ZUSE INSTITUTE BERLIN

Modernizing Legacy Codes for Next-Generation Storage Infrastructures: A Case Study of PALM on Intel DAOS

Steffen Christgau and Thomas Steinke
Zuse Institute Berlin (ZIB)

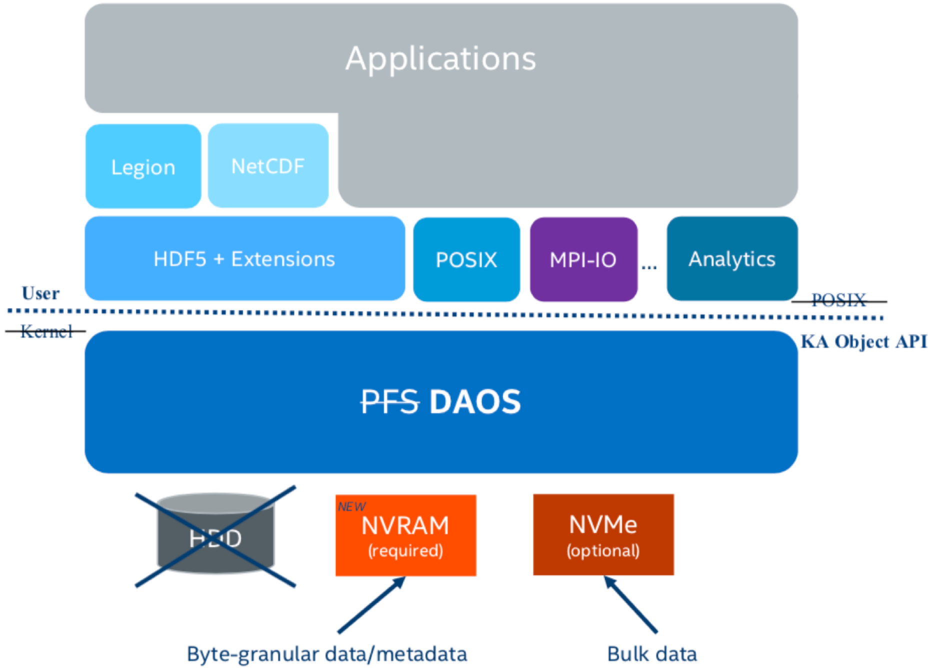
DAOS User Group, November 20, 2019

Motivation

- Evaluation of future scalable storage solutions for next-gen system architectures
  - Hardware: DCPMM + NVMe SSDs
  - Software: DAOS storage software stack

- DAOS with high-level libraries: netCDF, HDF5, …, MPI (,POSIX)

- Start with ”simple” use case: Checkpointing

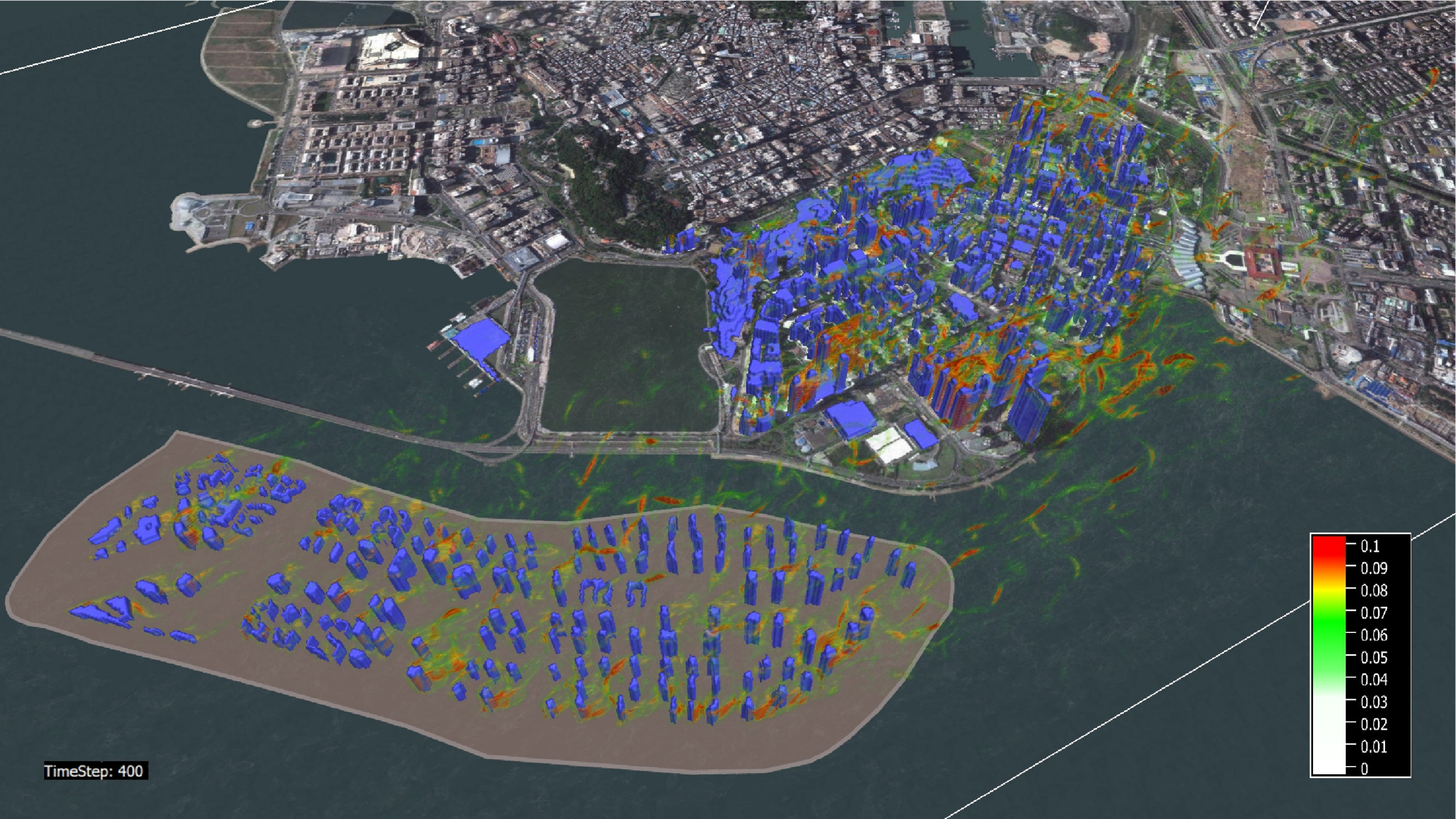


Source: A. Dilger DAOS: Scale-out Object Storage for NVRAM, Dagstuhl Workshop, May 2017

About PALM

Parallelized Large-Eddy Simulation Model

- Developed by IMUK @ Univ. Hannover
- Computes turbulent air-flows, solves incompressible Navier-Stokes equations
- 3D compute domain, typical dimensions *O(1k) x O(1k) x O(100)*
- Fortran 2003 (95) code base, GPL
  - since 1997, 200k+ lines of code, lots of modules
- MPI/OpenMP parallelization
- large memory footprint
- netCDF for data output


0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
TimeStep: 400

Checkpointing in PALM: Initial Version

- Motivation: save the application state when job reaches wall time limit … multiple restarts in job chains
- Fortran unformatted I/O
- Checkpoint creation: write name and raw data to hard-coded unit

```
CALL wrd_write_string( ’topography’ )
WRITE ( 14 ) topography
```

- Restore: read name from other hard-coded unit + large select statement

```
SELECT CASE ( restart_string(1:length) ) ...
CASE ( ’topography’ ) READ ( 13 ) topography
```

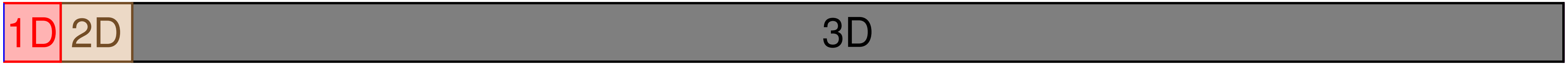
Checkpointing in PALM: Challenges

- Raw binary data: hard to postprocess (useful for debugging)
- Unformatted Fortran IO: "hidden" additional small writes
- One file per MPI process(!)
- Support of variable task/thread configuration
  → PE-independence results in complicated restore code
- interface to DAOS? … Via POSIX!?
- **no abstraction**
- **no expressed parallelism**

Which Data are Checkpointed?

- Not all application data are written to checkpoint file(s)
- Typical composition (simplified) for 4096 × 4096 × 256 compute domain

Type	Count	Data
scalars	192	907 B
1D arrays	57	1.00 GB
2D arrays	21	1.25 GB
3D arrays	9	258.00 GB
4D arrays	1	806 KB
total	**280**	**261 GB**



New Checkpointing Implementation

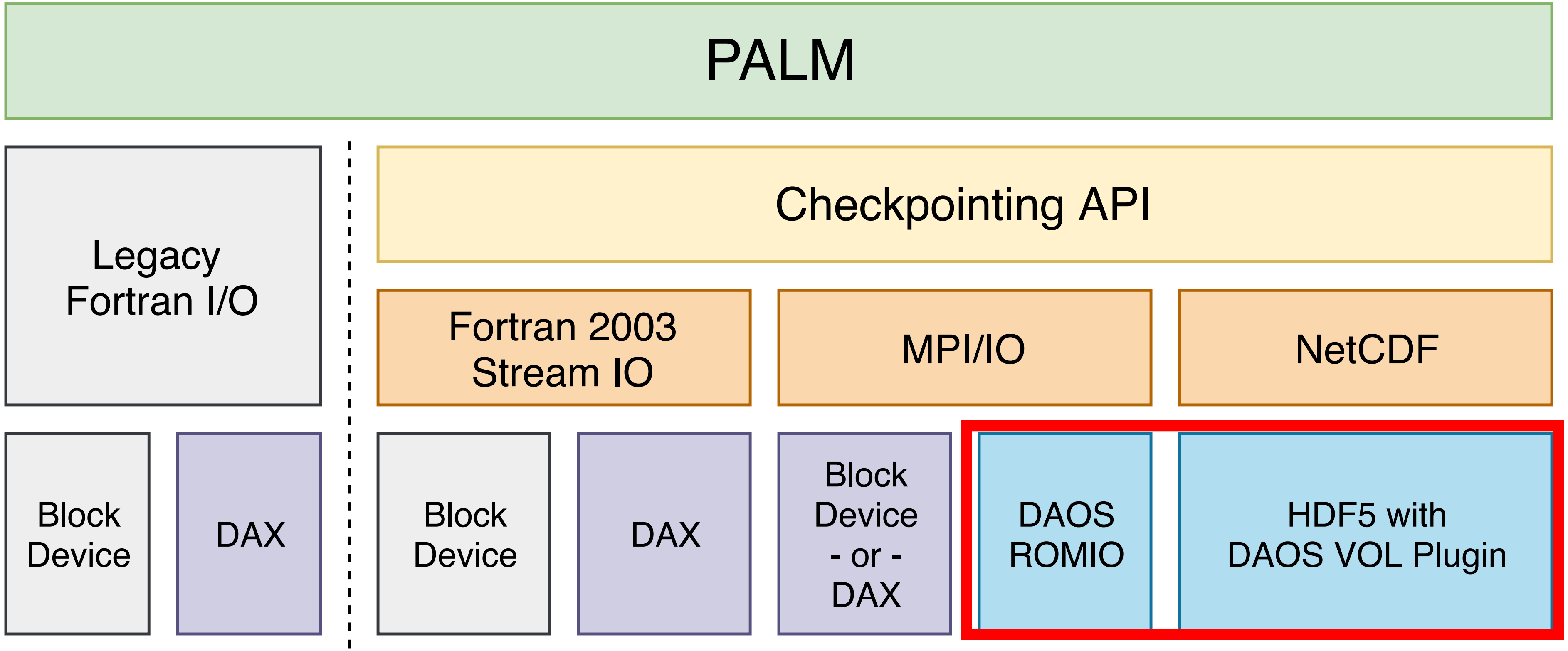
- Design a new checkpointing abstraction layer

```
!-- optional call to define variable
CALL checkpoint_define(’topography’, topography, dt_replicated)
CALL checkpoint_write(’topography’, topography, dt_replicated)
CALL checkpoint_read(’topography’, topography, dt_replicated)
```

- Enables to abstract from actual storage API
  - Remove Fortran unformatted IO
  - Use of generic functions
  - Replace storage backend with modern technologies without application changes

New Checkpointing Abstraction Layer

- Access target hardware via **high-level libraries**
  - **ROMIO:** implementation of MPI’s IO chapter
  - **netCDF:** IO for named and typed arrays; self-described files, existing ecosystem
- Target hardware: **NVRAM**, via DAX or **DAOS**



Using NVRAM + DAOS

Motivation recap: use high level libraries to interface DAOS

stream

- **no interface to DAOS**, maybe FUSE → not investigated
- use NVRAM via **DAX-mounted file system**

MPI/IO

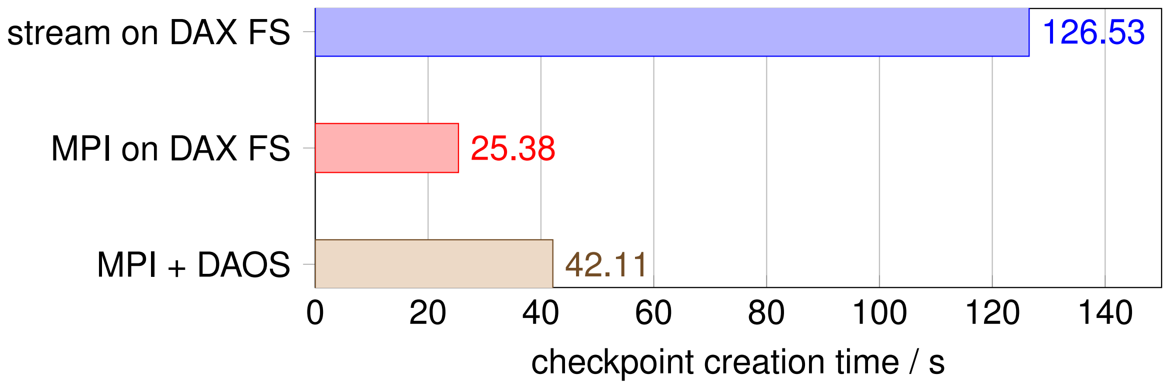
- **interfaces DAOS’** low level API
- ROMIO implementation from DAOS team, based on MPICH 3.3
- `configure, make, make install` and **just run your MPI IO code**
- (some intended limitations; do not apply to PALM)
- alternative: use **DAX-mounted file system**

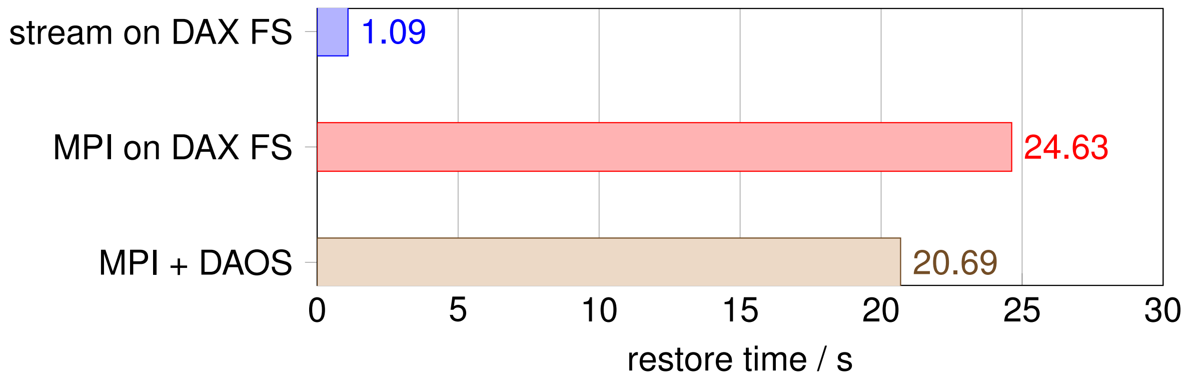
netCDF

- our assumption: just built netCDF on top of DAOS-aware HDF5
- HDF5 VOL plugin for DAOS is work in progress,
  available soon, netCDF likely to need adjustments (?)
- *DAOS v0.6 was not usable via netCDF*

Preliminary Results: DAOS Testbed

- Single node:
  - dual Xeon Platinum 8260L (CL-SP, 24C/48T)
  - 6 TB NVRAM (2 × 6 DCPMM) + 768 GB DRAM
  - CentOS 7, gcc/gfortran 9.1
  - 32 MPI procs, domain size = 4096 × 4096 × 256
- Mimics "DAOS on every compute node" scenario (vs. burst buffer-like setup)
- Compare backend stream, MPI on DAX FS, and MPI on top of DAOS





Experiences with DAOS v0.6

- Installation:
  - Scons
  - use "download dependencies" feature
- Setup:
  - currently cannot use all DCPMMs on NUMA system
  - sometimes confusing configuration:
    - Values for unimplemented features in examples
    - Leave defaults where unsure
    - Sufficient to define what contributes to pool
  - notice *immutable* notes
- Usage:
  - Permissions: required to use DAOS account to get access to pool
  - (1+x) threads per target, i.e. DCPMM, hog CPU cores → energy consumption!?
  - `first_core` option helpful

Experiences with DAOS v0.6

- **Installation** smoothly with Scons, use "download dependencies" feature
- **Setup**:
  - currently cannot use all DCPMMs on NUMA system
  - notice immutable notes
- **Usage:**
  - `first_core` option helpful

- **Setup**: sometimes confusing configuration:
  - Values for unimplemented features in examples
  - Leave defaults where unsure
  - Sufficient to define what contributes to pool
- **Usage:**
  - Permissions: required to use DAOS account to get access to pool
  - (1+x) threads per target, i.e. DCPMM, hog CPU cores → energy consumption!?

Summary

- Avoid raw (POSIX) IO, although it seems to be easy.
- Use established high-level libraries.
- DAOS is under heavy development! Expect some trouble, give to feedback developers!
- Early DAOS performance numbers promising but room for improvements.

Thank You!

Questions:

Steffen Christgau <christgau@zib.de>