A Framework for the Automatic Generation of FPGA-based Near-Data Processing Accelerators in Smart Storage Systems
Authors

Lukas Weber  
ESA, TU Darmstadt

Lukas Sommer  
ESA, TU Darmstadt

Leonardo Solis-Vasquez  
ESA, TU Darmstadt

Andreas Koch  
ESA, TU Darmstadt

Tobias Vincon  
DBLab, Reutlingen University

Christian Knödler  
DBLab, Reutlingen University

Arthur Bernhardt  
DBLab, Reutlingen University

Ilia Petrov  
DBLab, Reutlingen University
Near-Data Processing

Data-to-Code
Near-Data Processing

Data-to-Code

Code-to-Data
Near-Data Processing (NDP)

- Computation **can** happen close to the data
- Intermediate compatibility layers **can** be removed
- Fine-granular control over storage & compute resources
Types of NDP

Software-based

- Removes compatibility layers
- Exploits on-device compute resources
Types of NDP

Software-based

- Removes compatibility layers
- Exploits on-device compute resources

Hardware

- Fully moves computational load to logic resources
- Less software-interaction
- Potentially even faster
Types of NDP

Software-based

• Removes compatibility layers
• Exploits on-device compute resources

Hardware

• Fully moves computational load to logic resources
• Less software-interaction
• Potentially even faster

Hybrid

• Exploits on-device compute resources
• Exploits available logic resources
• Software-controlled
• Hardware-accelerated
Downsides of Hybrid/Hardware NDP

• Requires device-specific knowledge
• Requires background in hardware design
• Typically tedious and time-consuming
  • Long development- & debug-cycles
Application

• General-Purpose Key-Value Store Operations:
  • GET: Retrieve the value of a single key
  • SCAN: Retrieve all KV-pairs matching some predicate

• Requests issued by a host-CPU
• Cosmos+ OpenSSD as Smart Storage Device
• Computation happens on-device
Prior Development Flow
Our Development Flow

1. nKV/RocksDB-based Database Application
2. Hardware Generation
3. Integration
4. Testing & Deployment
Generated Accelerators
Advantages

• No knowledge of hardware design necessary
• Processing Elements (PE) and Interfaces are automatically generated
• Integration becomes relatively easy task
  • Similar interfaces for different generated PEs
• Overall: Faster & easier
Further Additions

- Multi-Stage Filtering
  - More complex filtering predicates
  - Chaining multiple Filtering Units
- Disregard unimportant data to save logic resources
- Generate from simple annotated C-Code

```c
/* @autogen define parser Point3DTO2D with chunksize = 32, input = Point3D, output = Point2D, mapping = { output.x = input.y, output.y = input.z } */
typedef struct { uint32_t x, y, z; } Point3D;
typedef struct { uint32_t x, y; } Point2D;
```
Disadvantages

• Performance?
• Hardware-Utilization?
Disadvantages

- Performance
- Hardware-Utilization

Short Answer:
- Slightly decreased performance
- Slightly increased utilization
Evaluation – Performance

(a) GET Operation

(b) SCAN Operation

Execution times of GET & SCAN operations
Evaluation – Hardware Utilization

Slice utilization of generated accelerators
Evaluation – Multi-Stage Filtering

Slice utilization using multiple Filtering Stages
Evaluation – Multi-Stage Filtering

Slice utilization using multiple Filtering Stages
Conclusion

- Similar performance & utilization in comparison to prior work
- Significantly decreased development overhead
- More functionality and flexibility