OctoRay: Framework for Scalable FPGA Cluster Acceleration of Python Big Data Applications

Zaid Al-Ars[1], Jakoba Petri-Koenig[1], Luc Dierick[1], Peter Hofstee[2], Joost Hoozemans[3]

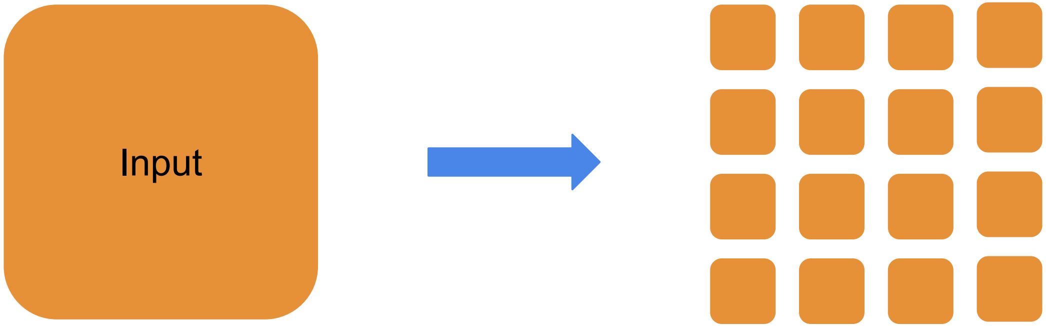
[1] Delft University of Technology
[2] IBM Infrastructure
[3] Voltron Data

Paper objectives

“To be able to scale out a data analytics task to 100s of FPGAs using Python transparently and efficiently”

Big data scalability

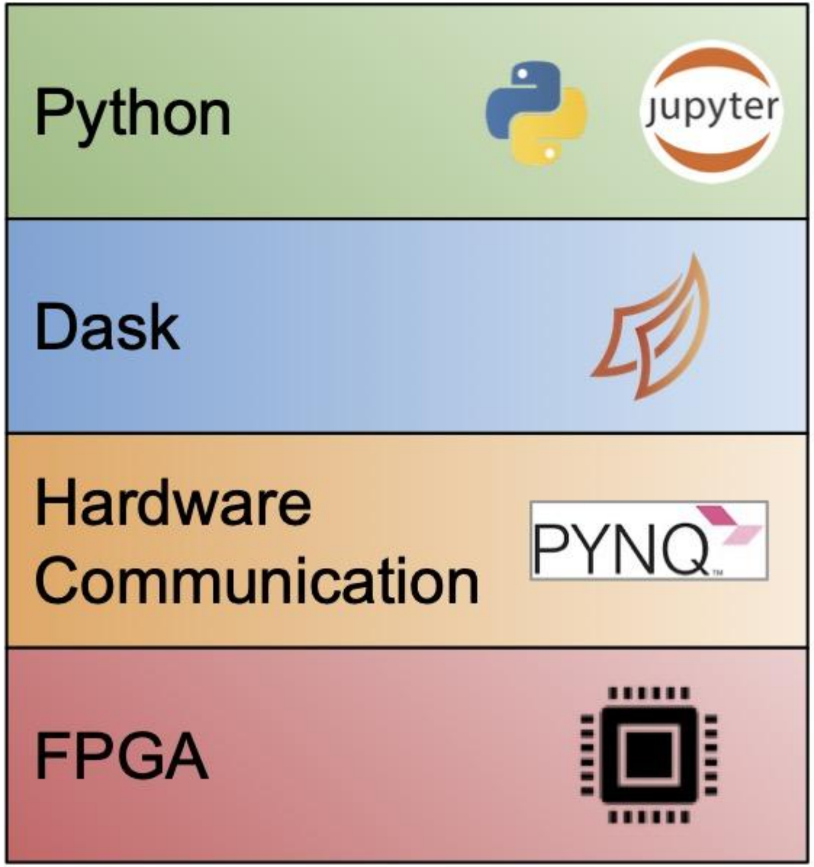
- **Big data scalability => input data parallelism** Concurrent execution of the same task on multiple computing cores/nodes on different subsets of the data.



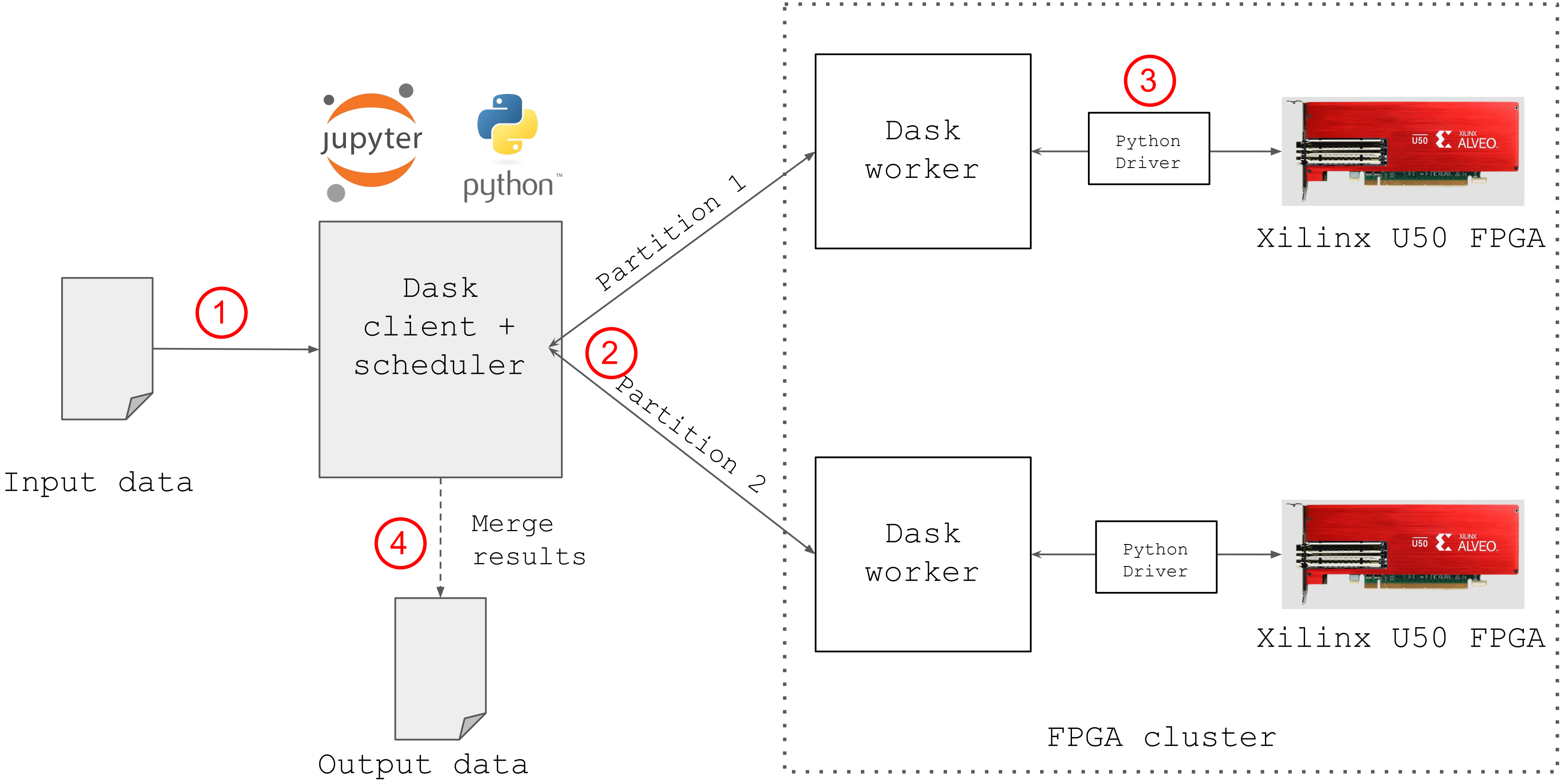
- **Advantages**:
  - Lower costs
  - Reliability
  - Flexibility

Architecture: SW stack

- Big data SW stack
  - Python
  - Dask
- Integration with common HW tools
  - Pynq
  - FPGA

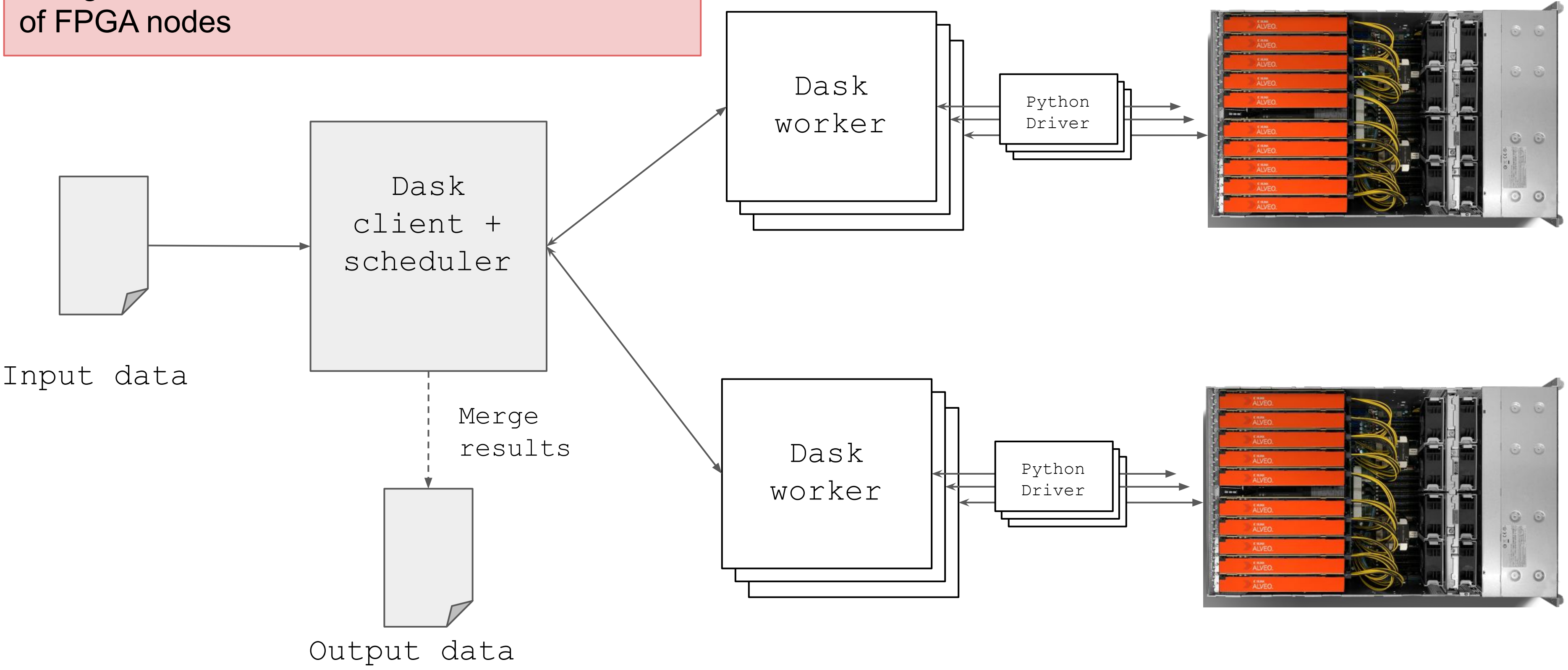


Architecture: scalability

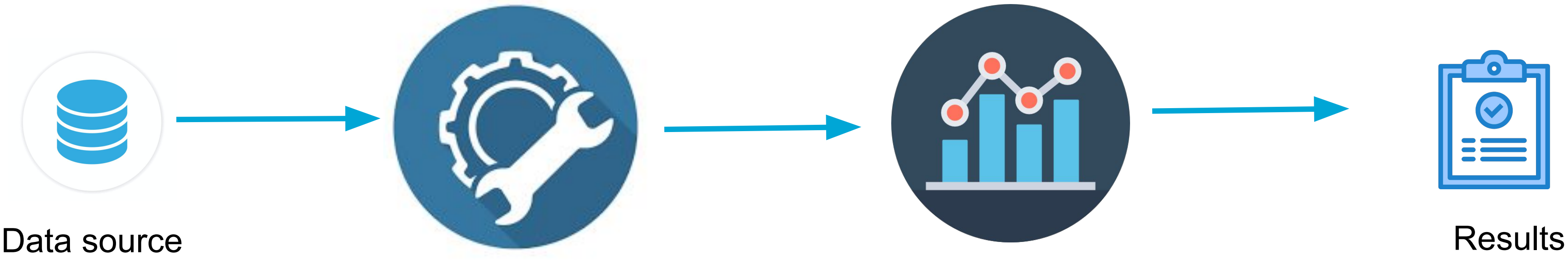


Architecture: scalability

Using this architecture, we can scale to 10s or of FPGA nodes



Data analytics pipeline



STAGE 1
DATA PREPARATION
Eg: Data (de)compression

STAGE 2
DATA ANALYTICS
Eg: Machine learning

Acceleration of **both** stages possible with OctoRay

```python
    data_split.append(total[start:]) #Last partition

# Scatter the data to the workers before calling run_on_worker on the workers
distributed_data = client.scatter(data_split)
futures = client.map(run_on_worker, distributed_data, range(num_of_workers))
results = client.gather(futures)
print("Received data from workers")

# Reorder the response based on original input order
results.sort(key = lambda result: result['index'])
compression_time = max([r['time'] for r in results])

print("Writing combined (compressed) data to " + FINAL_COMPRESSED_FILE)
with open(FINAL_COMPRESSED_FILE, "wb") as f:
    for result in results:
        f.write(result['data'])


t1 = time.time()
print("MAX COMPRESSION TIME (in s): ", compression_time)
print("TOTAL EXECUTION TIME (in s): ", t1 - t0)
```

```
Splitting input file into 2 chunk(s)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
Received data from workers
Writing combined (compressed) data to compressed.gz
MAX COMPRESSION TIME (in s):  0.7149055004119873
TOTAL EXECUTION TIME (in s):  14.521609544754028
```

In [ ]:

```python
FILE_COPY = FILE_TO_BE_COMPRESSED + ".copy"
COMMAND_TO_RUN = "gzip -dc " + FINAL_COMPRESSED_FILE + " > " + FILE_COPY
print("Extracting", FINAL_COMPRESSED_FILE, "using command: ")
print(COMMAND_TO_RUN)
os.system(COMMAND_TO_RUN)
print("Comparing", FILE_COPY, "to", FILE_TO_BE_COMPRESSED)
with open(FILE_TO_BE_COMPRESSED, 'rb') as f1:
    with open(FILE_COPY, 'rb') as f2:
        if f1.read() == f2.read():
            print("Validation succeeded !!")
        else:
            print("Validation failed !!")
```

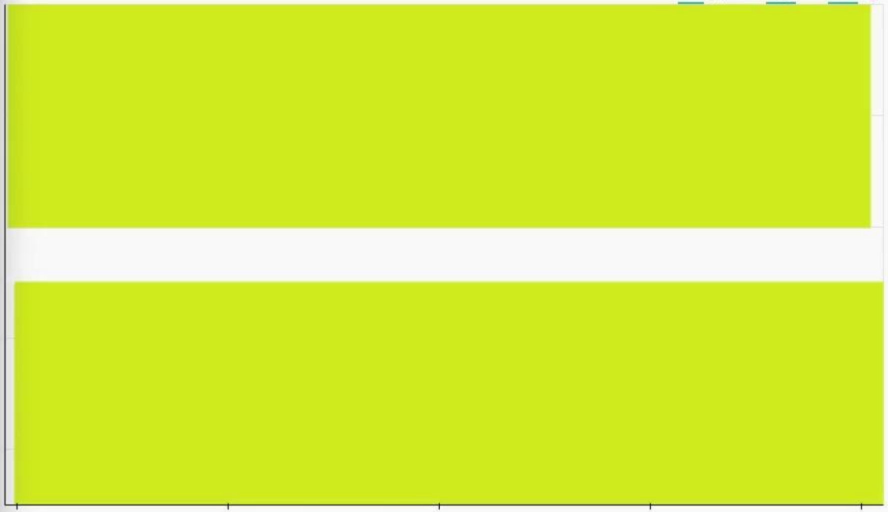
In [ ]:

Status | Workers | Tasks | System | Profile | Graph | Info

Bytes stored: 1.66 GB

Tasks Processing

Task Stream

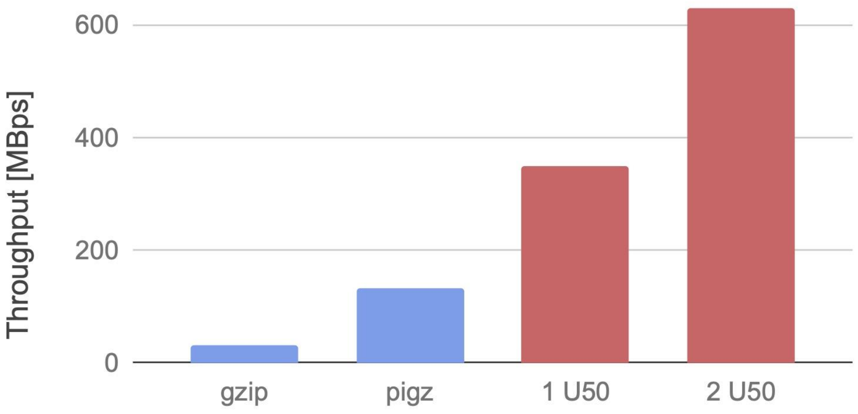


Progress -- total: 4, in-memory: 4, processing: 0, waiting: 0, erred: 0

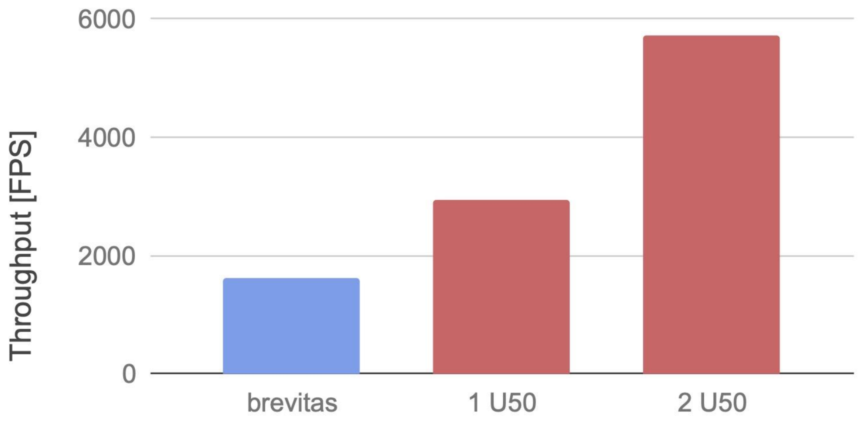
bytes 2 / 2

run_on_worker 2 / 2

Results: pipeline stages on FPGA

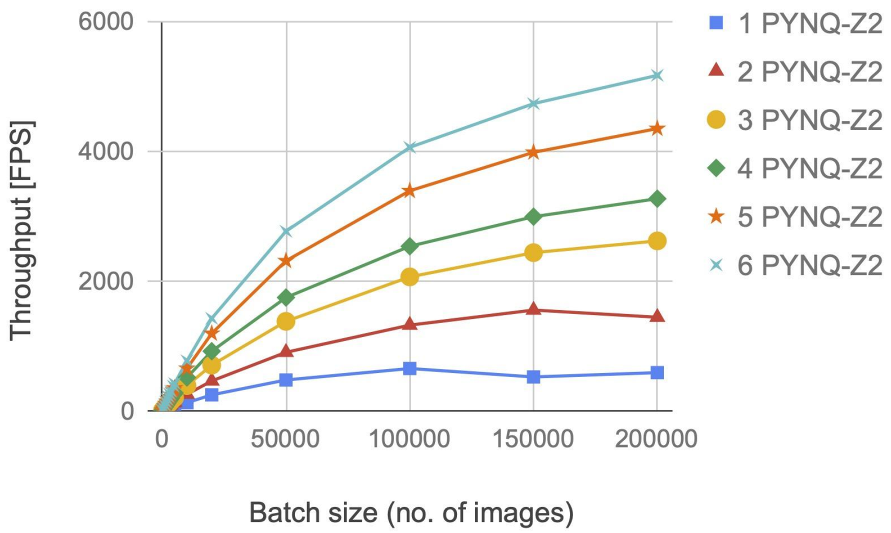


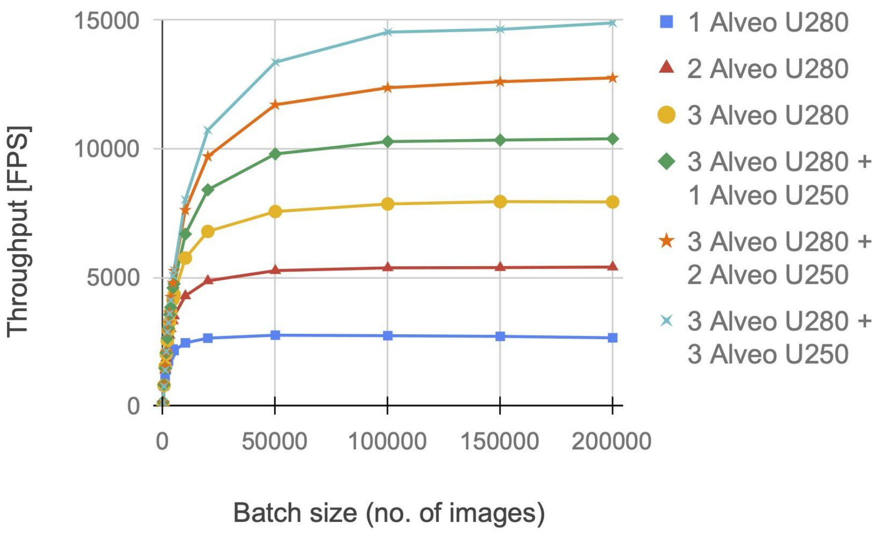
Data (de)compression



CNN inference

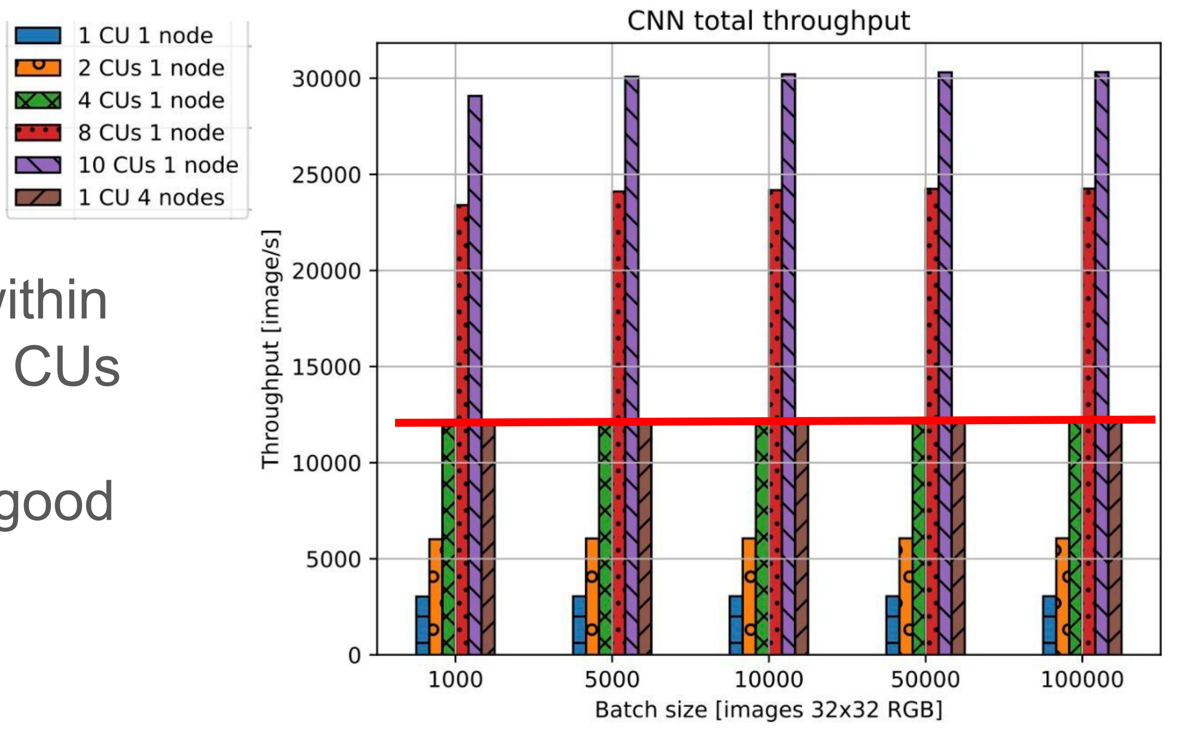
Results: flexible scalability on various FPGA





- **Scalability on various FPGA platforms: Pynq/ Alveo**
- **CNN throughput for increasing batch size**

Results: inter vs intra FPGA scalability



- Enables scalability within FPGA using multiple CUs
- Scalability is just as good as between FPGAs

Easy scalability both on an FPGA and between FPGAs

Did we achieve objectives?

“To be able to scale out a data analytics task to 100s of FPGAs using Python transparently and efficiently”

Yes .. partly

- Up to 10 boards & up to 10 CUs per board
- OctoRay works .. but end-to-end integration still challenging (tooling is still HW centric)

Conclusions

- OctoRay’s multi-FPGA setup provides speedup for both stages of a data analytics pipeline:
  - Linear scalability for 10s of FPGAs
  - Compression: 2 FPGAs 4x faster than SW
  - Neural network: 2 FPGAs 12x faster than SW
- OctoRay supports:
  - Various **infrastructure setups**: Multi-FPGA hosts or single-FPGA hosts
  - Various **types of accelerators**: Vitis Library, FINN, PYNQ and custom kernels
  - Various **hardware platforms**: Pynq-Z1, AWS-F1, Nimbix Cloud, in-house servers

GitHub repo reference

The complete code for this project can be found at
https://github.com/abs-tudelft/octoray

Acknowledgment

We would like to thank Xilinx for donating the U50 Alveo FPGA board for the purpose of this project, and for providing access to the ETH XACC cluster.