BPF
on Computational Storage

Niclas Hedam | nhed@itu.dk

DASYA
Data-Intensive Systems and Applications

IT UNIVERSITY OF COPENHAGEN

Outline

- Introduction (problem, approach, contributions)
- Background
- Architecture
- Mechanisms
- Use-cases
- Conclusion

Introduction

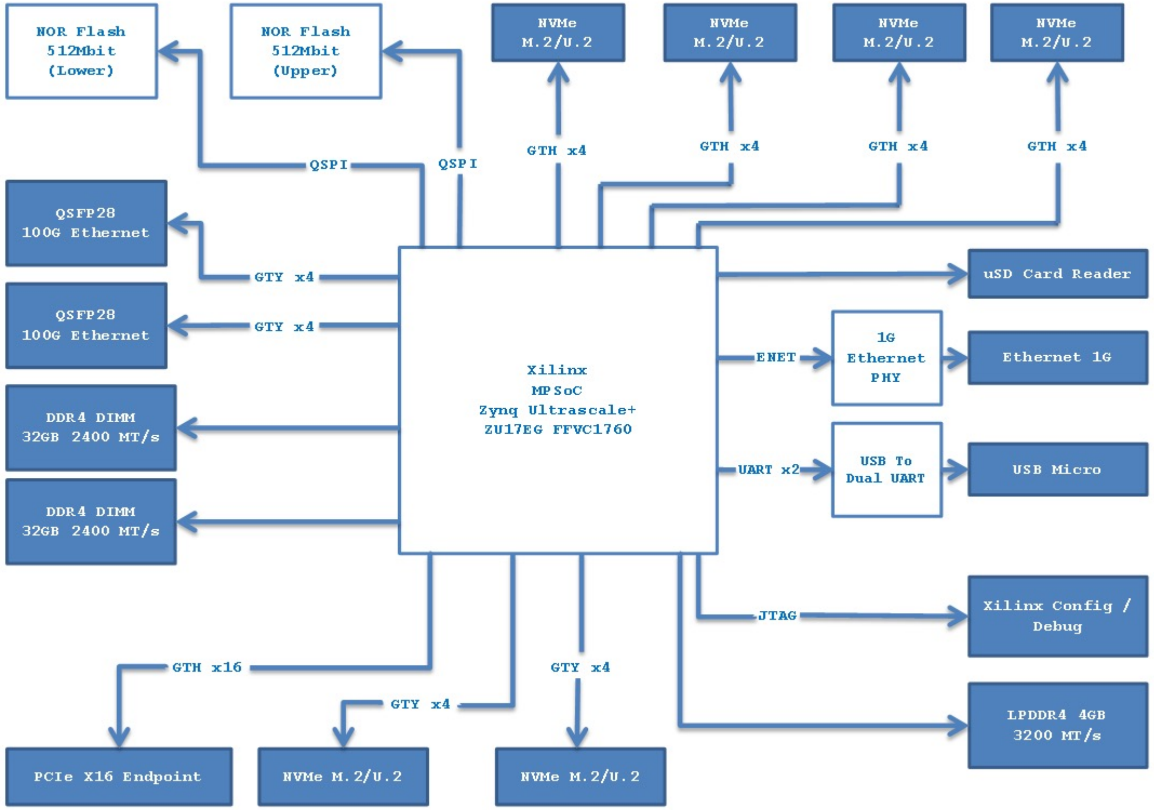
- We are using BPF to orchestrate function calls on computational storage devices (CSDs).
- Our approach is experimental, contributing to the required mechanisms, in the context of use-cases provided by the DAPHNE project.
- So far our contributions include:
    - Architecture for BPF on CSD
    - Extension of the uBPF virtual machine to support registered functions on CSD
    - XDMA mechanisms and BPF execution in Eid-Hermes (QEMU)
    - DAPHNE use-cases

Background: BPF

- BPF is a assembly like format bytecode
    - Support in clang (and gcc) for C code
        - Not so much for C++
    - **Vendor neutral ISA**
        - Possible execution at various steps of I/O path: kernel, NIC, CSD
        - **But:**
            - Verification in kernel, NIC or CSD are different, e.g., focus on confidentiality for in-kernel execution vs. focus on integrity for CSD verification, bytecode vs. C-level analysis
                - Verification on CSD is an open-problem!
            - No ARM64 JIT for BPF
                - Neither in-kernel or in uBPF!
            - Different VMs in kernel, NIC or CSD.
                - uBPF for user-space VM on CSD.
                    - Who is in charge of uBPF? Pull request from Eideticom blocked for weeks…
                    - Universal uBPF vs. CSD-specific BPF VM?

Background: CSD

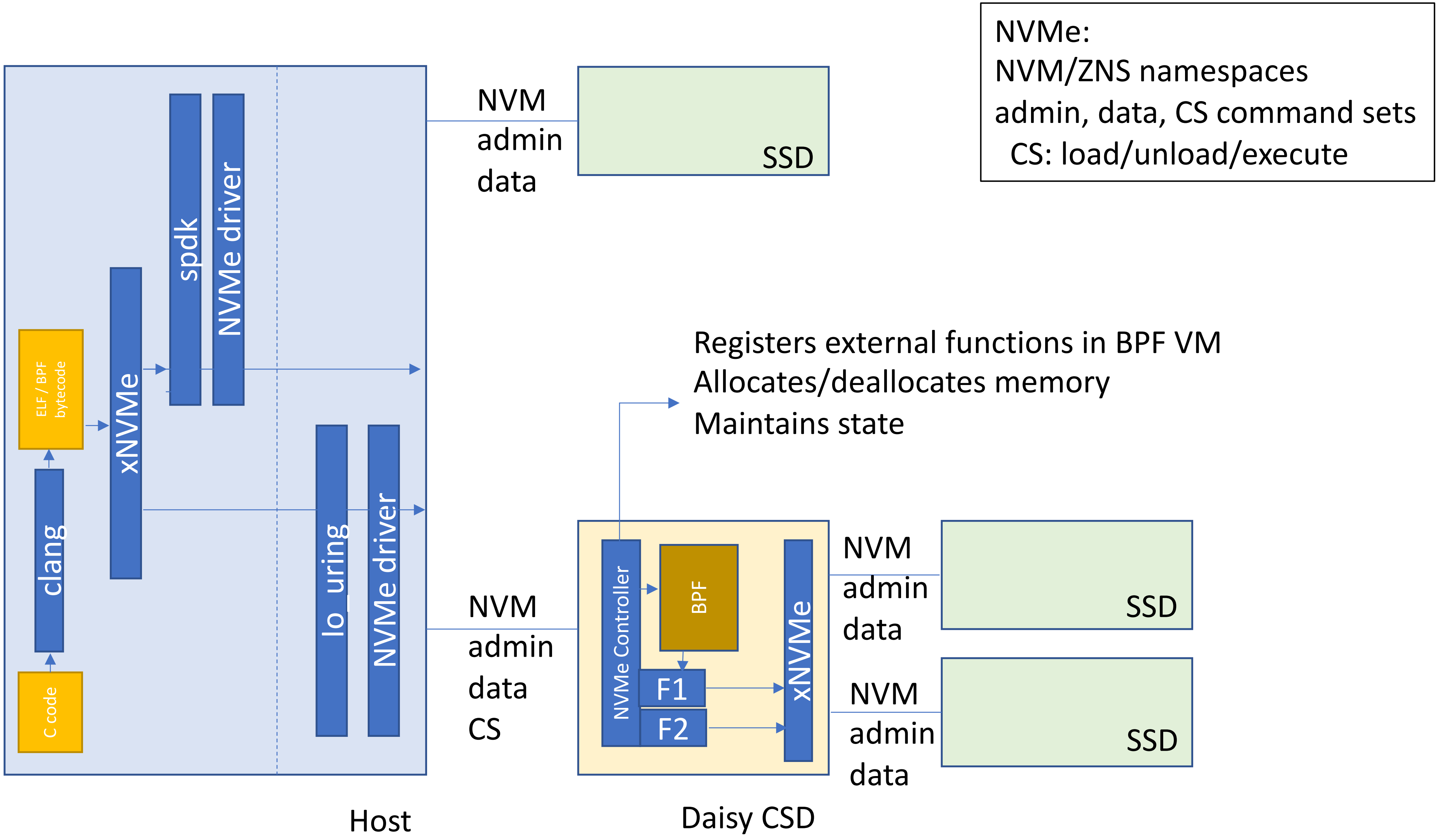
- We will experiment on the Daisy OpenSSD.
- Specs
  - 1.5 GHz Quad-core ARM main processor
  - 0.6 GHz Dual-core ARM real time processor
  - 6 NVMe expansion slots



Architecture

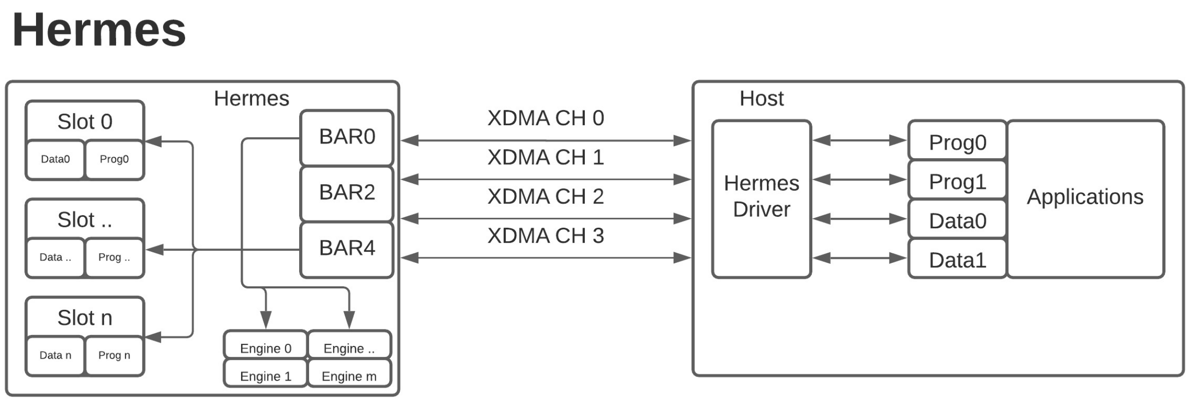
- Programs are loaded into a BPF virtual machine on CSD.
  - We use a modified version of the uBPF VM by iovisor.

- VMs are managed by a *loader*, which is in charge of loading programs into the VM and managing the state and interconnect.
  - The loader is also responsible for exposing *registered functions*.
  - We are building this loader ourselves at ITU.



NVMe:
NVM/ZNS namespaces
admin, data, CS command sets
CS: load/unload/execute
NVM
admin
data
SSD
ELF / BPF bytecode
C code
clang
xNVMe
spdk
NVMe driver
Io_uring
NVMe driver
Registers external functions in BPF VM
Allocates/deallocates memory
Maintains state
NVM
admin
data
CS
NVMe Controller
BPF
F1
F2
xNVMe
NVM
admin
data
SSD
NVM
admin
data
SSD
Host
Daisy CSD

Mechanisms



- uBPF
  - Added support for registered functions in VM.

- Hermes
  - A BPF accelerator built for AWS F1 instances.
  - Hermes supports loading, unloading and executing programs.
  - Device information and data is stored on BARs (Base Address Registers)
    - The host discover the capabilities of the device from the BAR0.
    - The host uses a driver to offload a BPF program to the BAR2, and then stores it on BAR4.
    - The host executes the BPF program by writing the execute command to BAR0.
      - The device will do an interrupt once the execution is done.
    - The host initiates a DMA transfer to transfer the resulting data back to the host
  - However, Hermes does not support external/registered functions.
    - Thus you can only run what is embedded in the BPF ELF binary itself.
    - The use-cases are therefore limited to very simple programs.

Mechanisms

- xNVME
  - Our prototype loader will use xNVME to access SSD from the CSD controller.
    - This will make it easy to read from many different storage systems using a single unified interface.
  - Furthermore, we are working on introducing BPF to xNVME, such that xNVME can be used to offload BPF programs to Hermes and our prototype.
    - The interface will be like Hermes: load/unload/execute.
  - Open issues relates to exposing the registered functions against which BPF code should be written.
    - **The device may not be attached at compile-time (!)**

Use-cases

DAPHNE

- DAPHNE is an infrastructure for Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning funded by EU.

- We are starting with reading CSV.
  - Baseline on host on top of file system
  - First version of BPF-based code, where file access is offloaded
    - Naming issues!
  - Next: offloading parsing and structuring of the data.
    - Many issues (see next slides)

Use-cases

DAPHNE

- Finding a balance is the main challenge.
  - DAPHNE is written in C++, which is not easily reducible to BPF code.
    - DAPHNE also uses templating, which cannot be reduced.
  - DAPHNE data structures are scattered in memory, so we'll have to move data to the data buffer before transferring it back to the host.
  - Ideally, we should support the the data structures, but this may be unreasonable due to the number of different data structures in applications.

- Potential solution is reducing data types to bit-lengths and let the host cast the data to the right types.
  - It also introduces the design question on whether the resulting data should be of a structure that the application already implements or a buffer that the application should morph/convert.

DAPHNEs perspective:

long	double	uint

CSD perspective:

64b	64b	32b

Conclusion

- Implementation goals stemming from Daphne
  1. push down CSV parsing and structuring via BPF.
     - Issues includes finding the balance in DAPHNE.
       - Create simplified data structures to push to CSD – in C, not C++.
       - Define a structure for the data buffer, that both the runtime and CSD conforms to.
  2. Run on Daisy OpenSSD as CSD
     - This allows us to experiment and collect performance metrics.
     - The open issues include standardizing the interconnect (how data and programs are transferred), implementing xNVME in the prototype and implementing the BPF interface in xNVMe.
- Paper on BPF for computational storage: A Reality Check with DASYA team
  - Analysis of opportunities, challenges and roadblocks.

Thanks! Questions?

Niclas Hedam | nhed@itu.dk

DASYA
Data-Intensive Systems and Applications

IT UNIVERSITY OF COPENHAGEN