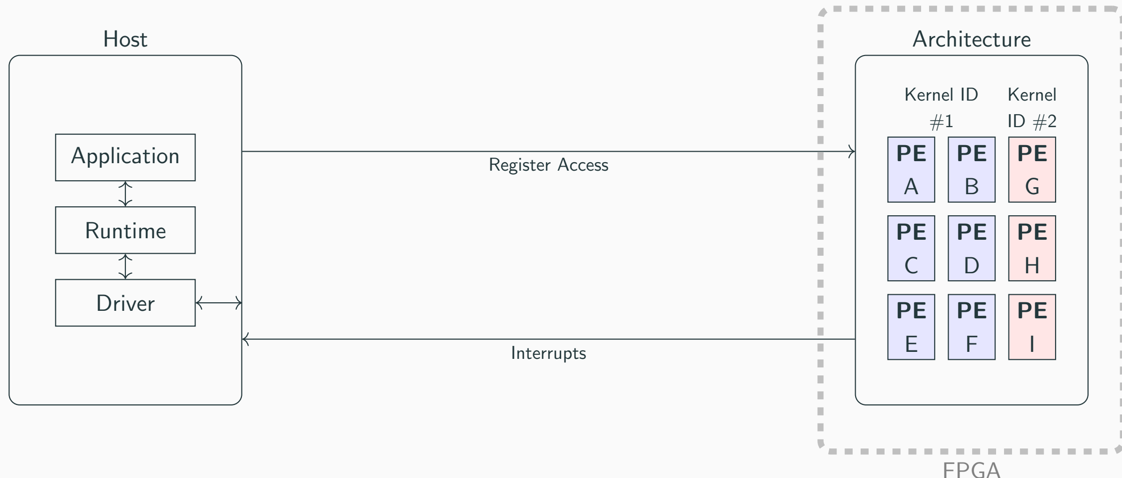
TaPaSCo Runtime

- Runtime is responsible for:
  - Initialization
  - Job Launch
  - Job Finalization
- Reimplemented in Rust
  - Improve maintainability
  - Enable safer concurrency
- C and C++ API provided by a Foreign-Function-Interface (FFI)
- Interrupt handling uses eventfd mechanism

TaPaSCo Cascabel - Hardware Offloading

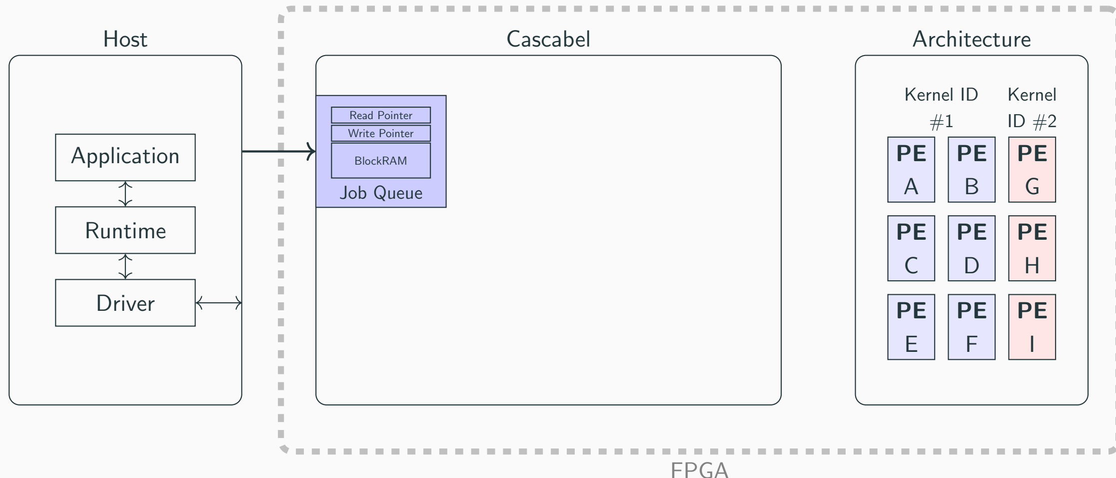
- Offloading launching functionality of runtime to FPGA
  - Reduce CPU load
  - Reduce communication between host and FPGA
- Custom hardware module specific for configuration
  - Queue stores list of jobs
  - Job is launched on available processing elements
  - Barriers for synchronization

TaPaSCo Cascabel - Hardware Offloading

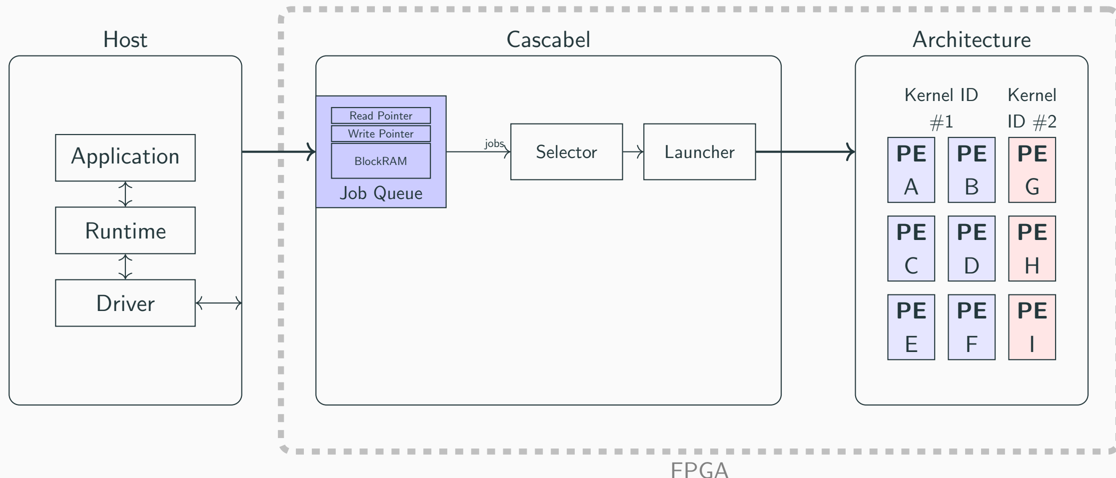


TaPaSCo Cascabel - Hardware Offloading

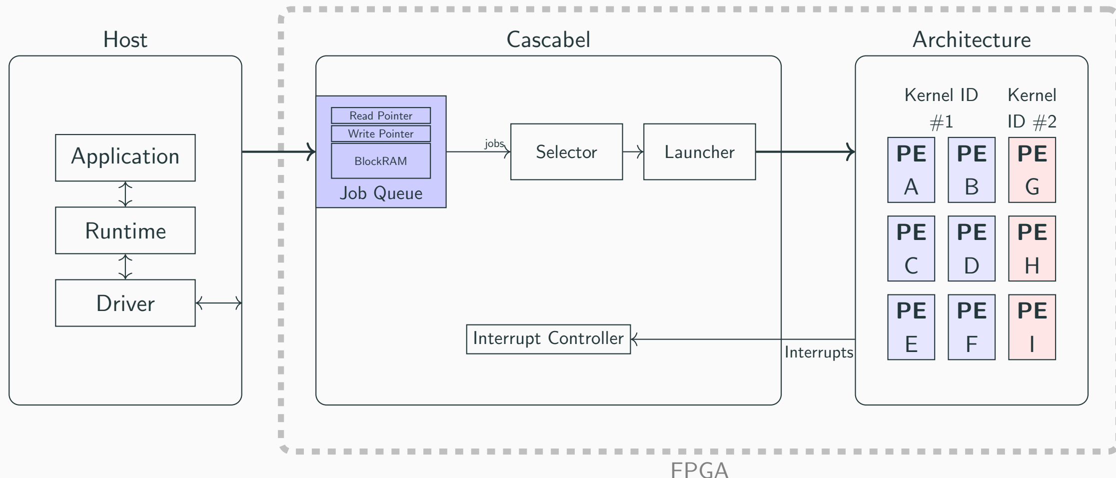
TaPaSCo Cascabel - Hardware Offloading



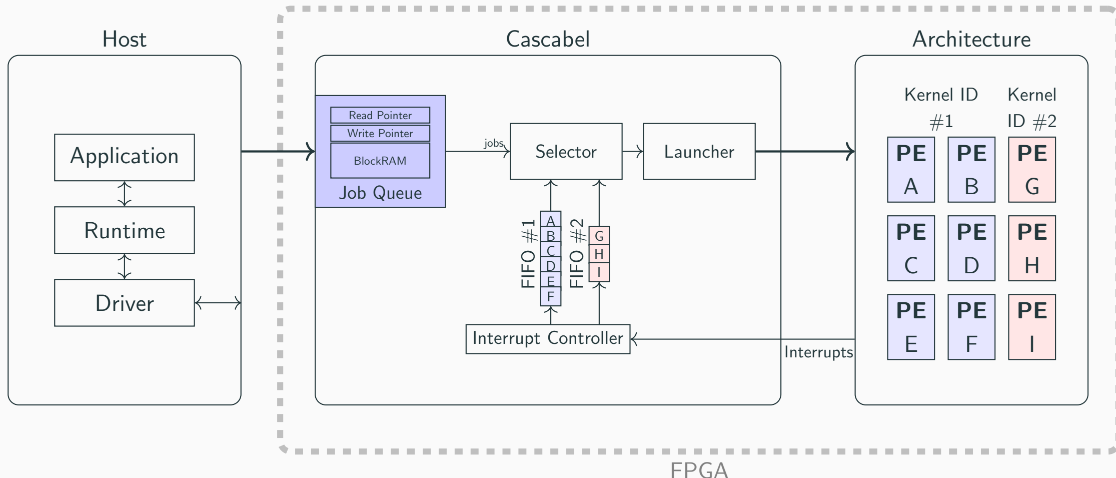
TaPaSCo Cascabel - Hardware Offloading



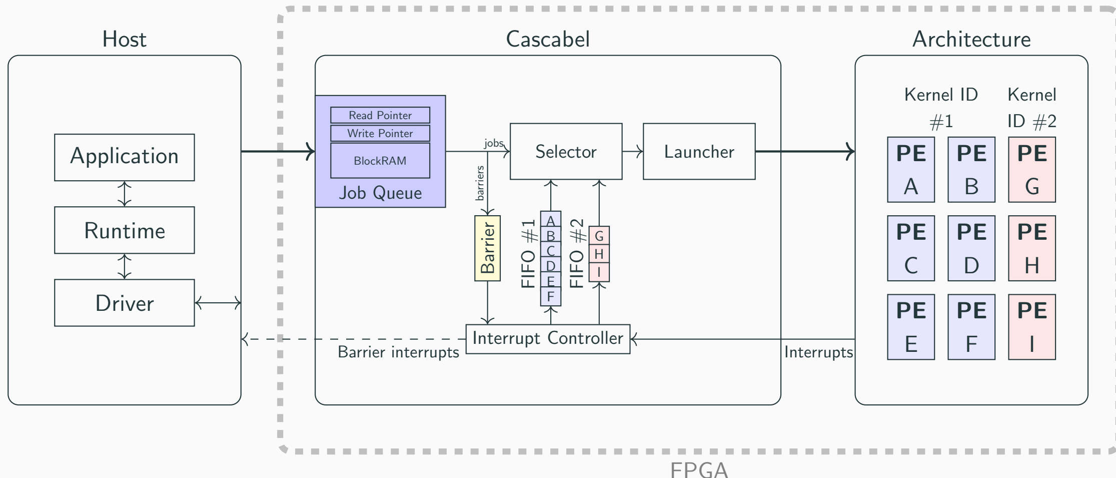
TaPaSCo Cascabel - Hardware Offloading



TaPaSCo Cascabel - Hardware Offloading



TaPaSCo Cascabel - Hardware Offloading



TaPaSCo Cascabel - Current Limitations

- No advanced scheduling, just FIFO
- No on-chip job creation
- No inter-PE communication

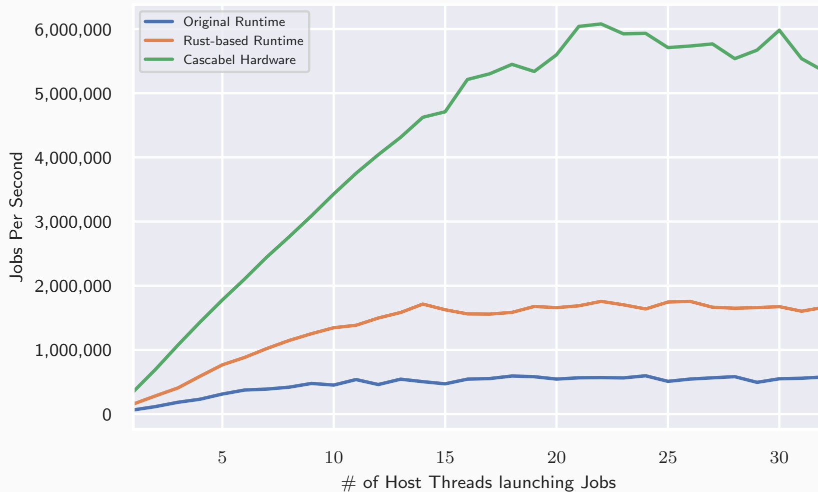
Evaluation

- Throughput and Latency
- Array pipeline application
- Resource overhead

Evaluation

- Throughput and Latency
- Array pipeline application
- Resource overhead
- System:
  - FPGA: Xilinx Alveo U280
  - Host: AMD EPYC 7351P (16 cores)
  - Results for a second platform in paper

Evaluation - Throughput



Evaluation - Launch Latency

Cycles @ 450 MHz	Original Runtime [µs]	Rust-based Runtime [µs]	Cascabel Hardware [µs]
1	19.40	8.54	13.92
16	21.75	8.56	13.94
8192	22.26	10.33	12.64
2^{22}	158.23	45.46	49.15

Evaluation - Array Pipeline Application

- Application with data dependencies between jobs

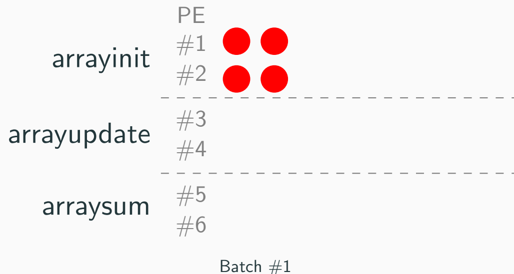
arrayinit

arrayupdate

arraysum

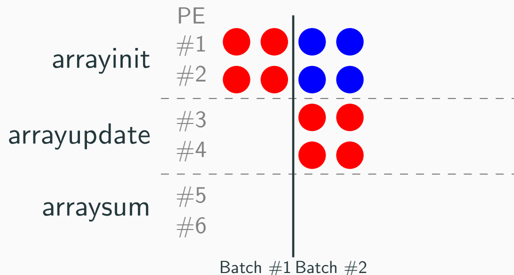
Evaluation - Array Pipeline Application

- Application with data dependencies between jobs



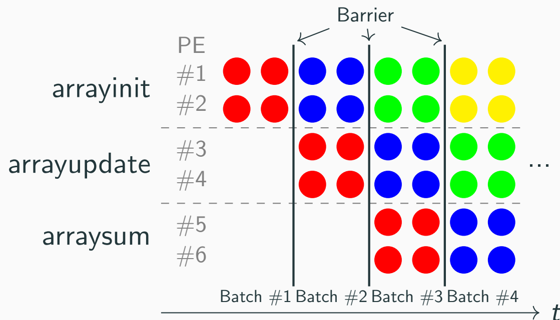
Evaluation - Array Pipeline Application

- Application with data dependencies between jobs



Evaluation - Array Pipeline Application

- Application with data dependencies between jobs



Evaluation - Array Pipeline Application

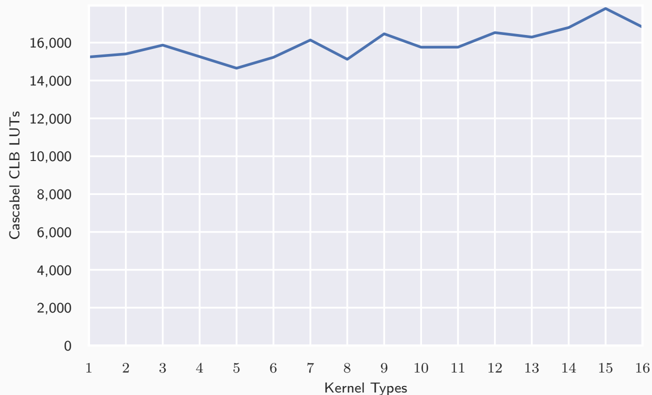
- Application with data dependencies between jobs
- Execution time for 100,000 iterations
- Single thread:
  - Rust-based runtime: 16.1 seconds
  - Cascabel: 6.25 seconds
- 8 threads:
  - Rust-based runtime: 5.7 seconds
  - Cascabel: 4.42 seconds
- Up to 2.8x speedup

Evaluation - Resource Utilization

- Depends on PE configuration
  - FIFOs for every Kernel type
- Always below 2% of available resources

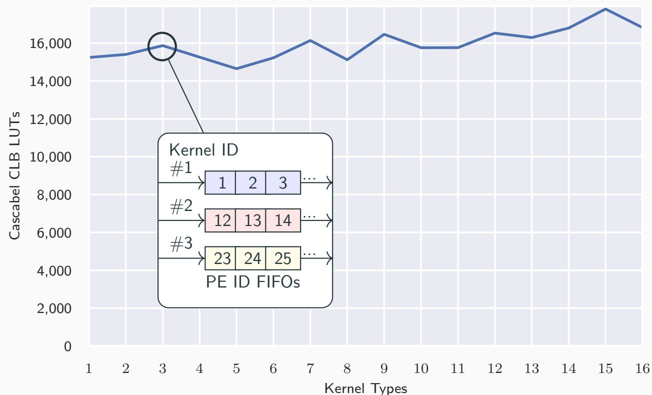
Evaluation - Resource Utilization

- Increasing number of different kernels, 32 PEs in total



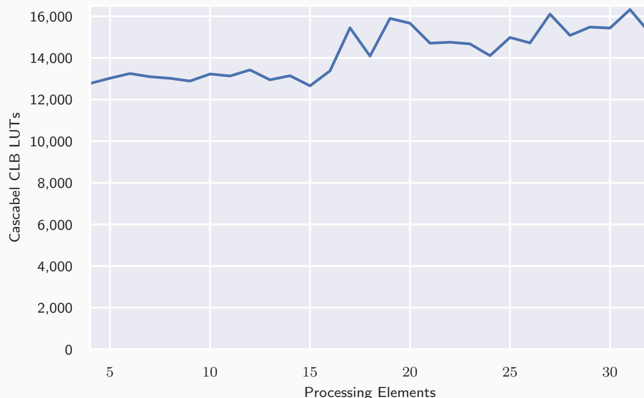
Evaluation - Resource Utilization

- Increasing number of different kernels, 32 PEs in total



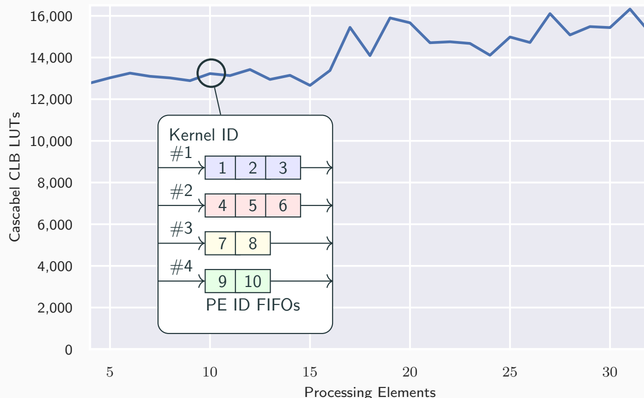
Evaluation - Resource Utilization

- Increasing total number of PEs, four different kernels



Evaluation - Resource Utilization

- Increasing total number of PEs, four different kernels



Summary and Outlook

- Improved software based runtime
- Cascabel: hardware offloading of launching functionality
- Low resource overhead
- Up to 6x increase in throughput

- Future work:
  - Improve in-hardware scheduling of jobs
  - On-chip job launches

Interested?

TaPaSCo is available on Github

github.com/esa-tu-darmstadt/tapasco