Accessible Near-Storage Computing with FPGAs

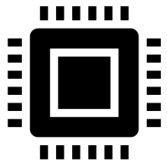
Robert Schmid, Max Plauth, Lukas Wenzel, Felix Eberhardt, Andreas Polze
Professorship for Operating Systems and Middleware, Hasso-Plattner-Institute

Fifteenth European Conference on Computer Systems (EuroSys '20), April 27–30, 2020

Near-Data Computing for Data-Intensive Applications

Bandwidth of interconnects and memory buses limits the scalability of data-intensive applications

Performing computations close to the data source reduces data movements in the system



Trend towards heterogenous system architectures:

Computing DRAM, Smart SSDs, Smart NICs, …

Programming Interfaces for Near-Storage Compute

- Near-Storage Computing: SSDs with compute capabilities

- Employing near-storage compute for database acceleration
  - Smart SSDs (Do et al., 2013)
  - Ibex (Woods et al., 2013)

- What are suitable programming interfaces for near-storage compute?
  - Insider (Ruan et al., 2019): Virtual file abstraction

Hardware Testbed

Nallatech N250S

XILINX ALL PROGRAMMABLE™

CAPI SNAP

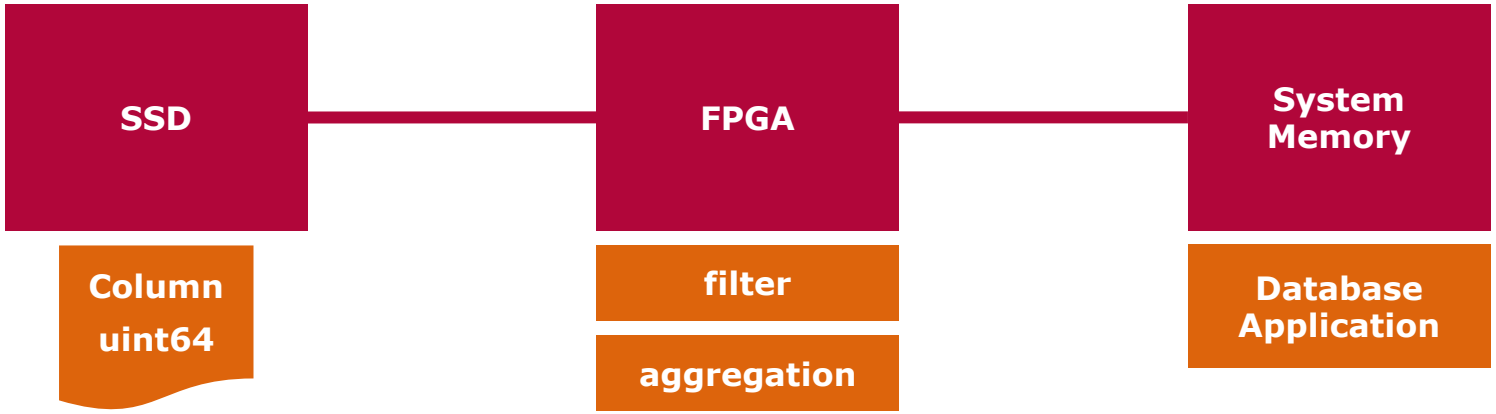
OpenPOWER

IBM

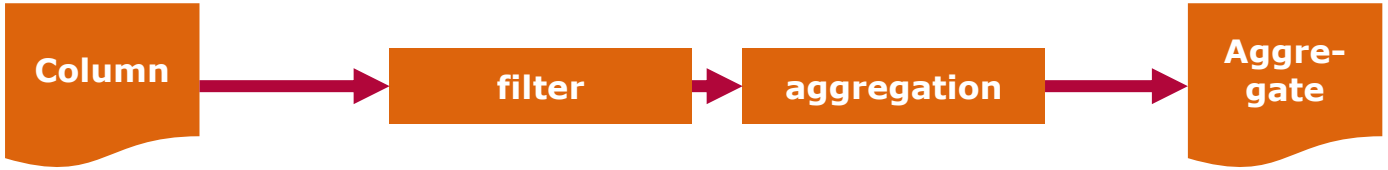
Accessible Near-Storage Computing with FPGAs

Robert Schmid
EuroSys '20
April 27–30, 2020

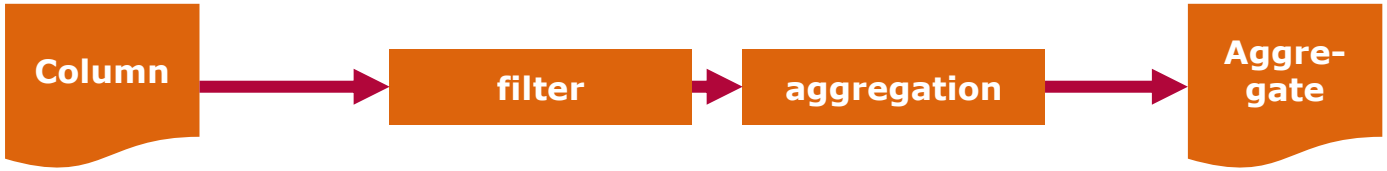
Scenario



Near-Storage Compute Graph:



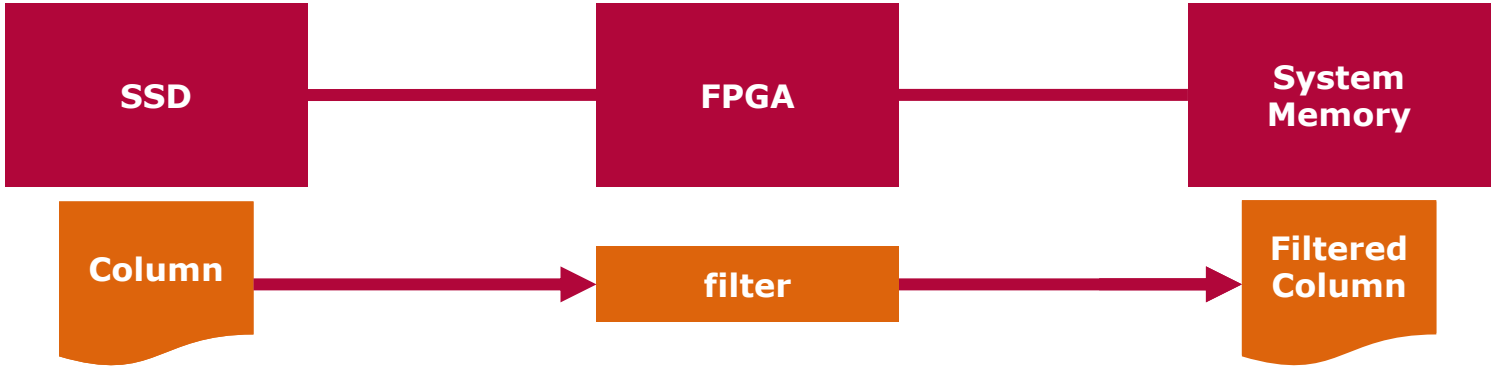
Introducing Metal FS



- Metal FS is a framework for orchestrating near-storage compute

- Re-uses Unix Operating System concepts:
  - Data items (streams of bytes): Files
  - Computation kernels (‘Operators’): Executables
  - Composition primitives: Pipe and Redirection Shell-Operators

Metal FS: Files and Operators



Metal FS Demo

```
/ $ metal-cat -p /mtl/files/column_uint64 | filter --lower-bound 0 --upper-boun
d 2 | pv > /dev/null
STREAM  BYTES TRANSFERRED  ACTIVE CYCLES  DATA WAIT      CONSUMER WAIT  MiB/s
output  1073741824         16777216   7%  193520489 78%  37713732  15%  1082.31
 512MiB 0:00:01 [ 495MiB/s] [ <=>                                            ]
/ $
```

Accessible Near-Storage Computing with FPGAs

Robert Schmid
EuroSys '20
April 27–30, 2020

Metal FS Core Components

- Highlighted Aspects
  - Operator definition
  - Detecting Unix Pipe expressions

- More features not covered in this presentation
  - Manifest-driven FPGA image build process
  - Hybrid filesystem implementation
  - Package manger for distributing operator source code
  - Docker-based hardware and software development environment
  - Use as a library, C++ API

Operators as FPGA Computation Primitives

- Data Stream *Operators* encapsulate computations
- Defined in HLS or VHDL/Verilog
- Operate on untyped byte streams
- Parameterizable at runtime

- HLS Example Operator:

```
void my_operator(mtl_stream &in, mtl_stream &out) {
    mtl_stream_element element;
    do {
        element = in.read();
        // TODO: Transform element.data
        out.write(element);
    } while (!element.last);
}
```

Metal FS: Detecting Unix Pipe Expressions

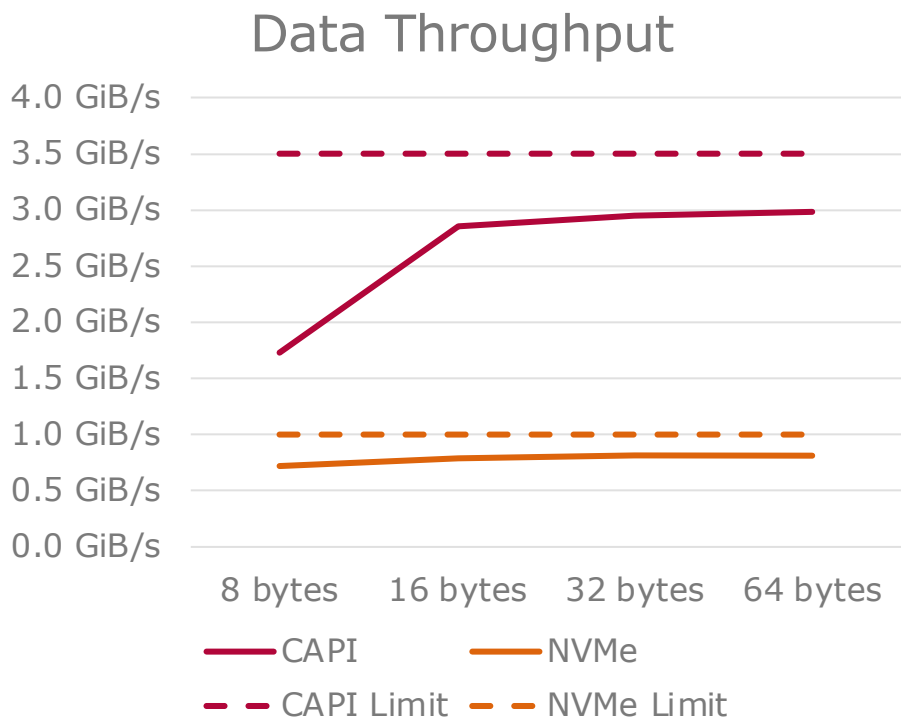
- Metal FS runs entirely in user-space
- Operators are represented by proxy executables in the file system

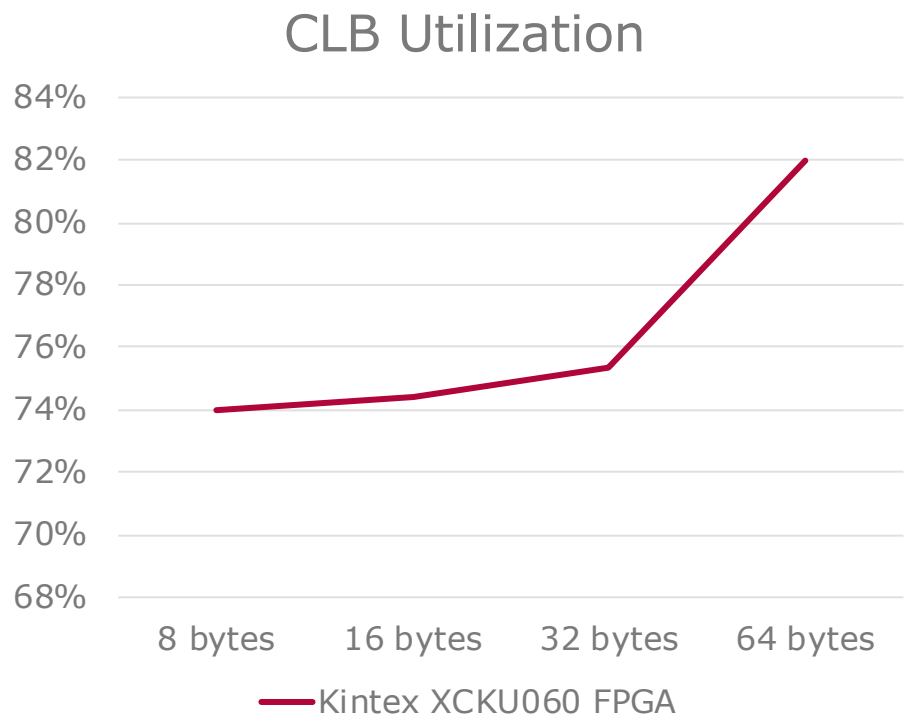
- Detect composition of proxy executables by using ‘reflection’
  - Scan Linux’ `procfs` for matching `stdin`, `stdout` file descriptors
  - `/proc/<pid>/fd/0,1` ➔ `pipe:[<id>]`

- FUSE filesystem process collects information from all running proxy processes and invokes FPGA processing

Evaluation

- CAPI/NVMe Throughput and FPGA Resource Utilization
  - FPGA Image with 4 Passthrough-Operators
  - Different Stream Word Widths





Accessible Near-Storage Computing with FPGAs

Robert Schmid
EuroSys '20
April 27–30, 2020
Chart **11**

Conclusion

- Existing operating system interfaces are suitable for near-storage compute
- Metal FS attempts to improve accessibility of near-storage compute on multiple levels
  - Orchestration Interface, Development Environment, Reusable Operators
- Outlook
  - Integration in real-world application scenarios
  - Further evaluate the tradeoff for our abstraction:
    Exposing only necessary hardware specifics to maximize portability across different hardware architectures

Thank you!

Metal FS Documentation and Source Code

https://metalfs.github.io

https://github.com/osmhpi/metalfs

Thanks!

To the IBM Lab Team in Böblingen: Jörg-Stephan Vogt, Frank Haverkamp, Sven Boekholt, Thomas Fuchs, Sven Peyer and Nicolas Mäding as well as the CAPI SNAP Team: Bruno Mesnet and Alexandre Castellane

Contact

Robert Schmid
robert.schmid@hpi.uni-potsdam.de